sec3™

DRAFT

FOR REVIEW ONLY

Security Assessment Report

Kamino yvaults v0.1.0

September 28th, 2022

Summary

The sec3 team (formerly Soteria) was engaged to do a thorough security analysis of the Kamino yvaults v0.1.0 Solana smart contract programs. The artifact of the audit was the source code of the following on-chain smart contracts excluding tests in a private repository.

The audit was done on the following contract

- **Contract "yvaults":**
    - Commit `b155e16ead1d4c1e06e57623c2f3169ff7e77b82`

The audit revealed 11 issues or questions. This report describes the findings and resolutions in detail.

Table of Contents

Methodology and Scope of Work 3

Result Overview 4

Findings in Detail 5

[C-1] Multiple global configuration accounts allowed 5

[C-2] Missing token collateral id validation in strategy initialization 6

[C-3] Missing reward collateral id validations 8

[H-1] Stale oracle risk due to large MAX_PRICE_AGE_SECONDS constant 11

[H-2] Unvalidated scope_program and scope_prices 13

[L-1] Unsafe “as” type conversion 14

[L-2] Validate the tick array accounts when opening the position 16

[L-3] Unhandled exponent scenario 17

[L-4] old_position_or_rent insufficiently validated 18

[L-5] Check user-provided parameters 19

[I-1] Questions 23

Methodology and Scope of Work

The sec3 (formerly Soteria) audit team, which consists of Computer Science professors and industrial researchers with extensive experience in Solana smart contract security, program analysis, testing and formal verification, performed a comprehensive manual code review, software static analysis and penetration testing.

Assisted by the sec3 Scanner developed in-house, the audit team particularly focused on the following work items:

- Check common security issues.
  - Missing ownership checks
  - Missing signer checks
  - Signed invocation of unverified programs
  - Solana account confusions
  - Arithmetic over- or underflows
  - Numerical precision errors
  - Loss of precision in calculation
  - Insufficient SPL-Token account verification
  - Missing rent exemption assertion
  - Casting truncation
  - Did not follow security best practices
  - Outdated dependencies
  - Redundant code
  - Unsafe Rust code
- Check program logic implementation against available design specifications.
- Check poor coding practices and unsafe behavior.
- The soundness of the economics design and algorithm is out of scope of this work

Result Overview

In total, the audit team found the following issues.

CONTRACT YVAULTS v0.1.0

Issue	Impact	Status
[C-1] Multiple global configuration accounts allowed	Critical	Fixed
[C-2] Missing token collateral id validation in strategy initialization	Critical	Open
[C-3] Missing reward collateral id validations	Critical	Open
[H-1] Stale oracle risk due to large MAX_PRICE_AGE_SECONDS constant	High	Open
[H-2] Unvalidated scope_program and scope_prices	High	Open
[L-1] Unsafe “as” type conversion	Low	Open
[L-2] Validate the tick array accounts when opening the position	Low	Open
[L-3] Unhandled exponent scenario	Low	Open
[L-4] old_position_or_rent insufficiently validated	Low	Open
[L-5] Check user-provided parameters	Low	Open
[I-1] Questions	Informational	Open

Findings in Detail

IMPACT – CRITICAL

[C-1] Multiple global configuration accounts allowed

The global config is not enforced to be a singular account. For example, it is not a PDA with constant seeds. Besides, the `initialize_global_config` instruction can also be called by anyone to arbitrarily initialize an empty `GlobalConfig` account. Since it contains important global settings (e.g. authorizations), this instruction should not be permissionless.

```
/* yvaults/programs/yvaults/src/lib.rs */
054 | pub fn initialize_global_config(ctx: Context<InitializeGlobalConfig>) -> Result<()> {
055 |     instructions::initialize_global_config::handler(ctx)
056 | }

/* yvaults/programs/yvaults/src/instructions/initialize_global_config.rs */
016 | pub struct InitializeGlobalConfig<'info> {
017 |     #[account(mut)]
018 |     pub admin_authority: Signer<'info>,
019 |
020 |     #[account(zero)]
021 |     pub global_config: AccountLoader<'info, GlobalConfig>,
022 |
023 |     pub system_program: Program<'info, System>,
024 | }
```

Resolution

This issue has been fixed. The instruction access control has been added and this instruction is only available in the `integration_tests` mode.

IMPACT – CRITICAL

[C-2] Missing token collateral id validation in strategy initialization

For a given amount of token X, its USD value is determined by the token price and the token decimals.

For the token decimals, they come from the token mints (in strategy initialization), which are validated against the orca whirlpool token mints. Therefore, the token decimals saved in the strategy are consistent with the actual tokens.

For the token price, the contract maps a unique token collateral id to the token price obtained from the scope contract as follows:

```rust
/* yvaults/programs/yvaults/src/utils/scope.rs */
pub fn get_prices(
    scope_prices: &AccountInfo,
    strategy: &mut WhirlpoolStrategy,
) -> Result<Box<TokenPrices>> {
    let scope_prices = get_price_account(scope_prices)?;
    let token_a = CollateralToken::try_from(strategy.token_a_collateral_id)
        .map_err(|_| VaultError::WrongRewardCollateralID)?;
    let token_b = CollateralToken::try_from(strategy.token_b_collateral_id)
        .map_err(|_| VaultError::WrongRewardCollateralID)?;
    let price_a = get_price_usd(&scope_prices, token_a, Clock::get()?.slot)?;
    let price_b = get_price_usd(&scope_prices, token_b, Clock::get()?.slot)?;
    let mut prices = Box::new(TokenPrices::default());
    prices.set(token_a, price_a);
    prices.set(token_b, price_b);
```

However, when initializing the strategy, the user-provided collateral ids are not validated to make sure they are consistent with the actual tokens. As a result, the loaded prices that are determined by `strategy.token_a_collateral_id` and `strategy.token_b_collateral_id` can be different from the actual prices.

Although there is an additional price check against orca prices by `assert_valid_orca_price`, it's done on the price ratio (`price_a/price_b`) instead of the absolute value. Therefore, as long as the price ratio is within the range, this check can be bypassed. In addition, this extra price cross-check is enforced in some of the instructions (e.g. deposit and swap_uneven_vaults).

```
/* yvaults/programs/yvaults/src/operations/vault_operations.rs */
pub fn assert_valid_orca_price(
    strategy: &WhirlpoolStrategy,
    prices: &TokenPrices,
    sqrt_orca_price: U192,
) -> VaultResult<()> {
    let price_a =
    prices.get(CollateralToken::try_from(strategy.token_a_collateral_id).unwrap())?;
    let price_b =
    prices.get(CollateralToken::try_from(strategy.token_b_collateral_id).unwrap())?;

    let scope_price = calc_scaled_a_to_b_price(&price_a, &price_b)?;
    // Compute the normal price scaled by 2^64:
    // p * 2^64 = (sqrt(p) * 2^64)^2 * 2^64
    let orca_price = sqrt_orca_price
        .checked_mul(sqrt_orca_price)
        .ok_or(VaultError::IntegerOverflow)?
        >> U192::from(64);

    if orca_price < mul_bps_u192(scope_price,
                                 10_000 - strategy.max_price_deviation_bps)?
        || orca_price > mul_bps_u192(scope_price,
                                 10_000 + strategy.max_price_deviation_bps)?
    {
        return Err(VaultError::OrcaPriceTooDifferentFromScope);
    }

    Ok(())
}
```

This will affect the USD values of tokens and lead to inconsistent results in scenarios like LP share calculation.

IMPACT – CRITICAL

[C-3] Missing reward collateral id validations

Assume the token ids and prices are correct. Reward tokens have similar issues. More importantly, the id-price mapping can be manipulated at any time. For example, attackers may use correct settings to attract participation and change them right before the attack.

When setting the reward mappings via the `update_reward_mapping` instruction, the handler doesn't validate if the reward collateral id is consistent with the actual token. For example, it's possible to set the collateral id to the id of a much cheaper token while keeping the correct mint and decimals.

```
/* yvaults/programs/yvaults/src/instructions/update_reward_mapping.rs */
pub struct UpdateRewardMapping<'info> {
    #[account(init,
        payer = admin_authority,
        token::mint = reward_mint,
        token::authority = base_vault_authority
    )]
    pub reward_vault: Box<Account<'info, TokenAccount>>,

/* yvaults/programs/yvaults/src/instructions/update_reward_mapping.rs */
pub fn handler(
    ctx: Context<UpdateRewardMapping>,
    reward_id: u8,
    collateral_token: u8,
) -> Result<()> {
    match reward_id {
        0 => {
            strategy.reward_0_collateral_id = collateral_token as u64; // validate
        }
        1 => {
            strategy.reward_1_collateral_id = collateral_token as u64;
        }
        2 => {
            strategy.reward_2_collateral_id = collateral_token as u64;
        }
        _ => unimplemented!(),
    };
}
```

Although only admin_authority can do so, given that initialize_strategy becomes permissionless, attackers may create their strategies and set inconsistent reward mappings.

More importantly, this is a more convenient attack vector because it's possible to set the reward mapping repeatedly, as long as a new reward_vault is provided to satisfy the anchor init requirement. So, attackers may manipulate the reward mappings anytime.

Potential attacking scenarios could be:

1. Get more shares in deposit. The share price is largely set by existing_holding_value / share_issued. Since the holding value contains the reward value, by lowering the reward price and thus making the existing reward value less expensive, the share price will be cheaper such that more shares can be obtained for the same amount of the deposit.

```
/* yvaults/programs/yvaults/src/operations/vault_operations.rs */
pub fn get_price_per_full_share_impl(
    holdings: &Holdings,
    shares_issued: u64,
    shares_decimals: u64,
) -> VaultResult<U128> {
    if shares_issued == 0 {
        Ok(underlying_unit(shares_decimals))
    } else {
        let res = Ok(U128::from(
            Decimal::from(underlying_unit(shares_decimals).as_u128())
                .try_mul(holdings.total_sum.as_u128())?
                .try_div(shares_issued)?
                .try_ceil_u128()?,
        ));
        println!(
            "Calculating share_price={:?} with holdings={} and issued={}",
            res, holdings.total_sum, shares_issued
        );

        res
    }
}
```

2. Deposit fewer tokens but get more rewards in `swap_rewards`. For a given USD value of tokens a and b swapped-in, the swapped-out reward token amount is calculated by `calc_token_amount_from_usd_value()`. If the reward token price is cheaper due to an incorrect reward token collateral id, it's possible to get more reward tokens that are worth more than the input a and b tokens.

```rust
/* yvaults/programs/yvaults/src/utils/price.rs */
pub fn calc_token_amount_from_usd_value<'a>(
    usd_value: u64,
    price: impl Into<Option<&'a Price>>,
    token_decimals: u8,
) -> Result<U128, VaultError> {
    let price: Option<&Price> = price.into();
    // Don't check if the price is valid unless we really need it (amount > 0)
    if usd_value == 0 {
        return Ok(U128::from(0_u128));
    }

    let price = price.ok_or(VaultError::IntegerOverflow)?;

    U128::from(usd_value)
        .checked_mul(U128::from(ten_pow(
            token_decimals
                .checked_add(price.exp as u8)
                .ok_or(VaultError::IntegerOverflow)?
                .checked_sub(USD_DECIMALS_PRECISION)
                .ok_or(VaultError::IntegerOverflow)?,
        )))
        .ok_or(VaultError::IntegerOverflow)?
        .checked_div(U128::from(price.value))
        .ok_or(VaultError::IntegerOverflow)
}
```

IMPACT – HIGH

[H-1] Stale oracle risk due to large MAX_PRICE_AGE_SECONDS constant

The length that determines staleness of the oracle is set to an unusually high amount. This is especially relevant when the oracle has either low amount of price sources and/or sources with low liquidity. An example of when this led to a cascade of transactions that hurt users was the Socean incident. The oracle time tolerance yields significant risk under certain conditions.

```
/* yvaults/programs/yvaults/src/utils/scope.rs */
fn get_price_max_age(token: ScopePriceId) -> clock::Slot {
    match token {
        ScopePriceId::SOL => MAX_PRICE_AGE_SLOT,
        ScopePriceId::ETH => MAX_PRICE_AGE_SLOT,
        ScopePriceId::BTC => MAX_PRICE_AGE_SLOT,
        ScopePriceId::SRM => MAX_PRICE_AGE_SLOT,
        ScopePriceId::RAY => MAX_PRICE_AGE_SLOT,
        ScopePriceId::FTT => MAX_PRICE_AGE_SLOT,
        ScopePriceId::MSOL => MAX_PRICE_AGE_SLOT,
        ScopePriceId::BNB => MAX_PRICE_AGE_SLOT,
        ScopePriceId::AVAX => MAX_PRICE_AGE_SLOT,
        ScopePriceId::DaoSOL_SOL => MAX_PRICE_AGE_SLOT,
        ScopePriceId::SaberMSOL_SOL => MAX_PRICE_AGE_SLOT,
        ScopePriceId::USDH => MAX_PRICE_AGE_SLOT,
        ScopePriceId::StSOL => MAX_PRICE_AGE_SLOT,
        ScopePriceId::CSOL_SOL => MAX_PRICE_AGE_SLOT,
        ScopePriceId::CETH_ETH => MAX_PRICE_AGE_SLOT,
        ScopePriceId::CBTC_BTC => MAX_PRICE_AGE_SLOT,
        ScopePriceId::CMSOL_SOL => MAX_PRICE_AGE_SLOT,
        ScopePriceId::scnSOL_SOL => MAX_PRICE_AGE_SLOT,
        ScopePriceId::wstETH => MAX_PRICE_AGE_SLOT,
        ScopePriceId::LDO => MAX_PRICE_AGE_SLOT,
        ScopePriceId::USDC => MAX_PRICE_AGE_SLOT,
        ScopePriceId::CUSDC_USDC => MAX_PRICE_AGE_SLOT,
        ScopePriceId::USDT => MAX_PRICE_AGE_SLOT,
        ScopePriceId::ORCA => MAX_PRICE_AGE_SLOT,
        ScopePriceId::MNDE => MAX_PRICE_AGE_SLOT,
        ScopePriceId::HBB => MAX_PRICE_AGE_SLOT,
    }
}
```

As mentioned in [C-2], although the scope prices are further checked against the orca price via `assert_valid_orca_price` for some of the instructions, it's done on the price ratio (`price_a / price_b` v.s. `orcal_price`) instead of on the absolute price values. So, if both prices momentarily experience significant changes but their ratio is still within the range, the calculation of the LP share for a deposit could be off.

As discussed in the meeting, it could be helpful to

- Customize the time tolerance for different tokens.
- Not use the orca price in scope such that the scope-orca cross-check is more effective
- The TWAP-based checks added in a newer version is helpful too

IMPACT – HIGH

[H-2] Unvalidated scope_program and scope_prices

In strategy initialization, the user-provided scope_program and scope_prices are saved without proper validation. In the version for audit, instruction initialize_strategy is restricted to protocol maintainers so it's relatively safe. However, now this instruction is permissionless and the risk becomes higher. Since strategy.scope_prices is used by other instructions to validate scope price accounts and query token prices, if a faked scope_prices account is set here, all token prices can be manipulated.

```
/* yvaults/programs/yvaults/src/instructions/initialize_strategy.rs */
pub struct InitializeStrategy<'info> {
    /// CHECK: Trust it to be hardcoded by the admin
    #[account(owner = *scope_program.key)]
    pub scope_prices: AccountInfo<'info>,
    /// CHECK: Trust it to be hardcoded by the admin
    pub scope_program: AccountInfo<'info>,

/* yvaults/programs/yvaults/src/instructions/initialize_strategy.rs */
pub fn handler(
    ctx: Context<InitializeStrategy>,
    token_a_collateral_id: u64,
    token_b_collateral_id: u64,
) -> Result<()> {
    // Scope
    strategy.scope_prices = *ctx.accounts.scope_prices.key;
    strategy.scope_program = *ctx.accounts.scope_program.key;
```

Instead, the scope program id and scope price accounts should be set by protocol admin in the global_config, it's better to validate them against scope_program_id and scope_price_id in global_config.

```
/* yvaults/programs/yvaults/src/state.rs */
pub struct GlobalConfig {
    pub scope_program_id: Pubkey,
    pub scope_price_id: Pubkey,
```

IMPACT – LOW

[L-1] Unsafe "as" type conversion

The type casting via `as` is heavily used, which could be unsafe if it leads to silently truncating a primitive.

- i64 -> i32

```
/* yvaults/programs/yvaults/src/instructions/open_liquidity_position.rs */
pub fn handler(
    tick_lower_index: i64,
    tick_upper_index: i64,
    bump: u8,
) -> Result<()> {
    orca_operations::cpi_open_position_orca(
        tick_lower_index as i32,
        tick_upper_index as i32,
    )?;
```

- u64 -> i64

```
/* yvaults/programs/yvaults/src/instructions/update_strategy_config.rs */
pub fn handler(ctx: Context<UpdateStrategyConfig>, mode: u16, value: u64) -> Result<()> {
    let mode = StrategyConfigOption::try_from(mode).unwrap();
    let strategy = &mut ctx.accounts.strategy.load_mut()?;
    match mode {
        StrategyConfigOption::UpdateWithdrawalCapACapacity => {
            strategy.withdrawal_cap_a.config_capacity = value as i64
        }
        StrategyConfigOption::UpdateWithdrawalCapACurrentTotal => {
            strategy.withdrawal_cap_a.current_total = value as i64
        }
        StrategyConfigOption::UpdateWithdrawalCapBCapacity => {
            strategy.withdrawal_cap_b.config_capacity = value as i64
        }
        StrategyConfigOption::UpdateWithdrawalCapBCurrentTotal => {
            strategy.withdrawal_cap_b.current_total = value as i64
        }
    }
}
```

- u128 -> u64

```
/* yvaults/programs/yvaults/src/utils/clmm_calcs.rs */
pub fn quote_position_remove_liquidity_below_range(
    liquidity: u128,
    sqrt_price_lower: u128,
    sqrt_price_upper: u128,
    slippage: u64,
) -> VaultResult<RemoveLiquidityInput> {
    let token_est_a =
        get_amount_a_for_liquidity(...);
    Ok(RemoveLiquidityInput {
        token_est_a: token_est_a as u64, // u128 -> u64
    })
}

/* yvaults/programs/yvaults/src/utils/clmm_calcs.rs */
pub fn quote_position_add_liquidity_above_range(
    is_token_a: bool,
    sqrt_price_lower: u128,
    sqrt_price_upper: u128,
    amount: u64,
    slippage: u64,
) -> VaultResult<IncreaseLiquidityInput> {
    let token_est_b =
        get_amount_b_for_liquidity(...);
    Ok(IncreaseLiquidityInput {
        token_est_b: token_est_b as u64,  // u128 -> u64
    })
}
```

IMPACT – LOW

[L-2] Validate the tick array accounts when opening the position

When opening a position, `ticker_array_lower` and `ticker_array_upper` should be validated:

- they are associated with the correct `strategy.whirlpool`
- the tick array contains the provided tick index.

  e.g. `tick_array_lower.get_tick(tick_lower_index, whirlpool.tick_spacing)?`

Although they will be validated in the CPI calls to orca, since these accounts saved in `strategy` will be used to check user-provided tick arrays, it's better to validate them early.

```
/* yvaults/programs/yvaults/src/instructions/open_liquidity_position.rs */
pub fn handler(
    ctx: Context<OpenLiquidityPosition>,
    tick_lower_index: i64,
    tick_upper_index: i64,
    bump: u8,
) -> Result<()> {
    orca_operations::cpi_open_position_orca(
        &ctx,
        tick_lower_index as i32,
        tick_upper_index as i32,
        bump,
    )?;
    strategy.tick_array_lower = ctx.accounts.ticker_array_lower.key();
    strategy.tick_array_upper = ctx.accounts.ticker_array_upper.key();
}

/* yvaults/programs/yvaults/src/instructions/open_liquidity_position.rs */
pub struct OpenLiquidityPosition<'info> {
    /// CHECK: admin must send this correctly
    #[account(owner = orca_whirlpools::ID)]
    pub ticker_array_lower: AccountInfo<'info>,
    /// CHECK: admin must send this correctly
    #[account(owner = orca_whirlpools::ID)]
    pub ticker_array_upper: AccountInfo<'info>,
```

IMPACT – LOW

[L-3] Unhandled exponent scenario

`token_decimals + price.exp - USD_DECIMALS_PRECISION < 0` is not handled.

```rust
/* yvaults/programs/yvaults/src/utils/price.rs */
pub fn calc_market_value_token_usd<'a>(
    amount: u64,
    price: impl Into<Option<&'a Price>>,
    token_decimals: u8,
) -> Result<U128, VaultError> {
    let price = price.ok_or(VaultError::IntegerOverflow)?;
    U128::from(amount)
        .checked_mul(U128::from(price.value))
        .ok_or(VaultError::IntegerOverflow)?
        .checked_div(U128::from(ten_pow(
            token_decimals
                .checked_add(price.exp as u8)
                .ok_or(VaultError::IntegerOverflow)?
                .checked_sub(USD_DECIMALS_PRECISION)
                .ok_or(VaultError::IntegerOverflow)?,
        )))
        .ok_or(VaultError::IntegerOverflow)
}

pub fn calc_token_amount_from_usd_value<'a>(
    usd_value: u64,
    price: impl Into<Option<&'a Price>>,
    token_decimals: u8,
) -> Result<U128, VaultError> {
    let price: Option<&Price> = price.into();
    let price = price.ok_or(VaultError::IntegerOverflow)?;
    U128::from(usd_value)
        .checked_mul(U128::from(ten_pow(
            token_decimals
                .checked_add(price.exp as u8)
                .ok_or(VaultError::IntegerOverflow)?
                .checked_sub(USD_DECIMALS_PRECISION)
                .ok_or(VaultError::IntegerOverflow)?,
        )))
        .ok_or(VaultError::IntegerOverflow)?
        .checked_div(U128::from(price.value))
        .ok_or(VaultError::IntegerOverflow)
}
```

IMPACT – LOW

[L-4] old_position_or_rent insufficiently validated

The only requirement is that the old position has to be owned by orca. However, the old position's whirlpool, position_mint, tick_lower_index and tick_upper_index are not checked. It's possible to bypass the fee owned checks at line 35.

```
/* yvaults/programs/yvaults/src/instructions/open_liquidity_position.rs */
072 | pub struct OpenLiquidityPosition<'info> {
135 |     /// CHECK: checked in code
137 |     pub old_position_or_rent: AccountInfo<'info>,
138 | }

/* yvaults/programs/yvaults/src/instructions/open_liquidity_position.rs */
014 | pub fn handler(
015 |     ctx: Context<OpenLiquidityPosition>,
016 |     tick_lower_index: i64,
017 |     tick_upper_index: i64,
018 |     bump: u8,
019 | ) -> Result<()> {
020 |     let strategy = &mut ctx.accounts.strategy.load_mut()?;
021 |     let old_position_or_rent = *ctx.accounts.old_position_or_rent.key;
022 |     let status = StrategyStatus::try_from(strategy.status).unwrap();
023 |     match status {
027 |         StrategyStatus::Rebalancing => {
028 |             let old_position_acc = ctx.accounts.old_position_or_rent.clone();
029 |             require!(
030 |                 *old_position_acc.owner == orca_whirlpools::ID,
031 |                 VaultError::InvalidPositionAccount
032 |             );
033 |             let mut data = &old_position_acc.data.borrow_mut()[..];
034 |             let position: Position = Position::try_deserialize(&mut data)?;
035 |             if position.fee_owed_a > 0 || position.fee_owed_b > 0 {
036 |                 msg!("Cannot rebalance while fees unharvested");
037 |                 return Err(VaultError::UnharvestedAmounts.into());
038 |             }
039 |         }
```

IMPACT – LOW

[L-5] Check user-provided parameters

- Check the range of user provided collateral_id against the size of the treasury_fee_vaults.

```
/* yvaults/programs/yvaults/src/config/global_config_operations.rs */
pub fn update_treasury_fee_vault(
    collateral_id: u16,
) -> Result<(), VaultError> {
    global_config.treasury_fee_vaults[usize::from(collateral_id)] = ...;
}

/* yvaults/programs/yvaults/src/state.rs */
pub struct GlobalConfig {
    pub treasury_fee_vaults: [Pubkey; 19],
```

- Range check collateral_id

```
/* yvaults/src/instructions/update_global_config.rs */
pub fn handler(
    index: u16,
) -> Result<()> {
        GlobalConfigOption::SwapDiscountBps => {
            let collateral_id = index as usize;
            let discount = u64::from_le_bytes(value[..8].try_into().unwrap());
            global_config.swap_rewards_discount_bps[collateral_id] = discount;
        }
        GlobalConfigOption::TreasuryFeeVaults => {
            let collateral_id = index as usize;
            let vault = Pubkey::new(value);
            global_config.treasury_fee_vaults[collateral_id] = vault;
        }
```

- collateral_id range check is missing.

```
/* yvaults/programs/yvaults/src/instructions/update_treasury_fee_vault.rs */
pub fn handler(ctx: Context<UpdateTreasuryFeeVault>, collateral_id: u16) ->...{
    global_config_operations::update_treasury_fee_vault(
        collateral_id,
    )?;
}

/* yvaults/programs/yvaults/src/config/global_config_operations.rs */
pub fn update_treasury_fee_vault(
    collateral_id: u16,
) -> Result<(), VaultError> {
    global_config.treasury_fee_vaults[usize::from(collateral_id)] = ...
}

/* yvaults/programs/yvaults/src/state.rs */
#[account(zero_copy)]
#[derive(Debug)]
pub struct GlobalConfig {
    pub treasury_fee_vaults: [Pubkey; 19],
}
```

- Range checks for `value` for different scenarios.

```
/* yvaults/programs/yvaults/src/config/global_config_operations.rs */
pub fn update_u64_config(
    value: u64,
) -> Result<(), VaultError> {
    match key {
        GlobalConfigOption::EmergencyMode => {
            set_config!(global_config, emergency_mode, value);
        }
        GlobalConfigOption::BlockDeposit => {
            set_config!(global_config, block_deposit, value);
        }
        GlobalConfigOption::BlockInvest => {
            set_config!(global_config, block_invest, value);
        }
        GlobalConfigOption::BlockWithdraw => {
            set_config!(global_config, block_withdraw, value);
        }
        GlobalConfigOption::BlockCollectFees => {
            set_config!(global_config, block_collect_fees, value);
        }
        GlobalConfigOption::BlockCollectRewards => {
            set_config!(global_config, block_collect_rewards, value);
        }
        GlobalConfigOption::BlockSwapRewards => {
            set_config!(global_config, block_swap_rewards, value);
        }
        GlobalConfigOption::BlockSwapUnevenVaults => {
            set_config!(global_config, block_swap_uneven_vaults, value);
        }
        GlobalConfigOption::FeesBps => {
            set_config!(global_config, fees_bps, value);
        }
        _ => return Err(VaultError::GlobalConfigKeyError),
    }
```

- check user-provided `index/value`

```
/* yvaults/programs/yvaults/src/instructions/update_global_config.rs */
pub fn handler(
    ctx: Context<UpdateGlobalConfig>,
    key: u16,
    index: u16,
    value: &[u8; VALUE_BYTE_ARRAY_LEN],
) -> Result<()> {
    match key {
        GlobalConfigOption::SwapDiscountBps => {
            let collateral_id = index as usize;
            let discount = u64::from_le_bytes(value[..8].try_into().unwrap());
            global_config.swap_rewards_discount_bps[collateral_id] = discount;
            Ok(())
        }
        GlobalConfigOption::TreasuryFeeVaults => {
            let collateral_id = index as usize;
            let vault = Pubkey::new(value);
            global_config.treasury_fee_vaults[collateral_id] = vault; // range check
            Ok(())
        }
```

- check user-provided `value`

```
/* yvaults/programs/yvaults/src/instructions/update_strategy_config.rs */
pub fn handler(ctx: Context<UpdateStrategyConfig>, mode: u16, value: u64) -> ...{
    let mode = StrategyConfigOption::try_from(mode).unwrap();
    let strategy = &mut ctx.accounts.strategy.load_mut()?;
    match mode {
        StrategyConfigOption::UpdateDepositCap => strategy.deposit_cap_usd = value,
        StrategyConfigOption::UpdateDepositCapIxn =>
            strategy.deposit_cap_usd_per_ixn = value,
        StrategyConfigOption::UpdateWithdrawalCapACapacity => {
            strategy.withdrawal_cap_a.config_capacity = value as i64
        }
        StrategyConfigOption::UpdateWithdrawalCapAInterval => {
            strategy.withdrawal_cap_a.config_interval_length_seconds = value
        }
        StrategyConfigOption::UpdateWithdrawalCapACurrentTotal => {
            strategy.withdrawal_cap_a.current_total = value as i64
        }
```

```
            StrategyConfigOption::UpdateWithdrawalCapBCapacity => {
                strategy.withdrawal_cap_b.config_capacity = value as i64
            }
            StrategyConfigOption::UpdateWithdrawalCapBInterval => {
                strategy.withdrawal_cap_b.config_interval_length_seconds = value
            }
            StrategyConfigOption::UpdateWithdrawalCapBCurrentTotal => {
                strategy.withdrawal_cap_b.current_total = value as i64
            }
            StrategyConfigOption::UpdateMaxDeviationBps =>
                strategy.max_price_deviation_bps = value,
            StrategyConfigOption::UpdateSwapUnevenMaxSlippage => {
                strategy.swap_uneven_max_slippage = value
            }
            StrategyConfigOption::UpdateStrategyType => strategy.strategy_type = value,
            StrategyConfigOption::UpdateDepositFee => strategy.deposit_fee = value,
            StrategyConfigOption::UpdateWithdrawFee => strategy.withdraw_fee = value,
            StrategyConfigOption::UpdateCollectFeesFee => strategy.fees_fee = value,
            StrategyConfigOption::UpdateReward0Fee => strategy.reward_0_fee = value,
            StrategyConfigOption::UpdateReward1Fee => strategy.reward_1_fee = value,
            StrategyConfigOption::UpdateReward2Fee => strategy.reward_2_fee = value,
        }
}
```

IMPACT – INFO

[I-1] Questions

- when current_price = lower_tick_price, is it in range? It seems orca allows this scenario.

```
/* src/operations/vault_operations.rs */
pub fn assert_strategy_in_range(
    position: &Position,
    whirlpool: &Whirlpool,
    err: VaultError,
) -> VaultResult<()> {
    if !(lower_tick_price < current_price && current_price < upper_tick_price) {
        return Err(err);
    }
```

DISCLAIMER

The instance report ("Report") was prepared pursuant to an agreement between Coderrect Inc. d/b/a sec3 (the "Company") and Axiom Markets Ltd (the "Client"). This Report solely includes the results of a technical assessment of a specific build and/or version of the Client's code specified in the Report ("Assessed Code") by the Company. The sole purpose of the Report is to provide the Client with the results of the technical assessment of the Assessed Code. The Report does not apply to any other version and/or build of the Assessed Code. Regardless of the contents of the Report, the Report does not (and should not be interpreted to) provide any warranty, representation or covenant that the Assessed Code: (i) is error and/or bug free, (ii) has no security vulnerabilities, and/or (iii) does not infringe any third-party rights. Moreover, the Report is not, and should not be considered, an endorsement by the Company of the Assessed Code and/or of the Client. Finally, the Report should not be considered investment advice or a recommendation to invest in the Assessed Code and/or the Client.

This Report is considered null and void if the Report (or any portion thereof) is altered in any manner.

ABOUT

Founded by leading academics in the field of software security and senior industrial veterans, sec3 (formerly Soteria) is a leading blockchain security company that currently focuses on Solana programs. We are also building sophisticated security tools that incorporate static analysis, penetration testing, and formal verification.

At sec3, we identify and eliminate security vulnerabilities through the most rigorous process and aided by the most advanced analysis tools.

For more information, check out our website and follow us on twitter.

sec3™