Thala Labs

Audit

Presented by:

OtterSec contact@osec.io

Robert Chen notdeghost@osec.io

Akash Gurugunti sud0u53r.ak@osec.io

Ajay Kunapareddy d1r3wolf@osec.io

Contents

01 | Executive Summary

Overview

Thala Labs engaged OtterSec to perform an assessment of the `thala-modules` program. This assessment of the source code was conducted between May 22nd and May 31st, 2023. For more information on our auditing methodology, see Appendix B.

Critical vulnerabilities were communicated to the team prior to the delivery of the report to speed up remediation. After delivering our audit report, we worked closely with the team to streamline patches and confirm remediation.

Key Findings

During the audit engagement, we identified a total of 24 findings.

Specifically, we discovered several precision loss issues in arithmetic functions that resulted in unintended behaviours (OS-TLA-ADV-06), improper calculation formulas (OS-TLA-ADV-07, OS-TLA-ADV-10), and issues with interest and CR calculations (OS-TLA-ADV-02, OS-TLA-ADV-08, OS-TLA-ADV-09).

In addition, we provided recommendations for improving the LBP design (OS-TLA-SUG-02), validating edge cases (OS-TLA-SUG-06, OS-TLA-SUG-07), and implementing rounding directions for arithmetic operations (OS-TLA-SUG-03).

Overall, we commend the Thala Labs team for being responsive and knowledgeable throughout the audit.

02 | Scope

The source code was delivered to us in a git repository github.com/ThalaLabs/thala-modules. This audit was performed against commit dbefe09.

A brief description of the programs is as follows.

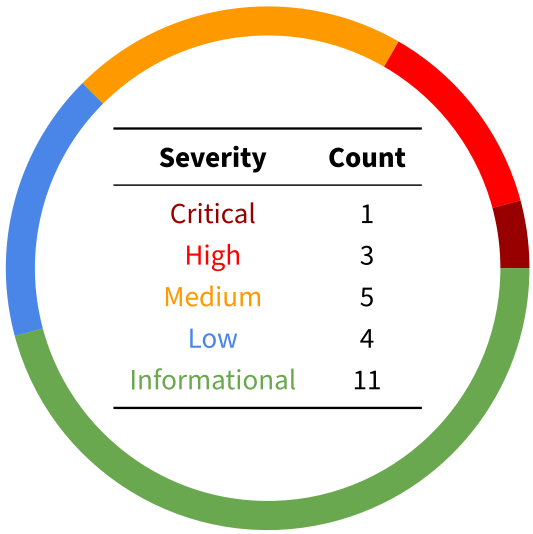
Name	Description
thala_farming	A farming protocol where users stake coins and earn rewards.
thala_launch	A liquidity boosting pool to increase the liquidity of a particular token or asset.
thala_manager	A common utility module for managing other Thala modules.
thala_oracle	A two-tier oracle implemented using Pyth and Switchboard oracles.
thala_protocol	An over-collateralized stablecoin.
thalaswap	A multi-asset stable pool for stablecoins and weighted pools.
thalaswap_math	A math utility module for stable and weighted pools.
thl_vesting	A simple linear vesting contract for THL tokens.
vetoken	A token vesting contract with dividend distributions for the vested tokens.

Note that `vetoken` was later moved out into a separate repository, github.com/ThalaLabs/vetoken.

03 | Findings

Overall, we report 24 findings.

We split the findings into **vulnerabilities** and **general findings**. Vulnerabilities have an immediate impact and should be remediated as soon as possible. General findings do not have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	1
High	3
Medium	5
Low	4
Informational	11

04 | Vulnerabilities

Here, we present a technical analysis of the vulnerabilities we identified during our audit. These vulnerabilities have *immediate* security implications, and we recommend remediation as soon as possible.

Rating criteria may be found in Appendix A.

ID	Severity	Status	Description
OS-TLA-ADV-00	Critical	Resolved	Inaccurate accumulator updates result in the incorrect calculation of additional rewards.
OS-TLA-ADV-01	High	Resolved	Improper accumulator updates result in incorrect calculation of user rewards.
OS-TLA-ADV-02	High	Resolved	Repaid interest should be deducted from the vault.
OS-TLA-ADV-03	High	Resolved	Incorrect calculations of rewards during distribution may result in certain users receiving reduced rewards.
OS-TLA-ADV-04	Medium	Resolved	Denial of service while vesting in `thl_vesting` due to improper usage of claim ID.
OS-TLA-ADV-05	Medium	Resolved	Issue while starting a new epoch by ending the current epoch.
OS-TLA-ADV-06	Medium	Resolved	Precision loss issue in weighted math leads to a loss of funds while swapping small amounts.
OS-TLA-ADV-07	Medium	Resolved	Improper price deviation calculation formula in oracle.
OS-TLA-ADV-08	Medium	Resolved	Interest is included while calculating the collateral ratio of a vault.
OS-TLA-ADV-09	Low	Resolved	Improper interest accumulation calculation while updating the global interest index.
OS-TLA-ADV-10	Low	Resolved	Improper withdrawal fee calculation formula in the stability pool leads to the incentivization of early withdrawals.

OS-TLA-ADV-11	Low	Resolved	Improper implementation of `withdraw_all` may lead to the creator removing all liquidity from live LBP.
OS-TLA-ADV-12	Low	Resolved	Improper implementation of the mint cap check in `borrow`.

OS-TLA-ADV-00 [crit] | Improper Accumulator Updates

Description

stake and unstake update parameters for thl coin rewards, which are also affected by the stake_amount. As a result, altering the stake amount may cause incorrect calculations of extra rewards.

A malicious user may exploit this vulnerability and take out a flash loan to increase their stake_amount, enabling them to collect rewards for the newly added stake.

Proof of Concept

1. The manager adds more reward coins to the farming protocol.
2. A user claims additional rewards from the farm, which updates the global accumulator based on the current pool stake value.
3. An attacker stakes a large amount and attempts to claim the reward.
4. Due to the failure to update user_pool_info.last_acc_rewards_per_share for extra rewards before staking and unstaking, the attacker may claim the entire pool balance by calling claim_extra_reward.

Remediation

stake and unstake should first update the accumulator for extra rewards using claim_extra_reward before modifying the stake amount. Creating a vector to store the names of all additional reward coins and using them in the claim function is a way to go about it.

Patch

Fixed in 2693e34 by adding an additional field to the PoolInfo structure for storing the amounts of extra reward coins. These amounts are updated during the staking and unstaking processes.

OS-TLA-ADV-01 [high] | Improper Accumulator Updates In V2 Mode

Description

The `stake` and `unstake` functions update the stake amount of the user. These functions also calculate the amount of rewards accrued until that time and store it, and then update the accumulator on the user pool.

In the recent changes introducing v2 mode for farming, when v2 mode is enabled, the `thl` rewards for a user are not accrued in `stake` and `unstake`; this results in improper rewards for users.

Remediation

`stake` and `unstake` should first update the accumulator for `thl` rewards using `accrue_user_reward` before modifying the stake amount.

Patch

Fixed in e55c601 by calling the `accrue_user_reward` function call when v2 mode is enabled.

OS-TLA-ADV-02 [high] | Deducting Vault Interest When Repaying Debt

Description

In the protocol module, `repay_internal` is used to repay amounts borrowed from the vault. In addition to the debt, clearing the interest should be done when repaying the borrowed amount.

Although the protocol uses `fees::absorb_fee` to calculate and absorb the repaid interest amount, the protocol does not subtract this amount from `vault.interest`. Consequently, a user is unable to clear the interest in their vault, even though the protocol absorbs it from the repayment amount

Remediation

Subtract `repay_interest_amount` from the `vault.interest`.

Patch

Fixed in 48f7c83 by subtracting `repay_interest_amount` from the `vault.interest`.

OS-TLA-ADV-03 [high] | Improper Reward Calculations

Description

In the protocol module, `accumulated_gain` calculates the earnings of a token solely based on the scale of the snapshot. However, a user's amount may have participated in the distribution of the subsequent scale as well. As a result, the failure to account for this may lead to incorrect calculations of token earnings.

Proof of Concept

thala-protocol-v1/sources/reward_distributor.move RUST

```rust
use std::debug;

#[test(account = @0xA)]
fun test_gain(account: &signer) {
    // prepare
    let rd = new();
    let mut_rd = &mut rd;

    deposit(mut_rd, @0x1, 10000); distribute<ETH>(mut_rd, 9999, 500);
    // debug::print(&mut_rd.current_scale); // Out: 0

    deposit(mut_rd, @0x2, 10000); distribute<ETH>(mut_rd, 8999, 450);
    // debug::print(&mut_rd.current_scale); // Out: 1

    deposit(mut_rd, @0x1, 9000); distribute<ETH>(mut_rd, 9999, 500);
    // debug::print(&mut_rd.current_scale); // Out: 2

    let d1 = account_deposit(mut_rd, @0x1);
    withdraw(mut_rd, @0x1, d1);
    let e1 = claim<ETH>(mut_rd, @0x1);
    debug::print(&d1);
    debug::print(&e1);

    let d2 = account_deposit(mut_rd, @0x2);
    withdraw(mut_rd, @0x2, d2);
    let e2 = claim<ETH>(mut_rd, @0x2);
    debug::print(&d2);
    debug::print(&e2); // Err: This has to be 500, but the output is 450

    // cleanup
    move_to(account, RewardDistributorHolder { reward_distributor: rd });
}
```

Remediation

Increase the scale factor and include two consecutive scales in the earnings calculation in the accumulated gain. This will ensure that the function takes into account all relevant factors for accurate calculations of token earnings.

Patch

Fixed in bdfabae by increasing scaling factor and improving reward calculation.

OS-TLA-ADV-04 [med] | Denial Of Service While Vesting

Description

`start_vesting` in `thl_vesting/farming_vesting.move` initiates vesting when called by a user to begin vesting. It uses a smart table on `vesting.claims` to store the claims of the users.

The length of the `vesting.claims` smart table is the claim ID for the current vesting. Claiming a claim removes the entry at the claim ID from the smart table, resulting in a decrease in the length of the smart table. Consequently, attempting to add a new claim causes a denial of service issue.

Proof of Concept

1. A user (userA) begins vesting when the length of the claims table is 10 and claims[10] will be userA's record and the length of the smart table becomes 11.
2. Now, if one of the previous claims got claimed, it will reduce the length of the smart table to 10.
3. Now, if another user (userB) attempts to start vesting, it reverts because it tries to add the claim in claims[10] although userA's claim record already exists at 10.

Remediation

Track the index using a structure field stored on `Vesting` and increment it every time a record is added to the smart table.

Patch

Fixed in af0d186 by storing the next claim id in `Vesting.next_claim_id` and incrementing it for every new claim.

OS-TLA-ADV-05 [med] | Issue While Starting New Epoch In Farming

Description

end_epoch in thala_protocol/farming.move ends the currently running epoch in-order to start a new epoch.

If an epoch ends earlier than epoch_end_seconds, the code sets farming.epoch_end_seconds to epoch_now. If the current epoch's starting time is in the future and ending this epoch is attempted, farming.epoch_end_seconds is set to farming.epoch_start_seconds, which is the future time even though the epoch has ended immediately. Now, starting another epoch is impossible until we reach farming.epoch_start_seconds.

Remediation

Set farming.epoch_end_seconds to timestamp::now_seconds() instead, if the epoch ended earlier than epoch_end_seconds.

Patch

Fixed in af5fc4a by setting farming.epoch_end_seconds to timestamp::now_seconds().

OS-TLA-ADV-06 [med] | Precision Loss Issue In Weighted Math

Description

The math module calculates the amount taken in during a swap based on the amount given out the balances in the pool and the weights of the assets. `calc_in_given_out_internal` is responsible for this calculation, which involves using `log_exp_math::pow` to perform the required exponentiation.

`log_exp_math::pow` used by `calc_in_given_out_internal` is vulnerable to precision errors, which may return incorrect values. For instance, the function may incorrectly calculate `1.0000000002 ** 1 = 1.0`. This precision issue may be exploited in `calc_in_given_out_internal`, leading to a return value of zero despite a non-zero `amount_out` value.

Proof of Concept

1. Assume bI = bO = 10000000000 ($10,000) and weights of the assets, wI = wO = 50.
2. When attempting to calculate the `amount_in` for 100 as the `amount_out`, the returned value is 97, whereas the expected value is 10.
3. The loss of precision becomes more significant as the asset balances increase.
4. Let's assume bI = bO = 1000000000000 ($1,000,000) and wI = wO = 50.
5. If the `amount_out` is set to 200, the returned value is zero, indicating that we may perform a swap by giving zero tokens and receiving 200 tokens in return.

Remediation

Improve the precision of `log_exp_math::pow` and set the return value to x when `log_exp_math::pow(x, 1)` is called.

Additionally, ensure that the invariant value does not decrease after a swap by asserting its non-decreasing nature, helping to avoid any rounding problems that may lead to a loss of value in the pool.

Patch

Fixed in 572dabb by improving precision in `log_exp_math::pow` and short-circuiting the return value to x when `log_exp_math::pow(x, 1)` is called.

OS-TLA-ADV-07 [med]| Improper Price Deviation Calculation Formula

Description

`get_price_diff_` is responsible for computing price deviation. However, to calculate the percentage of price deviation, the formula should be `(diff(new_price, old_price) / old_price) * 100`. The current implementation uses `new_price` as the denominator if `new_price > old_price`.

thala-protocol-v1/sources/oracle.move RUST

```rust
// Get the difference between two prices a and b in percentage. Result is
    ↪  rounded to nearest integer
fun get_price_diff_pct(a: FixedPoint64, b: FixedPoint64): u64 {
    if (fp64::gt(&a, &b)) {
        fp64::decode(fp64::mul(fp64::div_fp(fp64::sub_fp(a, b), a), 100))
    } else if (fixed_point64::lt(&a, &b)) {
        fp64::decode(fp64::mul(fp64::div_fp(fp64::sub_fp(b, a), b), 100))
    } else {
        0
    }
}
```

Remediation

Use b (`old_price`) as the denominator in both cases.

Patch

Fixed in bd1e275 by using b as the denominator in both cases.

OS-TLA-ADV-08 [med] | Including Interest In Vault CR Calculation

Description

redeem_collateral and liquidate calculate the collateral ratio (CR) for a vault, used in redemption and liquidation calculations. However, these functions do not account for the updated interest of the vault when calculating the CR.

As a result, the CR is calculated without considering the vault.interest, leading to the use of an incorrect CR value in other calculations.

Remediation

redeem_collateral and liquidate should be updated to consider the updated interest of the vault when calculating the collateral ratio (CR). Specifically, vault.interest should be updated using accrue_vault_interest just before calculating the CR in redeem_collateral.

In addition, the updated vault.interest should be taken into account when calculating the CR for a vault in both redeem_collateral and liquidate.

Patch

Fixed in 108cd74 by using vault_liability_amount, which returns the total liability of the vault, i.e., vault.debt + vault.interest.

OS-TLA-ADV-09 [low] | Improper Interest Accumulation Calculation

Description

`sync_interest_rate` derives the value of `days_elapsed` by dividing `seconds_elapsed` by the number of seconds in a day. This calculation truncates any remaining seconds, which may cause the value of `days_elapsed` to be rounded down.

As a result, if `seconds_elapsed` is equal to 1 day, 23 hours, and 59 minutes, the value of `days_elapsed` would be rounded down to one. Then, the new interest index would be calculated for only one day, and the last updated timestamp would be incorrect by 23 hours and 59 minutes. Consequently, the global interest index ratio and interest on vaults may be lower than expected.

Remediation

Increment `interest_last_update_seconds` with `days_elapsed * SECONDS_IN_DAY` instead of directly setting it to the current timestamp.

Patch

Fixed in #105.

OS-TLA-ADV-10 [low] | Improper Withdrawal Fee Calculation Formula

Description

`withdraw_mod` handles the withdrawal fee calculation. To determine the withdrawal fee amount, the intended formula should utilize the `(1 - (elapsed_time / withdrawal_fee_period)) * withdrawal_fee_max_ratio` formula, which decreases the withdrawal fee proportionally over time.

However, in the implemented formula, the `cover_ratio` is not subtracted from one when calculating the `fee_ratio` value.

As a result, users who withdraw shortly after depositing may encounter almost zero withdrawal fees, while those who withdraw just before the withdrawal period may face maximum withdrawal fees.

thala_protocol/sources/stability_pool.move RUST

```rust
let withdrawal_fee_amount =
    if (elapsed_time_seconds >= params.withdrawal_fee_period_seconds) 0
    else if (elapsed_time_seconds == 0) fixed_point64::decode(
        fixed_point64::mul(params.withdrawal_fee_max_ratio, amount)
    )
    else {
        let cover_ratio = fixed_point64::fraction(elapsed_time_seconds,
    ↪   params.withdrawal_fee_period_seconds);
        let fee_ratio = fixed_point64::mul_fp(cover_ratio,
    ↪   params.withdrawal_fee_max_ratio);
        fixed_point64::decode(fixed_point64::mul(fee_ratio, amount))
    };
```

Remediation

Use the correct formula for calculating `fee_ratio`.
`fee_ratio = (1 - cover_ratio) * withdrawal_fee_max_ratio`.

Patch

Fixed in dded61d by implementing the correct formula.

OS-TLA-ADV-11 [low] | Improper Implementation Of Withdraw All

Description

A pool creator may remove liquidity from the pool by specifying a percentage to remove in bps format. However, if the LBP is still ongoing, only a portion of the liquidity may be removed from the pool, with the remaining liquidity being only removable once the LBP has concluded.

This is implemented through a check of the `remove_bps` variable to see if it is equal to `BPS_BASE`. If this check evaluates as true, an assertion is made to ensure that the LBP has indeed ended before allowing for complete liquidity removal.

This may be easily bypassed by passing in `remove_bps = 9999` and calling the `remove_liquidity` multiple times. This removes all the liquidity from the pool even when the LBP has not ended.

thala-launchpad/sources/lbp.move RUST

```rust
let withdraw_all = remove_bps == BPS_BASE;
if (withdraw_all) {
    assert!(lbp_ended(lbp), ERR_LBP_NOT_ENDED)
};
```

Remediation

Assert `amount_0 != balance_0` && `amount_1 != balance_1` if `withdraw_all` is false.

Patch

Fixed in cd724d9 by removing `withdraw_all` and allowing the creator to remove any amount of liquidity from their pool.

OS-TLA-ADV-12 [low] | Improper Implementation Of Mint Cap Check

Description

The `mint_cap` parameter set by the protocol limits the number of mintable MOD tokens. The limit is checked by confirming that the newly minted amount alongside the previous total debt does not exceed the `mint_cap`.

But in implementation, the amount considered in this check (`amount`) does not match the amount actually minted (`total_amount = amount + fee_amount`). Hence, this leads to the minting of MOD exceeding the amount set in the `mint_cap` parameter.

Remediation

Use the amount actually minted while enforcing the `mint_cap` constraint.

Patch

Fixed in baa84d1 by using the same amount while checking the `mint_cap` constraint and minting MOD.

05 | General Findings

Here, we present a discussion of general findings during our audit. While these findings do not present an immediate security impact, they represent anti-patterns and may lead to security issues in the future.

ID	Description
OS-TLA-SUG-00	Implementing a setter function to update the Switchboard configuration.
OS-TLA-SUG-01	Configurable simple oracle updater address for ease of use.
OS-TLA-SUG-02	Using the creator address as a key for the LBP collection instead of calculating the hash.
OS-TLA-SUG-03	Implementing functions that specify the direction of rounding for swap calculations
OS-TLA-SUG-04	Potential overflow in a stable pool while adding liquidity to the pool.
OS-TLA-SUG-05	The constraint for penalty multiplier and minimum collateralization ratio.
OS-TLA-SUG-06	Critical checks in weighted math to avoid emptying the pool.
OS-TLA-SUG-07	The edge case in the auction price calculation leads to the improper status of the auction price.
OS-TLA-SUG-08	Specific rounding direction for liability while calculating vault interest.
OS-TLA-SUG-09	Additional fields for events to specify more information about the event taking place.
OS-TLA-SUG-10	A more precise fixed point invariant may be used instead of rounding down for assertion

OS-TLA-SUG-00 | Implementing Set Switchboard Config Function

Description

The oracle module utilizes the Switchboard oracle to obtain coin prices. The aggregator address for a specific coin type is stored in the `SwitchboardConfig<CoinType>` structure and set during the initialization process of the oracle for that coin.

However, once the oracle is initialized, the manager is unable to update the aggregator configuration for a given coin.

Remediation

Implement a `set_switchboard_config` function to enable the manager to update the aggregator address for a coin.

Patch

Fixed in 0e805a2 by implementing a `configure_switchboard` function.

OS-TLA-SUG-01 | Configurable Simple Oracle Updater

Description

The oracle module employs a straightforward oracle that saves the price of a coin in a resource. The price of a coin, frequently updated by a protocol bot, may be utilized as a backup in the event of a failure of the Pyth or Switchboard oracles.

The resource updater, responsible for updating the coin price in the resource, is statically set in the move configuration file and is fixed at compile time.

Remediation

Implement a configurable simple oracle updater address instead. The manager will then be able to update the address in the case of losing the updater private key.

Patch

Fixed in 778e6ee by implementing `configure_simple_oracle`.

OS-TLA-SUG-02 | Using Creator Address As Key For LBPCollection

Description

The launch module stores a list of LBP instances in the `LBPCollection<Asset0, Asset1>` resource using `sha3(asset_0_name + asset_1_name + creator_address)` as the key and LBP as the value.

However, since an `LBPCollection` has unique asset generics for each asset pair, calculating the hash with asset names is unnecessary. Instead, the creator's address may function as the key for the LBP instances.

Remediation

Use the creator address as the key for LBPs in `LBPCollection` and rewrite all functions that use the LBP collection.

Patch

Fixed in f4fb228.

OS-TLA-SUG-03 | Specific Direction Of Rounding For Swap Calculations

Description

The math calculations for swap pools are crucial from a security standpoint, as rounding issues are vulnerable to exploitation, resulting in a loss of funds from the protocol. Functions with a specific rounding direction are imperative in preventing exploits from surfacing.

For instance, the code should round down calculations when calculating the number of tokens given to the user for a specific input token.

The rounding in the formulae should involve rounding down all variables in the formula directly proportional to the output amount while rounding up those inversely proportional. Applying said changes should be done on arithmetic operations such as division and exponentiation.

Potential rounding issues may be avoided by following these guidelines.

Remediation

Implement arithmetic functions with specific directions for rounding and utilize them accordingly.

Patch

Fixed in 951ad53 by including mitigations.

OS-TLA-SUG-04 | Potential Overflow In Stable Pool

Description

There is a potential for the value of the `liquidity` variable to overflow during the execution of `add_liquidity`, if the `total_supply` and the difference between invariants are large enough.

thalaswap/sources/stable_pool.move RUST

```
let total_supply = base_pool::pool_token_supply<StablePoolToken<Asset0,
    ↪  Asset1, Asset2, Asset3>>();
let liquidity = (total_supply * (inv - prev_inv)) / prev_inv;
event::emit_event<AddLiquidityEvent<Asset0, Asset1, Asset2, Asset3>>(
    &mut pool.events.add_liquidity_events,
    AddLiquidityEvent { amount_0, amount_1, amount_2, amount_3,
    ↪  minted_lp_coin_amount: liquidity }
);
```

Remediation

Convert the `total_supply` and `inv - prev_inv` values to `u128` for multiplication and convert them back to `u64` after the division.

Patch

Fixed in 205acb7 by using `u256`.

OS-TLA-SUG-05 | Constraint For Penalty Multiplier And MCR

Description

The value of `penalty_multiplier` and MCR should be set such that if `CR ≤ 1`, then `(penalty_multiplier * (MCR - CR))` should be ≥ 1.

This implies that `penalty_multiplier ≥ 1/(MCR - 1)` in order to assert that the health of the vault increases after the liquidation.

Remediation

Enforce the `penalty_multiplier ≥ 1/(MCR - 1)` constraint while updating the MCR and `penalty_multiplier`.

Patch

Fixed in #118.

OS-TLA-SUG-06 | Critical Checks In Weighted Math

Description

In the math module, `calc_in_given_out_internal` and `calc_out_given_in_internal` are responsible for calculating the amounts to be given in and out.

It is critical to assert that `weight_ratio > 0` in `calc_in_given_out_internal` and `calc_out_given_in_internal` to avoid the cases where `amountIn = balanceIn` is irrespective of `amountOut` and `amountOut = 0` is irrespective of `amountIn`.

Remediation

Enforce the `weight_ratio > 0` constraint in `calc_in_given_out_internal` and `calc_out_given_in_internal`.

Patch

Fixed in #109.

OS-TLA-SUG-07 | Edge Case In Auction Price Calculation

Description

`status` returns a boolean, which determines if the auction requires a reset, along with the current auction price.

With `DEFAULT_RESERVE_RATIO_BPS` being zero, in an edge case where the `elapsed time == expiry_time_seconds`, the calculated `price` and `decrease_ratio` will be zero while the `below_reserve` will be false since `0 < 0 = false`.

Since a denominator in a formula in `bid` uses the returned auction price, a division with zero error will occur and abort. It may be a critical issue if bidding is completed successfully with `auction price = 0`.

Remediation

Change the condition in the `if` statement in `status` from `(elapsed_seconds > params.expiry_time_seconds)` to `(elapsed_seconds >= params.expiry_time_seconds)`.

Patch

Fixed in #120.

OS-TLA-SUG-08 | Specific Rounding Direction For Liability

Description

Updating the vault interest based on the global interest rate uses `accrue_vault_interest`. The newly calculated interest, `new_liability`, is decoded to u64 using `fixed_point64::decode`, which does not specify the direction of rounding.

If rounded down, it may lead to lesser interest rates in vaults.

Remediation

Specify the rounding direction to up by using `fixed_point64::decode_round_up`.

Patch

Fixed in d5c9605.

OS-TLA-SUG-09 | Additional Fields For Events In Oracle

Description

Whenever a value in `OracleParams` changes, the `OracleParamChangeEvent` event emits. However, the event fails to specify the coin name for which it changes the parameters.

Add the `coin_name` as a field in `OracleParamChangeEvent` to specify the coin for which the parameters are changed.

Patch

Fixed in #111.

OS-TLA-SUG-10 | Using Precise Invariant For Assertion

Description

`compute_invariant_weights_u64` is used in the math module to calculate an invariant and assert that it is non-decreasing after the swap. In this function, the calculated invariant is rounded down before returning it, which may result in a loss of precision and frequent reverting of swaps.

It is preferred to return the fixed point value directly and use it for asserting the invariant and decoding it to `u64` only when necessary.

Remediation

Return fixed point values directly for invariant assertion in `calc_out_given_in_weights_u64` and `calc_in_given_out_weights_u64`.

Patch

Fixed in #110.

A | Vulnerability Rating Scale

We rated our findings according to the following scale. Vulnerabilities have immediate security implications. Informational findings can be found in the General Findings section.

Critical	Vulnerabilities that immediately lead to loss of user funds with minimal preconditions Examples: • Misconfigured authority or access control validation • Improperly designed economic incentives leading to loss of funds
High	Vulnerabilities that could lead to loss of user funds but are potentially difficult to exploit. Examples: • Loss of funds requiring specific victim interactions • Exploitation involving high capital requirement with respect to payout
Medium	Vulnerabilities that could lead to denial of service scenarios or degraded usability. Examples: • Malicious input that causes computational limit exhaustion • Forced exceptions in normal user flow
Low	Low probability vulnerabilities which could still be exploitable but require extenuating circumstances or undue risk. Examples: • Oracle manipulation with large capital requirements and multiple transactions
Informational	Best practices to mitigate future security risks. These are classified as general findings. Examples: • Explicit assertion of critical internal invariants • Improved input validation

B | Procedure

As part of our standard auditing procedure, we split our analysis into two main sections: design and implementation.

When auditing the design of a program, we aim to ensure that the overall economic architecture is sound in the context of an on-chain program. In other words, there is no way to steal funds or deny service, ignoring any chain-specific quirks. This usually requires a deep understanding of the program's internal interactions, potential game theory implications, and general on-chain execution primitives.

One example of a design vulnerability would be an on-chain oracle that could be manipulated by flash loans or large deposits. Such a design would generally be unsound regardless of which chain the oracle is deployed on.

On the other hand, auditing the implementation of the program requires a deep understanding of the chain's execution model. While this varies from chain to chain, some common implementation vulnerabilities include reentrancy, account ownership issues, arithmetic overflows, and rounding bugs.

As a general rule of sum, implementation vulnerabilities tend to be more "checklist" style. In contrast, design vulnerabilities require a strong understanding of the underlying system and the various interactions: both with the user and cross-program.

As we approach any new target, we strive to get a comprehensive understanding of the program first. In our audits, we always approach targets with a team of auditors. This allows us to share thoughts and collaborate, picking up on details that the other missed.

While sometimes the line between design and implementation can be blurry, we hope this gives some insight into our auditing procedure and thought process.