Auro Finance

Audit Report

MOVEBIT

contact@bitslab.xyz

https://twitter.com/movebit_

Tue Apr 29 2025

Auro Finance Audit Report

1 Executive Summary

1.1 Project Information

Description	Auro Finance is an advanced Collateralized Debt Position (CDP) protocol that enables users to access liquidity using their crypto assets as collateral. Through Auro Finance, users can borrow USDA, the platform's native stablecoin, against multiple collateral options, allowing them to utilize their holdings without selling them
Type	DeFi
Auditors	MoveBit
Timeline	Tue Apr 15 2025 - Tue Apr 29 2025
Languages	Move
Platform	Aptos
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/Auro-Finance/core
Commits	1b7731fbdc0812ce0ce6c1e32773ed558234b9cf 589d8a7aff48cae61f424f83fae7afb3304da959

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MCH	core/sources/master_chef.move	a6433d44c8d33b7148a0c911ca3e8261b8cc74b2
EPO	core/sources/epoch.move	278665c994da835b7ea219214f5008c3d5a16587
CWR	core/sources/coin_wrapper.move	b577d333f9e85bdbe1a8ff9f62e57bfc1be1110f
ARE	core/sources/auro_rewards.move	914c95cf328fbb130238d7f5df0478382cb539c1
APO	core/sources/auro_pool.move	697686f67247c977b7e3261a93f884b5125a519c
GST	core/sources/global_state.move	124a73b88127160cb5886dd13f456781dd9fdf58
DAO	core/sources/dao.move	bbb5238fe42a39deb6b6b97a3a06630fc4995b37
ATO	core/sources/auro_token.move	d7e62402aeb51689ce4993574c702e8ca993a311
CON	core/sources/convert.move	dca4490c5277c316fcd45d370211390b8f728ed7
UTO	core/sources/usda_token.move	ce2d9dec1cff4779d6f769b466f5e0df155602b2
MOV	core/Move.toml	38356dee3beecc0727ce146eda9bb6523348c75f

RPO	core/sources/rewards_pool.move	e7057676e02c2ea10e29b3c61347c4a4a7fb3ff8
ORA	core/sources/oracle.move	1bb33ad8d693aa45369c848e473651c76150b18c
LPO	core/sources/lending_pool.move	057ae4e545fe68d7ecb21406cc8320deeff0cc91

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	8	8	0
Informational	3	3	0
Minor	1	1	0
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 Auro Finance to identify any potential issues and vulnerabilities in the source code of the Auro Finance 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
ATO-1	Rename LIQUIIDITY_ALLOCATION to LIQUIDITY_ALLOCATION	Informational	Fixed
LPO-1	Rename `last_depsited_times` to `last_deposited_times`	Informational	Fixed
ORA-1	The Default token Price Expiration time is too Long	Medium	Fixed
ORA-2	The case where the Expo is Positive is not Taken into Account	Medium	Fixed
ORA-3	The Price Calculation is Incorrect when `expo` is Positive	Medium	Fixed
RPO-1	The User may not Receive any Rewards	Major	Fixed
RPO-2	Duplicate Check and Setting	Minor	Fixed
RPO-3	Redundant Conditional Check in `unstake` Function	Informational	Fixed

3 Participant Process

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

Admin

- `withdraw_fees` : Collect protocol fees from the pool.
- `set_pause` : Toggle pause state of the pool.
- `set_limits` : Update supply/borrow caps.
- `set_ltv` : Modify Loan-to-Value ratios.
- `set_liquidation_threshold` : Adjust liquidation thresholds.
- `set_liquidation_config` : Configure liquidation parameters.
- `set_interest_rate` : Update interest rate model.
- `set_fees` : Set borrowing/protocol fees.
- `set_lock_times` : Modify withdrawal lock periods.
- `set_early_withdraw_fee_bps` : Set early withdrawal penalty.
- `create_pool` : Initialize a new liquidity pool with specified LP token and parameters.
- `set_pool_locked_time` : Update the locked period for a specific pool.
- `set_max_age_secs` : Set maximum valid duration for price data validity.
- `set_pyth_id` : Bind token to Pyth oracle price feed identifier.
- `whitelist_address` : Manage address whitelisting status for protocol access.
- `pause` : Toggle global pause state of the exchange protocol.
- `set_public` : Controls whether the protocol is only open to the whitelist.
- `whitelist_fungible_asset` : Add fungible asset to protocol's reward token whitelist.
- `whitelist_coin` : Whitelist native coin type by creating a wrapper asset.
- `set_pool_alloc_point` : Adjust reward distribution weights for liquidity pools.
- `pool_incentive_coin_entry` : Deposit native coins as incentives for the specified reward pool.

- `pool_incentive_fa_entry` : Deposit fungible assets as incentives for the specified reward pool.

User

- `deposit` : Stake assets in the lending pool.
- `withdraw` : Withdraw assets (may incur early fee).
- `borrow` : Take out a loan from the pool.
- `repay` : Repay borrowed assets.
- `liquidate` : Liquidate undercollateralized positions.
- `create_position_if_not_exists` : Initialize user position.
- `remove_position` : Delete empty position (requires no deposits/debts).
- `claim_rewards` : Claim accumulated rewards from a specified pool.
- `deposit` (masterchef): Stake LP tokens into a pool.
- `request_withdraw` (masterchef): Initiate a withdrawal request for staked LP tokens.
- `withdraw` (masterchef): Execute withdrawal of unstaked LP tokens.
- `position_claim_rewards` : Claim accumulated rewards from the Auro farming position.
- `masterchef_claim_rewards` : Claim farming rewards from the MasterChef pool.
- `advance_epoch` : Trigger epoch transition to distribute pending emissions.

4 Findings

ATO-1 Rename LIQUIIDITY_ALLOCATION to LIQUIDITY_ALLOCATION

Severity: Informational

Status: Fixed

Code Location:

core/sources/auro_token.move#35

Descriptions:

The issue in the provided code lies in the `LIQUIIDITY_ALLOCATION` constant, which has a typo in its name. It should be `LIQUIDITY_ALLOCATION`.

```
const LIQUIIDITY_ALLOCATION: u64 = 50_000_00_000_000;

primary_fungible_store::deposit(liquidity, mint(LIQUIIDITY_ALLOCATION));
```

Suggestion:

It is recommended to rename `LIQUIIDITY_ALLOCATION` to `LIQUIDITY_ALLOCATION`.

Resolution:

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

LPO-1 Rename `last_depsited_times` to `last_deposited_times`

Severity: Informational

Status: Fixed

Code Location:

core/sources/lending_pool.move#138

Descriptions:

In the `UserPosition` struct, the field name `last_depsited_times` appears to be a typo. It should likely be `last_deposited_times`.

```
struct UserPosition has key, copy, drop {
    emode_id: String,
    deposit_shares: SimpleMap<Object<LendingPool>, u128>,
    debt_shares: SimpleMap<Object<LendingPool>, u128>,
    last_depsited_times: u64,
}
```

Suggestion:

It is recommended to rename `last_depsited_times` to `last_deposited_times`.

Resolution:

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

ORA-1 The Default token Price Expiration time is too Long

Severity: Medium

Status: Fixed

Code Location:

core/sources/oracle.move#23

Descriptions:

In the oracle contract, the default price expiration age is set to 120 seconds.

```
const INITIAL_MAX_AGE_SECS: u64 = 120;
```

However, for some high-traffic tokens, a 120-second expiration time can result in significant price discrepancies.

Suggestion:

It is recommended to set the expiration time to **30 seconds or less** for such tokens to ensure more accurate and up-to-date pricing.

Resolution:

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

ORA-2 The case where the Expo is Positive is not Taken into Account

Severity: Medium

Status: Fixed

Code Location:

core/sources/oracle.move#109-118

Descriptions:

In the `get_pyth_price()` function, the protocol currently only considers the case where the exponent (`expo`) is negative, but it does not account for the case where the exponent is positive.

If the exponent is positive, the price should be calculated as follows:

```
price = raw_price * math128::pow(10, (i64::get_magnitude_if_positive(&i64_expo) as u128)) / PRICE_PRECISION
```

Suggestion:

It is recommended to get the price based on the sign of `expo` .

Resolution:

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

ORA-3 The Price Calculation is Incorrect when `expo` is Positive

Severity: Medium

Status: Fixed

Code Location:

core/sources/oracle.move#112-115

Descriptions:

The function `get_pyth_price()` is used to fetch the price from Pyth. The current explanation is incorrect. When `expo` is negative, the price is calculated as:

```
price = raw_price * PRICE_PRECISION / 10^abs(expo)
```

```
    if(i64::get_is_negative(&expo)){
        math128::mul_div(
          (raw_price as u128),
          PRICE_PRECISION,
          math128::pow(10, (i64::get_magnitude_if_negative(&expo) as u128)),
        )
      }
```

However, when `expo` is positive, the correct formula should be:

```
price = raw_price * 10^abs(expo) / PRICE_PRECISION
```

```
else{
        math128::mul_div(
          (raw_price as u128),
          PRICE_PRECISION,
          math128::pow(10, (i64::get_magnitude_if_positive(&expo) as u128)),
        )
      }
```

So the handling of positive `expo` needs to be corrected accordingly.

Suggestion:

It is recommended to calculate the price as `price = raw_price * expo / PRICE_PRECISION` when expo is positive.

Resolution:

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

RPO-1 The User may not Receive any Rewards

Severity: Major

Status: Fixed

Code Location:

core/sources/rewards_pool.move#573

Descriptions:

In the `add_rewards_current_epoch()` function, the `reward_per_sec` is multiplied by a constant called `ACCUM_REWARD_SCALE`, which is currently set to `1e8`.

```
    if (new_finish_time > timestamp::now_seconds()) {
        *reward_duration = new_finish_time - timestamp::now_seconds() ;
        *reward_period_finish = new_finish_time;
    };
    *reward_per_sec = math128::mul_div(
        (remaining + reward_amount),
        ACCUM_REWARD_SCALE,
        (*reward_duration as u128)
    );
```

In the `update_pool_per_token()` function, the `acc_token_per_share` is calculated as `acc_token_per_share = generate_reward / total_supply`:

```
let generate_reward =
    get_generate_reward(
        (reward_info.last_reward_time as u128),
        (timestamp::now_seconds() as u128),
        reward_info
    );

let acc_token_per_share = &mut reward_info.acc_token_per_share;
*acc_token_per_share += generate_reward / total_supply;
if (timestamp::now_seconds() > reward_info.reward_period_finish) {
```

```
        reward_info.reward_per_sec = 0;
    };
```

where `generate_reward = (to - from) * reward_per_sec`.

There is an issue here: if the time duration (`to - from`) is only 1 second, and the `total_supply` is a large number (with 8 decimal precision and continuously increasing), then `generate_reward / total_supply` may become zero due to integer division. As a result, `last_reward_time` is updated, but no reward is actually accounted for.

Suggestion:

It is recommended to consider increasing the precision of `ACCUM_REWARD_SCALE`

Resolution:

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

RPO-2 Duplicate Check and Setting

Severity: Minor

Status: Fixed

Code Location:

core/sources/rewards_pool.move#584;

core/sources/rewards_pool.move#360

Descriptions:

In the check `if (reward_info.last_reward_time >= timestamp::now_seconds() || reward_info.reward_per_sec == 0) return;` , it is ensured that `to>from` , and in the `assert` check, the lock time condition is checked.

```
 assert!(
        user_info.requested_withdraw_time + pool_info.locked_time <=
timestamp::now_seconds(),
        ERR_REQUEST_WITHDRAW_TIME_INVALID
    );
    if (user_info.requested_withdraw_time + pool_info.locked_time <=
timestamp::now_seconds()) {
        assert!(user_info.requested_withdraw_amount == amount,
ERR_INVALID_AMOUNT);
        user_info.requested_withdraw_time = 0;
        user_info.requested_withdraw_amount = 0;
    }
...
fun get_generate_reward(from: u128, to: u128, reward_info: & RewardsTokenInfo): u128 {
    let pool_end_time = (reward_info.reward_period_finish as u128);
    let last_reward_time = (reward_info.last_reward_time as u128);
    let reward_per_sec = reward_info.reward_per_sec;
    if (from >= to) return 0;
```

`epoch_rewards.is_distribution = true;` Repeat the setting.

```
assert!(epoch_rewards.is_distribution == false, EINCENTIVES_ALREADY_DISTRIBUTED);
epoch_rewards.is_distribution = true;
let total_amount = &epoch_rewards.total_amounts;
let tokens = &total_amount.keys();
```

```
tokens.for_each_ref(|token|{
    let amount = *total_amount.borrow(token);
    let asset = withdraw_from_pool(&incentive_data.extend_ref, *token, amount);
    add_rewards_current_epoch(pool, asset);
});
event::emit(RewardsDistributed {
    pool,
    epoch,
    timestamp: timestamp::now_seconds()
});
epoch_rewards.is_distribution = true;
```

Suggestion:

It is recommended to remove unnecessary and duplicate checks.

Resolution:

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

RPO-3 Redundant Conditional Check in `unstake` Function

Severity: Informational

Status: Fixed

Code Location:

core/sources/rewards_pool.move#344-366

Descriptions:

The following `if` condition is redundant:

```
if (user_info.requested_withdraw_time + pool_info.locked_time <=
timestamp::now_seconds()) {
```

This condition is already checked by the `assert!` statement immediately before it. Since the `assert!` will revert the transaction if the condition is false, the `if` block will only be executed if the condition is true, making the explicit `if` check unnecessary.

Suggestion:

Remove the redundant `if` condition and directly execute the logic inside it:

```
assert!(
    user_info.requested_withdraw_time + pool_info.locked_time <=
timestamp::now_seconds(),
    ERR_REQUEST_WITHDRAW_TIME_INVALID
);

assert!(user_info.requested_withdraw_amount == amount, ERR_INVALID_AMOUNT);
user_info.requested_withdraw_time = 0;
user_info.requested_withdraw_amount = 0;
```

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.