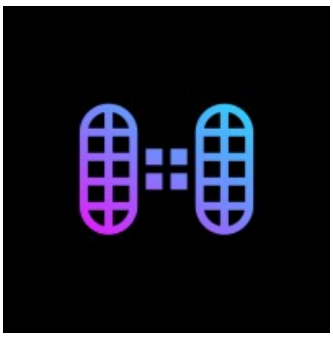


Yvaults

Audit

Presented by:

OtterSec contact@osec.io

Akash Gurugunti sud0u53r.ak@osec.io

Tamta Topuria tamta@osec.io

Robert Chen r@osec.io

Contents

Appendices

01 | Executive Summary

Overview

Hubble Protocol engaged OtterSec to perform an assessment of the `yvaults` program. This assessment was conducted between October 16th and November 14th, 2023. For more information on our auditing methodology, see Appendix B.

Key Findings

Over the course of this audit engagement, we produced 18 findings in total.

In particular, we identified several high-risk vulnerabilities, including one related to the arbitrary modification of a collateral's ID and mint fields without proper validation (OS-YVT-ADV-02) and another issue concerning the lack of validation of the reward collateral ID during the swapping of rewards, enabling the utilization of a collateral token with a high swap rewards discount basis points value (OS-YVT-ADV-01). Additionally, we highlighted an inconsistency in array lengths between two arrays (OS-YVT-ADV-03).

We also made recommendations around gas and storage optimizations (OS-YVT-SUG-00, OS-YVT-SUG-02) and the requirement of utilizing a two-step verification process to confirm a change in admin address (OS-YVT-SUG-03). We also brought to light several instances of unutilized and redundant code (OS-YVT-SUG-07, OS-YVT-SUG-09) and advised against the repetitive definition of the same seeds for the generation of program derived addresses (OS-YVT-SUG-01). Furthermore, we suggested incorporating proper error handling (OS-YVT-SUG-06) and the inclusion of specific checks (OS-YVT-SUG-05) to ensure the robustness and security of the system.

02 | Scope

The source code was delivered to us in a git repository at github.com/hubbleprotocol/yvaults. This audit was performed against commit 5cf3536.

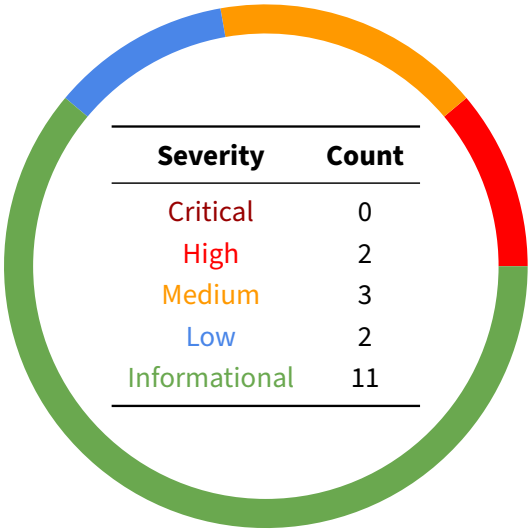
A brief description of the programs is as follows:

Name	Description
yvaults	A concentrated liquidity market maker for managing strategies, rewards, and various liquidity operations on the Solana blockchain, which integrates with external exchange protocols such as Orca and Raydium.

03 | Findings

Overall, we reported 18 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 aid in mitigating future vulnerabilities.



Severity	Count
Critical	0
High	2
Medium	3
Low	2
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-YVT-ADV-00	High	Resolved	Erasure of all rewards stored in the previous `reward_vault` by updating the `reward_vault` address.
OS-YVT-ADV-01	High	Resolved	`swap_rewards` lacks validation for `reward_collateral_id`, enabling the utilization of a collateral token with a high `swap_rewards_discount_bps`.
OS-YVT-ADV-02	Medium	Resolved	Arbitrary modification of a collateral's ID and mint fields without proper validation.
OS-YVT-ADV-03	Medium	Resolved	Inconsistency in array lengths between the `Price` and `infos` arrays.
OS-YVT-ADV-04	Medium	Resolved	The loss of fees and rewards associated with the old position resulted from the strategy not pointing to the position.
OS-YVT-ADV-05	Low	Resolved	Potential risk of loss of funds for non-manual re-balancing strategies that utilize the current pool price as the reference price for re-balancing.
OS-YVT-ADV-06	Low	Resolved	Invoking `open_liquidity_position` via cross-program invocations may yield unintended behavior during flash swaps of uneven vaults.

OS-YVT-ADV-00 [high] | Reward Vault Overwrite

Description

`UpdateRewardMapping`, which initializes and sets a new `reward_vault`, has no restrictions and is callable by anyone, allowing the modification of the `reward_vault` address. Thus, when calling the instruction with the same `reward_mint` set, it passes all the existing mint checks, allowing the replacement of the current `reward_vault` with a new empty vault, resulting in the loss of rewards that were present in the original `reward_vault`.

yvault-update_reward_mapping.rs RUST

```rust
#[derive(Accounts)]
pub struct UpdateRewardMapping<'info> {
    [...]
    #[account(init,
        payer = payer,
        token::mint = reward_mint,
        token::authority = base_vault_authority
    )]
    pub reward_vault: Box<Account<'info, TokenAccount>>,
    [...]
}
```

Furthermore, within `UpdateRewardMapping`, the values of `reward_0_amount`, `reward_1_amount`, and `reward_2_amount` are not explicitly set to zero after the update of the reward vault. Failing to reset these amounts appropriately may result in the retention of their previous values, potentially resulting in inaccurate calculations, unexpected behavior, or inconsistencies in reward management.

Remediation

Perform an initial verification at the start of the `UpdateRewardMapping` to confirm that `reward_0_amount`, `reward_1_amount`, and `reward_2_amount` are all set to zero and return an error if not.

Patch

Fixed in #579 by adding checks to ensure that reward vault is not already initialized before updation.

OS-YVT-ADV-01 [high]| Unchecked Reward Collateral

Description

`swap_rewards` triggers a rewards swap, transferring specified amounts of `token_a` and `token_b`. It swaps them for collateral tokens and ensures the resulting amount meets a minimum threshold: (`min_collateral_token_out`).

swap_rewards.rs RUST

```rust
pub fn handler_swap_rewards(
    ctx: Context<SwapRewards>,
    token_a_in: u64,
    token_b_in: u64,
    reward_index: u64,
    reward_collateral_id: u64,
    min_collateral_token_out: u64,
) -> Result<()> {
    [...]
    } = dbg_msg!(swap_reward(
        strategy,
        config,
        &prices,
        token_a_in,
        token_b_in,
        reward_index.try_into().unwrap(),
        reward_collateral_id.try_into().unwrap(),
        min_collateral_token_out
    ))?;
    [...]
}
```

However, there is a vulnerability that allows the attacker to manipulate the `reward_collateral_id` parameter as the program does not validate that the provided `reward_collateral_id` corresponds to a valid reward in the `CollateralInfos`. This enables them to specify an arbitrary collateral ID with more favorable swap parameters, such that when swapping tokens, it results in a more favorable exchange rate.

Remediation

Ensure the specified collateral ID corresponds to a valid collateral token in the `CollateralInfos` storage.

Patch

Fixed in #709 by directly fetching `reward_collateral_id` from the strategy instead of taking it from the input parameters.

OS-YVT-ADV-02 [med]| Arbitrary Modification Of Structure Fields

Description

`update_collateral_info` updates various fields of the `CollateralInfo` structure based on the provided `UpdateCollateralInfoMode` and corresponding `value`.

vault_operations.rs RUST

```rust
pub fn update_collateral_info(
    coll_info: &mut CollateralInfos,
    index: usize,
    mode: UpdateCollateralInfoMode,
    value: &[u8; VALUE_BYTE_ARRAY_LEN],
) {
    [...]
    match mode {
        UpdateCollateralInfoMode::CollateralId => {
            let value = Pubkey::from(*value);
            msg!("Value {:?}", value);
            coll_info.infos[index].mint = value;
        }
    }
}
```

The issue arises with modifying the mint associated with the collateral, as shown above. Changing the mint of a `CollateralInfo` structure at a specific index arbitrarily may result in significant consequences as the `mint` field typically represents the identifier of the associated token utilized as collateral. This information is crucial for various functionalities within the system.

vault_operations.rs RUST

```rust
pub fn update_strategy_config_u64(
    strategy: &mut WhirlpoolStrategy,
    mode: StrategyConfigOption,
    value: u64,
    pubkey_value: Pubkey,
    ts: u64,
) -> Result<(), VaultError> {
    [...]
    match mode {
        StrategyConfigOption::UpdateCollateralIdA => {
            strategy.token_a_collateral_id = value;
        }
        StrategyConfigOption::UpdateCollateralIdB => {
            strategy.token_b_collateral_id = value;
        }
    }
}
```

Similarly, in `update_strategy_config_u64`, updating the `token_a_collateral_id` and `token_b_collateral_id` fields in the `WhirlpoolStrategy` structure highlighted above without considering the associated fields such as `token_a_mint`, `token_b_mint`, `token_a_mint_decimals`, `token_b_mint_decimals`, etc., may result in inconsistencies in the data.

Since these fields are linked, updating collateral IDs without properly considering related fields may break the logic that relies on the integrity of these linked values. Inconsistencies frequently arise when identifying loans, trades, or any process that utilizes collateral, which references either the `mint` or the collateral IDs to identify and work with specific tokens.

Remediation

Include a check in `update_collateral_info` before updating the mint to ensure that the previous value is the default public key `Pubkey::default()`. Throw an error if not and restrict the modification of `token_a_collateral_id` and `token_b_collateral_id`.

Patch

Fixed in #693 by adding a check in `update_collateral_info` before updating the mint to ensure that the previous value is the default public key `Pubkey::default()` and by restricting the modification of `token_a_collateral_id` and `token_b_collateral_id`.

OS-YVT-ADV-03 [med]| Discrepancy In Array Length

Description

The `TokenPrices` structure manages an array of optional `Price` instances for multiple tokens. It provides methods to get and set prices for tokens and calculates the market value of a token in USD. Currently, the length of the prices array in `TokenPrices` is set to 128.

```rust
#[derive(Debug)]
pub struct TokenPrices {
    pub prices: [Option<Price>; 128],
}

#[account(zero_copy)]
#[derive(Debug, AnchorSerialize)]
pub struct CollateralInfos {
    pub infos: [CollateralInfo; 256],
}
```

This may conflict with the length of the `infos` field in the `CollateralInfos` structure. `TokenPrices` stores the price for different collateral tokens stored in the `CollateralInfos` structure. Thus, if the length of `TokenPrices::Price` and `CollateralInfos::infos` do not match, as in the above instance, it may result in inconsistencies when storing the price in `TokenPrices::price` as it only accepts a maximum of 128 entries as opposed to 256 possible entries.

Remediation

Ensure the `prices` field in `TokenPrices` is 256 instead of 128 to match the length of the `infos` field in `CollateralInfos`.

Patch

Fixed in #694 by doing a major refactoring that includes changing the `TokenPrices` to `StrategyTokenPrices`, which only stores prices of `token_a`, `token_b` and the reward tokens.

OS-YVT-ADV-04 [med]| Unharvested Funds

Description

When opening a new liquidity position via `open_liquidity_position`, there is a check to see if fees or rewards remain in the previous position. The function prevents the closing of the old position if fees or rewards are present. However, even though the code does not close the position, it updates the strategy to point to the new position. This may create a situation where the strategy is associated with the new position, with fees and rewards remaining in the previous position.

open_liquidity_position.rs RUST

```rust
pub fn handler<'c, 'info>(
    ctx: Context<'_, '_, 'c, 'info, OpenLiquidityPosition<'info>>,
    candidate_tick_lower_index: i64,
    candidate_tick_upper_index: i64,
    bump: u8,
) -> Result<()> {
    [...]
    (RebalanceEffects::WithdrawAndFreeze, StrategyStatus::Rebalancing) => false,
    (_, StrategyStatus::Uninitialized | StrategyStatus::NoPosition) => {
        // assert no account injected
        require!(
            old_position_or_base_vault_authority.key() ==
                ↪  strategy.base_vault_authority,
            VaultError::UnharvestedAmounts
        );
        msg!("Initializing a new strategy's position");
        false
    }
    [...]
}
```

As a result, a loss of fees and rewards may occur with the old position as the strategy no longer points to it.

Remediation

Handle fees and rewards from the previous position properly before updating the strategy to point to a new position. For example, transfer any remaining fees and rewards from the old position before updating the strategy.

Patch

Acknowledged by the development team.

OS-YVT-ADV-05 [low] | Oracle Price Manipulation

Description

The vulnerability affects non-manual re-balancing strategies that rely on the current pool price as a reference for re-balancing in `yvaults`. A user can manipulate the pool price by executing a flash loan, opening a position based on the changed price via `open_liquidity_position`, and then calling `invest` to invest in that position and repay the flash loan, all in the same transaction.

This manipulates the reference price during position opening and satisfies the checks in the `invest` instruction by resetting the pool price to a value within the expected deviation range. Thus, this creates a scenario where, despite the initial checks passing, the dynamic nature of the pool price during the transaction leads to an eventual loss for liquidity providers.

Proof of Concept

1. The attacker initiates a flash loan and manipulates the current price on the pool to be artificially lower than the actual market price. For example, the market price is $10, but the attacker manipulates the pool price to $8.
2. The attacker calls `open_liquidity_position` with the manipulated price.
3. The position is determined by re-balancing parameters and the manipulated reference price (in this case, $8). Say the position decides to open in the range [$7 - $9].
4. After opening the position, the attacker resets the pool price to a higher value, in this case, $9, to satisfy checks in `invest`, specifically the oracle market price deviation check and the current pool price in the position's price range check.
5. The attacker then calls `invest` with the manipulated parameters, which checks that the current pool price is within the expected deviation range at the start of the transaction, satisfying the checks in place.
6. As the transaction progresses, the real market price remains $10, but the manipulated pool price increases from $9 due to the liquidity pool investment.
7. Eventually, the pool price goes beyond the upper limit of the position's price range ([$7 - $9]), resulting in a loss for liquidity pools.

Remediation

Ensure that the manipulated reference price remains consistent throughout the transaction.

Patch

Acknowledged by the development team and mitigated by only using `ReferencePriceType::TWAP` for rebalancing.

OS-YVT-ADV-06 [low] | Missing Cross-Program Invocation Check

Description

The program may trigger `open_liquidity_position` via a cross-program invocation in a flash swap involving uneven vaults. The lack of `check_cpi!(ctx)` permits the invocation of `open_liquidity_position` within a cross-program invocation during a flash swap. Flash swaps, particularly with uneven vaults, may introduce unanticipated alterations in asset balances, giving rise to unintended outcomes due to the dynamic characteristics of flash swaps and uneven vaults.

open_liquidity_position.rs RUST

```rust
pub fn handler<'c, 'info>(
    ctx: Context<'_, '_, 'c, 'info, OpenLiquidityPosition<'info>>,
    candidate_tick_lower_index: i64,
    candidate_tick_upper_index: i64,
    bump: u8,
) -> Result<()> {
    let strategy = &mut ctx.accounts.strategy.load_mut()?;
    let dex = DEX::try_from(strategy.strategy_dex).unwrap();
    let base_vault_authority_bump =
        ↪  u8::try_from(strategy.base_vault_authority_bump)
        .map_err(|_| VaultError::OutOfRangeIntegralConversion)?;
    let clmm: Box<dyn Clmm> = clmm!(&ctx, dex);

    let clock = Clock::get().unwrap();
    let rebalance_effects = {
        let twap_prices = crate::utils::scope::get_twap_prices(
            &ctx.accounts.scope_prices,
            &ctx.accounts.token_infos.load()?.infos,
            strategy,
            clock.slot,
        )
        .ok();
    [...]
}
```

Remediation

Add the `check_cpi!(ctx)` macro as a safeguard.

Patch

Fixed in #583 by adding a stack height check in `open_liquidity_position` instruction to avoid CPI calls.

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 result in security issues in the future.

ID	Description
OS-YVT-SUG-00	`update_collateral_info` must be called multiple times to set each field, resulting in higher gas costs.
OS-YVT-SUG-01	Defining the same seeds multiple times to generate program derived addresses redeclares the same seed multiple times.
OS-YVT-SUG-02	Recommendations regarding optimization of storage space.
OS-YVT-SUG-03	Utilize a two-step verification process to confirm a change in admin address.
OS-YVT-SUG-04	Immediate unwrapping of the `Option<bool>` parameter `round_up` panics if the argument is `None`.
OS-YVT-SUG-05	Incorporate absent validations.
OS-YVT-SUG-06	Lack of proper error handling.
OS-YVT-SUG-07	Removal of unutilized code to improve code readability.
OS-YVT-SUG-08	Minor modifications to the code base.
OS-YVT-SUG-09	Removal of redundant code may improve efficiency.
OS-YVT-SUG-10	Suggestions regarding best coding practices.

OS-YVT-SUG-00 | Gas Optimization

Description

`update_collateral_info` allows for updating individual fields of a `CollateralInfo` instance.

vault_operations.rs RUST

```rust
pub fn update_collateral_info(
    coll_info: &mut CollateralInfos,
    index: usize,
    mode: UpdateCollateralInfoMode,
    value: &[u8; VALUE_BYTE_ARRAY_LEN],
) {
    msg!(
        "Updating mode={:?} id={} name={} value={:?}",
        mode,
        index,
        std::str::from_utf8(value.as_ref()).unwrap_or("NOT SET"),
        value
    );
    match mode {
        [...]
    }
}
```

However, since the program initializes all fields with default values, users who wish to add new collateral must specify all the fields. This requires them to call the function multiple times, specifying different fields each time. Setting multiple fields individually will be inconvenient for users and incur a high gas cost.

Remediation

Implement a functionality that takes a `CollateralInfo` structure as an argument, allowing users to set all the required fields in a single transaction.

OS-YVT-SUG-01 | Seed Reuse

Description

Program derived addresses involve the creation of derived addresses based on a combination of seeds. Currently, the protocol defines seeds while creating the program derived address. This results in a lack of localization and increased effort when the protocol must update seeds later, as the seeds must change at multiple locations throughout the code base.

Thus, it hampers the code base's maintainability and readability and increases the possibility of assigning an incorrect seed or failure to update a particular section of the code base.

Remediation

Utilize constants as seeds to make the code more maintainable and adaptable by providing a single control point for managing seeds. Also, give constants meaningful names to improve code readability.

OS-YVT-SUG-02 | Storage Optimization

Description

In `init_kamino_reward`, a `reward_vault` is passed as a parameter and directly stored in the `WhirlpoolStrategy` structure. For each Kamino reward, the program stores the corresponding `reward_vault` address explicitly, resulting in increased storage costs.

vault_operations.rs RUST

```
pub fn init_kamino_reward(
    strategy: &mut WhirlpoolStrategy,
    kamino_reward_index: usize,
    collateral_token: u64,
    reward_mint: Pubkey,
    reward_vault: Pubkey,
    decimals: u64,
) {
    [...]
    strategy.kamino_rewards[kamino_reward_index].reward_vault = reward_vault;
    [...]
}
```

Similarly, most program derived addresses utilized while initializing `strategy` must not be stored in the strategy struct, as the stored seeds may be employed to derive and validate them dynamically when needed.

Remediation

1. Utilize a program derived address generated dynamically during initialization instead of directly passing and storing the `reward_vault`. Use `init_if_needed` to create the vault if it does not exist yet.
2. Utilize the stored seeds instead of storing the program derived addresses.

OS-YVT-SUG-03 | Double Verification For Owner Change

Description

The `GlobalConfig` structure represents the global configuration settings. Currently, the admin change process for `GlobalConfig` is single-step; there is no confirmation step. Once the transaction is submitted, the admin change is irreversible. This may result in a denial of service when the current admin accidentally sends an unintended input as a parameter while executing an admin change.

Remediation

Utilize a two-step process to change the admin of the lending market.

OS-YVT-SUG-04 | Immediate Option Unwrapping

Description

`get_liquidity_from_amounts` expects `round_up` to be an `Option<bool>`, however, it immediately unwraps `Option` utilizing `unwrap()`. This implies that the program always expects `round_up` argument to have a value (`Some`), and if it is `None`, the `unwrap()` will panic, resulting in a runtime error.

orca_clmmm.rs RUST

```rust
pub fn get_liquidity_from_amounts(
    current_sqrt_price: u128,
    mut sqrt_price_a: u128,
    mut sqrt_price_b: u128,
    amount_a: u64,
    amount_b: u64,
    round_up: Option<bool>,
) -> u128 {
    [...]
}
```

Remediation

Modify the function signature to accept a non-optional `round_up` parameter directly if the function should always receive the argument.

OS-YVT-SUG-05 | Missing Checks

Description

1. Currently, there is a lack of checks when updating `global_config`, `strategy` and `CollateralInfos`, especially when dealing with basis points values. It is crucial to ensure that these values do not exceed certain upper bounds to prevent them from being too high, which would result in inconsistent values.
2. In `collect_fees_and_rewards` and `swap_uneven_vaults`, there is no check for the public key of the `raydium_protocol_position_or_base_vault_authority` account against the value stored in `strategy`, potentially raising issues from unexpected or manipulated account addresses.

Remediation

1. Include checks in the code logic to ensure that values are within expected ranges while updating `global_config`, `strategy` and `CollateralInfos`.
2. Validate the public key of `raydium_protocol_position_or_base_vault_authority` with the value stored in `strategy` to ensure the account being accessed is the expected one.

OS-YVT-SUG-06 | Error Handling

Description

`SwapUnevenVaults` currently supports only Orca pools. Therefore, using it with Raydium strategies may have unintended effects as it is unsupported. It is suitable to prevent unintended utilization of `SwapUnevenVaults` with Raydium strategies.

Remediation

Include an early panic with a clear error message stating the incompatibility with Raydium.

OS-YVT-SUG-07 | Unused Code

Description

1. `treasury_fee_vaults` is currently not utilized in any operations or instructions, and the protocol directly collects fees into program derived address token accounts. Therefore, storing 256 public keys in `global_config.treasury_fee_vaults` is unnecessary. Consequently, the `UpdateTreasuryFeeVault` instruction is unnecessary.

state.rs RUST

```rust
#[account(zero_copy)]
#[derive(Debug)]
pub struct GlobalConfig {
    [...]
    pub treasury_fee_vaults: [Pubkey; 256],
    [...]
}
```

2. The program does not utilize the `global_config` and `pool_program` accounts passed to the `ChangePool` anywhere.

instructions/change_pool.rs RUST

```rust
#[derive(Accounts)]
pub struct ChangePool<'info> {
    [...]
    /// CHECK: has_one in strategy
    pub global_config: AccountLoader<'info, GlobalConfig>,

    /// CHECK: dex account
    pub pool_program: AccountInfo<'info>,
    [...]
}
```

3. Certain fields within the `strategy` appear redundant, such as `token_a_vault_authority`, `token_a_vault_authority_bump`, `token_b_vault_authority`, `token_b_vault_authority_bump`, `scope_program`, and `shares_mint_authority`, and may be omitted. Additionally, consider removing the `fees_bps` field in `GlobalConfig`.

Remediation

1. Utilize the `init_if_needed` pattern to create program derived address token accounts dynamically as required instead of storing all 256 public keys in advance. This saves space and reduces the initialization overhead. Also remove `UpdateTreasuryFeeVault`.
2. Remove the unused accounts.
3. Remove the unutilized fields.

OS-YVT-SUG-08 | Code Refactoring

Description

1. `get_amount_b_for_liquidity` in `utils/clmm_calcs.rs` does not utilize `math::div_round_up`, whereas `get_amount_a_for_liquidity` utilizes it for executing the same computation.
2. In `get_next_expansion_step`, `is_expansion_in_range` is called repeatedly to calculate `reference_price` via `get_reference_price` in each iteration of the loop, even though `reference_price` is static within the loop. This results in redundant operations and more gas costs.

expander.rs RUST

```
pub fn get_next_expansion_step(
        &self,
        rebalance_request: &RebalanceRequest,
    ) -> VaultResult<ExpanderStep> {
        [...]
        while lower_bound < upper_bound {
            let candidate_expansions = (lower_bound + upper_bound) / 2;
            if self.is_expansion_in_range(rebalance_request,
                [...]
        }
}
```

3. A redundant match statement is within the invest instruction in the `InvestEffects` calculation. The `clmm!(ctx, dex)` macro already handles the logic of determining the appropriate concentrated liquidity market maker based on the provided exchange.
4. In `deposit`, `holdings` are calculated twice. The program does not utilize the value of `holdings` obtained in the first calculation for any intermediate computations or modifications before the second calculation.

Remediation

1. Utilize `math::div_round_up` in `get_amount_b_for_liquidity` to ensure consistency with `get_amount_a_for_liquidity` and to align the rounding behavior for calculating amounts.
2. Call `is_expansion_in_range` once outside the loop to calculate `reference_price`.
3. Remove the redundant match statement.
4. Reuse the value obtained in the first calculation.

OS-YVT-SUG-09 | Redundant Code

Description

1. The utility functions that transfer funds to and from token vaults and treasury fee vaults are redundantly defined across multiple files, decreasing readability and potentially starting complications during updates to this functionality.
2. `get_prices` in `utils::scope` accepts a `clmm` parameter, suggesting that it internally computes prices. This renders calculating and storing prices in the `Prices` structure within `SwapUnevenVaults`, `OpenLiqudityPosition`, `ExecutiveWithdraw`, and `EmergencySwap` instructions unnecessary.
3. The `new_account` parameter within `UpdateStrategyConfig` is solely utilized to retrieve its public key. Directly passing this public key as an argument would enhance readability and provide only the necessary information.

Remediation

1. Define these helper functions in one `utils` file.
2. Remove the storage of prices in the `Prices` structure within `SwapUnevenVaults`, `OpenLiqudityPosition`, `ExecutiveWithdraw`, and `EmergencySwap`.
3. Pass the public key directly as an argument.

OS-YVT-SUG-10 | Code Maturity

Description

1. The comment on `swap_rewards_discount_bps` in the `GlobalConfig` structure incorrectly states 128 types of tokens when it should be 256.

valut_operations.rs RUST

```
pub struct GlobalConfig {
    // 128 types of tokens, indexed by token
    pub swap_rewards_discount_bps: [u64; 256],
    [...]
}
```

2. `STRATEGY` is spelled incorrectly as `STRATEY` in the `MAX_UNBALANCED_STRATEY_PRICE_DEVIATION_BPS` constant.
3. The error code `WrongRewardCollateralID` is raised in `get_twap_prices_from_data` and `get_prices_from_data`, implying an issue with collateral IDs, particularly in the context of rewards. This is the case even though the collateral IDs are used to fetch TWAP and regular prices rather than specifically for rewards.
4. Currently, the code utilizes `u64` to represent the program derived address bumps. However, as the range of bumps is relatively small, utilizing a smaller data type may increase storage efficiency.
5. The `InitializeCollateralInfo` instruction may be public, as its purpose is solely to initialize the `coll_info` account with default values and is not directly associated with the global configuration.
6. In `get_swap_vaults_effects`, the comment says community vaults have minimum slippage of **10 bps** while the actual value of `MIN_SLIPPAGE_COMMUNITY_VAULTS_WHEN_SLIPPAGE_DISABLED_BPS` is five.

Remediation

1. Modify the comment from 128 types of tokens to 256.
2. Rectify the spelling mistake in `MAX_UNBALANCED_STRATEY_PRICE_DEVIATION_BPS`.
3. Ensure the proper usage of error codes.
4. Utilize the `u8` data type to store program-derived address bumps.
5. Make the `InitializeCollateralInfo` instruction public.
6. Change the value to five.

A | Vulnerability Rating Scale

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

Critical	Vulnerabilities that immediately result in a 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 may result in a 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 may result in denial of service scenarios or degraded usability. Examples: • Computational limit exhaustion through malicious input. • Forced exceptions in the normal user flow.
Low	Low probability vulnerabilities, which are still 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 program's implementation 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 thumb, 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 comprehensively understand 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.