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

Appendices

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 January 5th and February 22nd, 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. We delivered final confirmation of the patches March 14th, 2023.

Key Findings

During the audit engagement, we identified a total of 21 findings.

Specifically, we discovered several precision loss issues in the `fixed_point64` arithmetic functions (OS-TLA-ADV-03), improper calculation formulas (OS-TLA-ADV-04, OS-TLA-ADV-07) that resulted in unintended behaviours, and issues with interest and CR calculations (OS-TLA-ADV-01, OS-TLA-ADV-05, OS-TLA-ADV-06).

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 1e2b574.

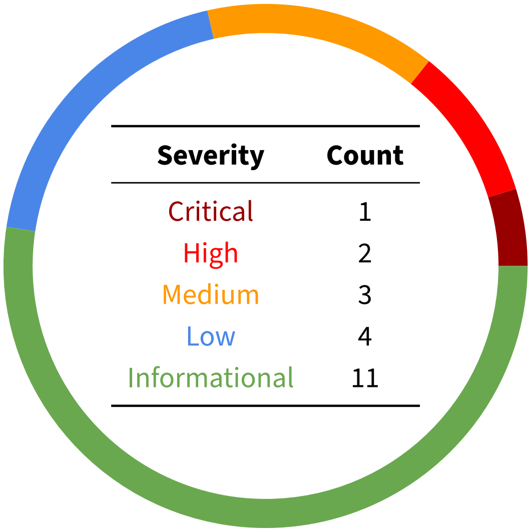
A brief description of the programs is as follows.

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.

03 | Findings

Overall, we report 21 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 don't have an immediate impact but will help mitigate future vulnerabilities.



Severity	Count
Critical	1
High	2
Medium	3
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 can 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	Repaid interest should be deducted from the vault.
OS-TLA-ADV-02	High	Resolved	Incorrect calculations of rewards during distribution may result in certain users receiving reduced rewards.
OS-TLA-ADV-03	Medium	Resolved	Precision loss issue in weighted math leads to a loss of funds while swapping small amounts.
OS-TLA-ADV-04	Medium	Resolved	Improper price deviation calculation formula in oracle.
OS-TLA-ADV-05	Medium	Resolved	Interest is included while calculating the collateral ratio of a vault.
OS-TLA-ADV-06	Low	Resolved	Improper interest accumulation calculation while updating the global interest index.
OS-TLA-ADV-07	Low	Resolved	Improper withdrawal fee calculation formula in the stability pool leads to the incentivization of early withdrawals.
OS-TLA-ADV-08	Low	Resolved	Improper implementation of `withdraw_all` may lead to the creator removing all liquidity from live LBP.
OS-TLA-ADV-09	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.

This vulnerability may be exploited by a malicious user who takes out a flash loan to significantly 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 can 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. This can be achieved by creating a vector to store the names of all extra reward coins and using them in the claim function.

Patch

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

OS-TLA-ADV-01 [high] | Deducting Vault Interest When Repaying Debt

Description

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

Although the protocol uses `fees::absorb_fee` to calculate and absorb the repaid interest amount, this amount is not subtracted from the `vault.interest`. Consequently, a user is unable to clear the interest in their vault, even though it is absorbed 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-02 [high] | Improper Reward Calculations

Description

In the protocol module, `accumulated_gain` calculates the earnings of a token based solely on the scale of the snapshot. However, it is possible for a user's amount to have participated in the distribution of the next 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

```
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 calculation of earnings 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-03 [med] | Precision Loss Issue In Weighted Math

Description

In the math module, the calculation of the amount taken in during a swap is 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 can 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. Let's 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 can 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, which will help avoid any rounding problems that could 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-04 [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-05 [med] | Including Interest In Vault CR Calculation

Description

`redeem_collateral` and `liquidate` calculate the collateral ratio (CR) for a vault, which is used in redemption and liquidation calculations. However, these functions do not take into account 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, the `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-06 [low] | Improper Interest Accumulation Calculation

Description

In sync_interest_rate, the value of days_elapsed is derived by dividing seconds_elapsed by the number of seconds in a day. This calculation results in the truncation of any remaining seconds, which can 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 1. This causes the new interest index to be calculated for only 1 day, and the last updated timestamp would be incorrect by 23:59 hours. Consequently, the global interest index ratio and interest on vaults may be lower than expected.

Remediation

Increment the 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-07 [low] | Improper Withdrawal Fee Calculation Formula

Description

The withdrawal fee calculation is handled by `withdraw_mod`. To determine the withdrawal fee amount, the intended formula should utilize the `(1 - (elapsed_time / withdrawal_fee_period)) * withdrawal_fee_max_ratio` formula, which is designed to proportionally decrease the withdrawal fee 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-08 [low] | Improper Implementation Of Withdraw All

Description

A pool creator has the option to remove liquidity from the pool by specifying a percentage of liquidity to be removed in bps format. However, if the LBP is still ongoing, only a portion of the liquidity can be removed from the pool, with the entire liquidity only being 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 can 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

```
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 his pool.

OS-TLA-ADV-09 [low] | Improper Implementation Of Mint Cap Check

Description

The number of MOD tokens to be minted in `borrow` is limited by the `mint_cap` parameter set by the protocol. This is enforced by checking if the newly minted amount along with the previous total debt exceeds the `mint_cap`.

But in implementation, the amount considered in this check (`amount`) is different from the amount that is actually minted (`total_amount = amount + fee_amount`). This leads to the minting of MOD exceeding the amount set in the `mint_cap` parameter.

Remediation

Use the amount that is 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 could 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>` struct and is set during the initialization process of the oracle for that coin.

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

Remediation

Implement a `set_switchboard_config` function to give the manager the ability 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. This price is frequently updated by a protocol bot and can 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. This allows the manager 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 a `LBPCollection` has unique asset generics for each asset pair, calculating the hash with asset names is unnecessary. Instead, the creator's address may be used as the key for the LBP instances.

Remediation

Use the creator address as the key for LBPs in `LBPCollection` and rewrite all of the 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 can be exploited to result in a loss of funds from the protocol. To prevent these issues, it is important to implement functions with a specific direction of rounding.

For instance, when calculating the amount to be given to the user for a specific input token, the calculations should be structured such that the amount is rounded down.

This involves rounding down all variables in the formula that are directly proportional to the output amount while rounding up those that are inversely proportional. This pattern should be applied to arithmetic operations such as division and exponentiation.

By following these guidelines, potential rounding issues can be avoided.

Remediation

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

Patch

Fixed by including mitigations in 951ad53.

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

```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.

In an edge case where the `elapsed time == expiry_time_seconds`, since the value of `DEFAULT_RESERVE_RATIO_BPS` is set to zero, the calculated `price` and `decrease_ratio` will be zero while the `below_reserve` will be false since `0 < 0 = false`.

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

Remediation

Change the condition in the `if` statement in the `status` function 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

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

This may lead to a lesser interest in vaults since it is possible for the value to be rounded down.

Remediation

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

Patch

Fixed in d5c9605.

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

Description

The OracleParamChangeEvent event is emitted whenever a value in OracleParams is changed. However, the event does not specify the coin name for which the parameters are changed.

Remediation

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

In the math module, `compute_invariant_weights_u64` is used 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.