Sudo Finance

Audit Report

MOVEBIT

contact@movebit.xyz

https://twitter.com/movebit_

Thu Feb 29 2024

Sudo Finance Audit Report

1 Executive Summary

1.1 Project Information

Description	Sudo Finance is a revolutionary on-chain real world asset & derivatives exchange protocol built on Sui
Type	Derivatives
Auditors	MoveBit
Timeline	Tue Oct 31 2023 - Thu Feb 29 2024
Languages	Move
Platform	Sui
Methods	Architecture Review, Unit Testing, Manual Review
Source Code	https://github.com/sudofina/sudo-contracts
Commits	6cc52574f30be908c786ea8b7461a0b12be2425d

1.2 Files in Scope

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

ID	File	SHA-1 Hash
MOV	sudo-core/Move.toml	2e9413ae139f4a8b1b1ddab82356b71479068df7
POS	sudo-core/sources/position.move	791ff25d9421b41d64de748c29f67855f4f7f0a5
ADM	sudo-core/sources/admin.move	22dc6ef703680bc1cbe21354974607919dfb4204
MOD	sudo-core/sources/model.move	6cf0aecea2de19836eec70a24c648ca3ad42e405
POO	sudo-core/sources/pool.move	c6d6ebf78f35c00d0fc90e7bf75792b99f0e55ad
APR	sudo-core/sources/agg_price.move	5c6be1f9a0348e80b6e7ac38c0564a3cfe9cd36b
MAR	sudo-core/sources/market.move	16e2194bfb03e15eaec96f3b95f3144a638f9af0
REF	sudo-core/sources/referral.move	3591cb342a3bf7075164585d4b39a39b48c19d8f
RAT	sudo-core/sources/math/rate.move	2358f1e7392d47421883c657eab927b6585132ab
SRA	sudo-core/sources/math/srate.move	027c9d34d31d8d951b088c3dddaf25653de333c6
DEC	sudo-core/sources/math/decimal.move	d137113cfa12eb50272ea1c4b11f1029e489bf2f

SDE	sudo-core/sources/math/sdecimal.move	0d6ebf83ad4febccbca087369547bebb0093e2b2
SLP	sudo-core/sources/slp.move	db766112bf08a58b24a2fdef87557728a9016b17

1.3 Issue Statistic

Item	Count	Fixed	Acknowledged
Total	15	12	3
Informational	0	0	0
Minor	4	2	2
Medium	3	2	1
Major	7	7	0
Critical	1	1	0

1.4 MoveBit Audit Breakdown

MoveBit aims to assess repositories for security-related issues, code quality, and compliance with specifications and best practices. Possible issues our team looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Integer overflow/underflow by bit operations
- Number of rounding errors
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting
- Unchecked CALL Return Values
- The flow of capability
- Witness Type

1.5 Methodology

The security team adopted the **"Testing and Automated Analysis"**, **"Code Review"** and **"Formal Verification"** strategy to perform a complete security test on the code in a way that is closest to the real attack. The main entrance and scope of security testing are stated in the conventions in the "Audit Objective", which can expand to contexts beyond the scope according to the actual testing needs. The main types of this security audit include:

(1) Testing and Automated Analysis

Items to check: state consistency / failure rollback / unit testing / value overflows / parameter verification / unhandled errors / boundary checking / coding specifications.

(2) Code Review

The code scope is illustrated in section 1.2.

(3) Formal Verification

Perform formal verification for key functions with the Move Prover.

(4) Audit Process

- Carry out relevant security tests on the testnet or the mainnet;
- If there are any questions during the audit process, communicate with the code owner in time. The code owners should actively cooperate (this might include providing the latest stable source code, relevant deployment scripts or methods, transaction signature scripts, exchange docking schemes, etc.);
- The necessary information during the audit process will be well documented for both the audit team and the code owner in a timely manner.

2 Summary

This report has been commissioned by Sudo Finance to identify any potential issues and vulnerabilities in the source code of the Sudo Finance smart contract, as well as any contract dependencies that were not part of an officially recognized library. In this audit, we have utilized various techniques, including manual code review and static analysis, to identify potential vulnerabilities and security issues.

During the audit, we identified 15 issues of varying severity, listed below.

ID	Title	Severity	Status
APR-1	The Price Obtained by `parse_pyth_feeder()` May not be Accurate	Medium	Fixed
MAR-1	Malicious Actors Can Exploit Outdated Valuations to Siphon Funds From The Protocol	Critical	Fixed
MAR-2	A Single Oracle Can Potentially Disrupt Protocol Operations	Major	Fixed
MAR-3	No Restrictions are Imposed When Creating Limit Orders.	Major	Fixed
MAR-4	`fee` Lacks Verification	Major	Fixed
MAR-5	The Checks for `delta_rate.value` And `vault.reserved_amount` Being 0 Are Missing	Medium	Fixed
MAR-6	Unused Private Function `burn_lp`	Minor	Fixed
MAR-7	Lack of Validation in Function Parameters When Adding `Vault` and `Symbol`	Minor	Acknowledged

MAR-8	Sensitive Operation Lacks Event	Minor	Fixed
MOV-1	Using an Excessively Outdated Version of the Sui Framework May Introduce Some Security Issues	Medium	Acknowledged
POO-1	Missing Disabled Functionality	Major	Fixed
POO-2	Swap Out with 0 Fee	Major	Fixed
POO-3	Avoid Redundant Operations for Rebate Rate of 0	Minor	Acknowledged
POS-1	Redemption of Collateral Lacking Verification	Major	Fixed
REF-1	Unable To Add Referral And Refresh `rebate_rate`	Major	Fixed

3 Participant Process

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

Admin

- Admin can create a new Vault through `add_new_vault()` or `add_new_vault_v1_1()` .
- Admin can create a new Symbol through `add_new_symbol()` or `add_new_symbol_v1_1()` .
- Admin can replace symbol feeder through `replace_symbol_feeder()` .
- Admin can add a new type of collateral for the Symbol through `add_collateral_to_symbol()` .
- Admin can remove a type of collateral for the Symbol through `remove_collateral_from_symbol()` .
- Admin can set symbol status through `set_symbol_status()` .
- Admin can replace position config through `replace_position_config()` .

User

- User can open a new position or pending order open a new position through `open_position()` or `open_position_v1_1()` .
- User can decrease their position through `decrease_position()` or `decrease_position_v1_1()` .
- User can decrease their reserve amount in the position through `decrease_reserved_from_position()` .
- User can add collateral to their position through `pledge_in_position()` .
- User can redeem collateral from their position through `redeem_from_position()` or `redeem_from_position_v1_1()` .
- User can clear the closed position through `clear_closed_position()` .
- User can clear the open position order through `clear_open_position_order()` or `clear_open_position_order_v1_1()` .
- User can clear the decrease position order through `clear_decrease_position_order()` or `clear_decrease_position_order_v1_1()` .

- User can deposit `C` coin into the vault to inject liquidity and get `SLP` coin through `deposit()`.
- User can burn the `SLP` coin to withdraw the liquidity an get the `C` coin through `withdraw()`.
- User can swap the `S` coin and get the `D` coin in the vault through `swap<L, S, D>()`.

Order Executor

- Order Excutor can execute the open position order through `execute_open_position_order()` or `execute_open_position_order_v1_1()`.
- Order Excutor can execute the decrease position order through `execute_decrease_position_order()` or `execute_decrease_position_order_v1_1()`.

Liquidator

- Liquidator can liquidate the position through `liquidate_position()` or `liquidate_position_v1_1()`.

4 Findings

APR-1 The Price Obtained by `parse_pyth_feeder()` May not be Accurate

Severity: Medium

Status: Fixed

Code Location:

sudo-core/sources/agg_price.move#58-62

Descriptions:

This function, `agg_price.parse_pyth_feeder()`, is responsible for parsing a Pyth feeder's data to create an `AggPrice` struct, which represents an aggregated price. The line of code `pyth_price::get_timestamp(&price) + config.max_interval >= timestamp` only considers the case where the timestamp in the price data (`get_timestamp(&price)`) is less than the current timestamp. However, it does not account for situations where the price data's timestamp is greater than the current timestamp. This omission could lead to the retrieval of inaccurate prices.

```
public fun parse_pyth_feeder(
    config: &AggPriceConfig,
    feeder: &PythFeeder,
    timestamp: u64,
  ): AggPrice {
    assert!(object::id(feeder) == config.feeder, ERR_INVALID_PRICE_FEEDER);

    let price = get_price_unsafe(feeder);
    assert!(
      pyth_price::get_timestamp(&price) + config.max_interval >= timestamp,
      ERR_PRICE_STALED,
    );
```

As a best practice, it's important to consider both cases: comparing the absolute difference between the two timestamps (the current timestamp and the price data's timestamp) to ensure that it falls within the acceptable range (`config.max_interval`). This approach would

provide a more robust validation of the price data, regardless of whether the price data is newer or older than the current timestamp.

```
fun check_price_is_fresh(price: &Price, clock: &Clock, max_age_secs: u64) {
    let age = abs_diff(clock::timestamp_ms(clock)/1000, price::get_timestamp(price));
    assert!(age < max_age_secs, E_STALE_PRICE_UPDATE);
}

fun abs_diff(x: u64, y: u64): u64 {
    if (x > y) {
        return x - y
    } else {
        return y - x
    }
}
```

Suggestion:

It is recommended to compare the absolute difference between the two timestamps (the current timestamp and the price data's timestamp) to ensure that it falls within the acceptable range.

Resolution:

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

MAR-1 Malicious Actors Can Exploit Outdated Valuations to Siphon Funds From The Protocol

Severity: Critical

Status: Fixed

Code Location:

sudo-core/sources/market.move#1482-1536

Descriptions:

The `market.deposit()` function facilitates LP token minting for users in a financial market protocol. Prior to calling this function, it is essential to create `VaultsValuation` and `SymbolsValuation` instances, updating relevant information with `valuate_vault()` and `valuate_symbol()`. Within the deposit function, the protocol invokes `pool.deposit()` to calculate mint_amount.

```
let mint_amount = pool::deposit(
        vault,
        model,
        &price,
        coin::into_balance(deposit),
        min_amount_out,
        lp_supply_amount,
        market_value,
        vault_value,
        total_vaults_value,
        total_weight,
    );

    // mint to sender
    let minted = mint_lp(market, mint_amount);
    pay_from_balance(minted, minter, ctx);
```

This calculation involves obtaining the `exchange_rate` by dividing `deposit_value` by `market_value` and then multiplying `lp_supply_amount` by the resulting `exchange_rate` to determine mint_amount.

```
let mint_amount = if (lp_supply_amount == 0) {
        assert!(decimal::is_zero(&market_value), ERR_UNEXPECTED_MARKET_VALUE);
        truncate_decimal(deposit_value)
    } else {
        assert!(!decimal::is_zero(&market_value), ERR_UNEXPECTED_MARKET_VALUE);
        let exchange_rate = decimal::to_rate(
            decimal::div(deposit_value, market_value)
        );
        decimal::floor_u64(
            decimal::mul_with_rate(
                decimal::from_u64(lp_supply_amount),
                exchange_rate,
            )
        )
    };
```

A potential exploit arises when malicious actors separately create two sets of `VaultsValuation` and `SymbolsValuation`. They first use one set as parameter for the deposit function, obtaining LP tokens. Subsequently, they employ the other set as parameters for another call to the deposit function. As `lp_supply_amount` increases after the initial deposit, the second set of `VaultsValuation` and `SymbolsValuation` retains outdated information. Consequently, during the second deposit, the calculation of `mint_amount` results in an elevated `exchange_rate`, leading to a larger `mint_amount`. This mismatch between the minted LP tokens and the actual value poses a risk of protocol losses.

Suggestion:

It is recommended to introduce a locking mechanism within the `create_vaults_valuation()` function, and subsequently, unlock it during the execution of `finalize_vaults_valuation()`.

Resolution:

This issue has been fixed. The client implemented a locking mechanism.

MAR-2 A Single Oracle Can Potentially Disrupt Protocol Operations

Severity: Major

Status: Fixed

Code Location:

sudo-core/sources/market.move#489-493

Descriptions:

The `open_position()` function is a complex operation that handles both limited orders and immediate trades within a market. As shown in the code snippet below, the protocol obtains price data from the Pyth oracle.

```
let index_price = agg_price::parse_pyth_feeder(
    pool::symbol_price_config(symbol),
    index_feeder,
    timestamp,
);
```

However, it's important to acknowledge that the Pyth oracle, like any external data source, can encounter issues or be subject to manipulation. If the Pyth oracle experiences exceptions or tampering, it can lead to incorrect price data being retrieved. This, in turn, can disrupt the protocol's functionality and potentially lead to undesirable outcomes.

Suggestion:

It is recommended to utilize dual oracles to enable failover between them and to cross-verify prices, ensuring accuracy within the correct range.

Resolution:

This issue has been fixed. The protocol is now using Pyth v1 and the Wormhole Oracle.

MAR-3 No Restrictions are Imposed When Creating Limit Orders.

Severity: Major

Status: Fixed

Code Location:

sudo-core/sources/market.move#500-538

Descriptions:

The function `market.open_position()` is responsible for managing the creation of positions in a market based on certain conditions and parameters. It handles both limited orders and immediate trading.

```
let is_limited = if (long) {
        decimal::gt(
            &agg_price::price_of(&index_price),
            &agg_price::price_of(&limited_index_price),
        )
    } else {
        decimal::lt(
            &agg_price::price_of(&index_price),
            &agg_price::price_of(&limited_index_price),
        )
    };

    if (is_limited) {
        assert!(!allow_trade, ERR_CAN_NOT_CREATE_LIMIT_ORDER);

        let order = OpenPositionOrder<C, I, D, F> {
            id: object::new(ctx),
            executed: false,
            owner,
            open_amount,
            reserve_amount,
            limited_index_price,
            collateral_price_threshold,
            position_config: position_config.inner,
            collateral: coin::into_balance(collateral),
            fee: coin::into_balance(fee),
```

```
    };
    let order_id = object::uid_to_inner(&order.id);

    transfer::transfer(
        OrderCap<C, I, D> { id: object::new(ctx), order_id },
        owner,
    );

    transfer::share_object(order);

    // emit order created
    event::emit(OrderCreated { order_id });
}
```

The issue at hand is that the system's validation for creating limit orders only checks whether the `index_price` is greater than the `limited_index_price`. Malicious actors can create numerous limit orders with minimal or zero collateral that are unlikely to be executed. The server processes these limit orders and displays them on the user interface (UI). If a substantial number of these "garbage" orders are created, it can potentially strain the server's resources and impact its performance.

Suggestion:

It is recommended to establish a minimum collateral amount or impose a fee when creating limit orders or limit the number of limit orders an account can create to mitigate this risk.

Resolution:

The issue has been addressed in the `open_position_v1_2()` function by adding a minimum fee payable with SUI.

MAR-4 `fee` Lacks Verification

Severity: Major

Status: Fixed

Code Location:

sudo-core/sources/market.move#470

Descriptions:

In the `open_position` and `decrease_position` functions, the `fee` is used as the execution fee without undergoing any validation. This means that users can input any type of coin as the fee, including potentially fake or arbitrarily minted coins. This lack of validation poses a risk to the security of the executor's rights, as there is no assurance regarding the legitimacy of the provided fee. Additionally, the absence of constraints on the size of the fee results in varying execution fees for each order, which is also unreasonable.

```
pay_from_balance(
    balance::withdraw_all(&mut order.fee),
    executor,
    ctx,
);
```

Suggestion:

It is recommended to implement validation checks for both the type of coin and the size of the fee to ensure security and consistency.

Resolution:

This issue has been fixed. The client has already added the checks for the fee.

MAR-5 The Checks for `delta_rate.value` And `vault.reserved_amount` Being 0 Are Missing

Severity: Medium

Status: Fixed

Code Location:

sudo-core/sources/market.move#167-183

Descriptions:

Within the `refresh_vault()` function , it calculates the `delta_rate` by calling the function `vault_delta_reserving_rate()` with parameters such as the vault, reserving fee model, supply amount, and timestamp

```
public fun vault_delta_reserving_rate<C>(
    vault: &Vault<C>,
    reserving_fee_model: &ReservingFeeModel,
    supply_amount: Decimal,
    timestamp: u64,
  ): Rate {
    if (vault.last_update > 0) {
      let elapsed = timestamp - vault.last_update;
      if (elapsed > 0) {
        return model::compute_reserving_fee_rate(
          reserving_fee_model,
          vault_utilization(vault, supply_amount),
          elapsed,
        )
      }
    };
    rate::zero()
  }
```

In the given code, if the `vault.reserved_amount` is 0, the calculated utilization `rate.value` becomes 0.

```
public fun vault_delta_reserving_rate<C>(
    vault: &Vault<C>,
    reserving_fee_model: &ReservingFeeModel,
```

```
    supply_amount: Decimal,
    timestamp: u64,
): Rate {
    if (vault.last_update > 0) {
        let elapsed = timestamp - vault.last_update;
        if (elapsed > 0) {
            return model::compute_reserving_fee_rate(
                reserving_fee_model,
                vault_utilization(vault, supply_amount),
                elapsed,
            )
        }
    };
    rate::zero()
}
```

Consequently, the result obtained from `vault_delta_reserving_rate` is also 0. As a result, both `vault.acc_reserving_rate` and `vault.unrealised_reserving_fee_amount` are incremented by 0.

```
let delta_rate = vault_delta_reserving_rate(
        vault,
        reserving_fee_model,
        supply_amount,
        timestamp,
    );
    vault.acc_reserving_rate = vault_acc_reserving_rate(vault, delta_rate);
    vault.unrealised_reserving_fee_amount =
        vault_unrealised_reserving_fee_amount(vault, delta_rate);
    vault.last_update = timestamp;
```

This scenario essentially leads to the reserving rate and fee calculations being updated without any actual change when the reserved amount is zero.

Suggestion:

It is recommended to include checks for both `delta_rate` being 0 and `vault.reserved_amount` being 0.

```
if (vault.last_update > 0 && vault.reserved_amount > 0) {
        let elapsed = timestamp - vault.last_update;
        if (elapsed > 0) {
            return model::compute_reserving_fee_rate(
                reserving_fee_model,
                vault_utilization(vault, supply_amount),
                elapsed,
            )
        }
    };
```

```
if (delta_rate.value > 0)
{
   vault.acc_reserving_rate = vault_acc_reserving_rate(vault, delta_rate);
      vault.unrealised_reserving_fee_amount =
         vault_unrealised_reserving_fee_amount(vault, delta_rate);

}
```

Resolution:

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

MAR-6 Unused Private Function `burn_lp`

Severity: Minor

Status: Fixed

Code Location:

sudo-core/sources/market.move#235-237

Descriptions:

The private function `burn_lp` is intended for burning LP tokens, but it is not utilized in the contract.

```
fun burn_lp<L>(market: &mut Market<L>, token: Balance<L>): u64 {
    balance::decrease_supply(&mut market.lp_supply, token)
}
```

Suggestion:

It is recommended to either delete this function or replace the following code in the `withdraw` function with the `burn_lp` function.

```
    // burn LP
    let burn_amount = balance::decrease_supply(
        &mut market.lp_supply,
        coin::into_balance(burn),
    );
```

Resolution:

This issue has been partially fixed. The client has removed the `burn_lp` function.

MAR-7 Lack of Validation in Function Parameters When Adding `Vault` and `Symbol`

Severity: Minor

Status: Acknowledged

Code Location:

sudo-core/sources/market.move#354,381

Descriptions:

Through the function `add_new_symbol()`, there is a lack of validation for some parameters of type `u64` when adding a `Symbol`.

For example, if the parameter `max_reserved_multiplier` is equal to 0, executing `open_position()` for position opening will fail. Although the `add_new_symbol()` function is an administrative operation, it should still be restricted in the code to avoid unexpected situations.

Other parameters, as well as the parameters of the `add_new_vault()` function, should also be checked for the need for validation.

Suggestion:

It is recommended to add appropriate validation for function parameters.

MAR-8 Sensitive Operation Lacks Event

Severity: Minor

Status: Fixed

Code Location:

sudo-core/sources/market.move#354,381,437,449

Descriptions:

In the contract, some sensitive operations lack event listeners, making it difficult for external tracking of changes in related data within the contract.

The functions affected by this issue include `add_new_vault()`, `add_new_symbol()`, `add_collateral_to_symbol()`, and `remove_collateral_from_symbol()`.

Suggestion:

It is recommended to add events similar to other operations to facilitate monitoring changes within the contract.

Resolution:

This issue has been fixed. The client has added the corresponding event.

MOV-1 Using an Excessively Outdated Version of the Sui Framework May Introduce Some Security Issues

Severity: Medium

Status: Acknowledged

Code Location:

sudo-core/Move.toml#9

Descriptions:

From the `move.toml` file, we can see that the protocol is currently using Sui v1.13.2, while the latest version is Sui v1.15.1. Numerous security issues have been addressed during this period, and it is recommended to upgrade to the latest version.

Suggestion:

It is recommended to use the latest version of the Sui framework.

POO-1 Missing Disabled Functionality

Severity: Major

Status: Fixed

Code Location:

sudo-core/sources/pool.move#205-242

Descriptions:

When creating a new `Vault` and `Symbol`, `enabled`, `open_enabled`, `decrease_enabled`, and `liquidate_enabled` are all initialized as `true`. During opening, decreasing, and liquidating, it checks whether these are enabled. However, the contract lacks the `disabled` functionality, which means that it's not possible to disable already created Vaults and Symbols. As a result, the checks for opening, decreasing and liquidating always pass.

```
public(friend) fun new_vault<C>(
    weight: u256,
    model_id: ID,
    price_config: AggPriceConfig,
): Vault<C> {
    Vault {
        enabled: true,
        weight: decimal::from_raw(weight),
        reserving_fee_model: model_id,
        price_config,
        last_update: 0,
        tax: balance::zero(),
        liquidity: balance::zero(),
        reserved_amount: 0,
        unrealised_reserving_fee_amount: decimal::zero(),
        acc_reserving_rate: rate::zero(),
    }
}

public(friend) fun new_symbol(
    model_id: ID,
    price_config: AggPriceConfig,
): Symbol {
    Symbol {
```

```
        open_enabled: true,
        decrease_enabled: true,
        liquidate_enabled: true,
        supported_collaterals: vec_set::empty(),
        funding_fee_model: model_id,
        price_config,
        last_update: 0,
        opening_amount: 0,
        opening_size: decimal::zero(),
        realised_pnl: sdecimal::zero(),
        unrealised_funding_fee_value: sdecimal::zero(),
        acc_funding_rate: srate::zero(),
    }
}
```

Suggestion:

It is recommended to add a `disabled` functionality for Vaults and Symbols.

Resolution:

This issue has been fixed. The client has added a `set_symbol_status` and `set_vault_status` function.

POO-2 Swap Out with 0 Fee

Severity: Major

Status: Fixed

Code Location:

sudo-core/sources/pool.move#413-420

Descriptions:

The `pool.swap_out()` is used for performing an asset swap operation, where assets are taken out from a target vault known as `dest_vault`. Inside the function, it calculates the fee rate for the destination vault, using the `compute_rebase_fee_rate` function.

```
let dest_fee_rate = compute_rebase_fee_rate(
        model,
        false,
        decimal::sub(dest_vault_value, swap_value),
        total_vaults_value,
        dest_vault.weight,
        total_weight,
    );
```

However, when `dest_vault_value` is equal to `swap_value`, the `dest_fee_rate` becomes 0, resulting in a fee of 0.

Suggestion:

It is recommended to not subtract `swap_value` from `dest_vault_value` initially, and likewise, when performing a `swap_in()`, avoid adding `source_vault_value` and `total_vaults_value` initially.

Resolution:

This issue has been fixed. The client made adjustments to the code.

POO-3 Avoid Redundant Operations for Rebate Rate of 0

Severity: Minor

Status: Acknowledged

Code Location:

sudo-core/sources/pool.move#548-554

Descriptions:

In `pool.open_position()`, when the owner has no referrer (resulting in `get_referral_data()` returning a 0 address and a 0 rate), the protocol calculates `rebate_amount` based on `rebate_rate`. However, when these values are 0, the calculation of re`bate_amount` and the subsequent withdrawal from `vault.liquidity` are unnecessary.

```
    // compute rebate
     let rebate_amount =
decimal::floor_u64(decimal::mul_with_rate(open_fee_amount_dec, rebate_rate));

     // update vault
     vault.reserved_amount = vault.reserved_amount + reserve_amount;
     let liquidity_amount = balance::join(&mut vault.liquidity, open_fee);
     assert!(liquidity_amount > rebate_amount, ERR_INSUFFICIENT_LIQUIDITY);
     let rebate = balance::split(&mut vault.liquidity, rebate_amount);
```

Suggestion:

It is recommended to add a check for `rebate_rate` being 0 in both the "compute rebate" and the "execute update vault" operation.

POS-1 Redemption of Collateral Lacking Verification

Severity: Major

Status: Fixed

Code Location:

sudo-core/sources/position.move#450

Descriptions:

Lack of verification for the following conditions when redeeming collateral:

```
balance::value(collateral) * config.max_reserved_multiplier >= reserve_amount
```

If the above conditions are not met, the transaction should be terminated.

Suggestion:

It is recommended to add this verification.

Resolution:

This issue has been fixed. The client has added corresponding validation.

REF-1 Unable To Add Referral And Refresh `rebate_rate`

Severity: Major

Status: Fixed

Code Location:

sudo-core/sources/referral.move#12-24

Descriptions:

The `new_referral` and `refresh_rebate_rate` functions are set as `public(friend)` but are not called anywhere in the `market` contract. The absence of this functionality results in the 'referrals' always being empty. Moreover, it has been consistently unable to update `Referral` and refresh `rebate_rate`.

```
public(friend) fun new_referral(
    referrer: address,
    rebate_rate: Rate,
): Referral {
    Referral { referrer, rebate_rate }
}

public(friend) fun refresh_rebate_rate(
    referral: &mut Referral,
    rebate_rate: Rate,
) {
    referral.rebate_rate = rebate_rate;
}
```

Suggestion:

It is recommended to add this functionality in the `market` contract to call `new_referral` and `refresh_rebate_rate`.

Resolution:

This issue has been partially fixed. The client added a call to the `new_referral` function.

Appendix 1

Issue Level

- **Informational** issues are often recommendations to improve the style of the code or to optimize code that does not affect the overall functionality.
- **Minor** issues are general suggestions relevant to best practices and readability. They don't post any direct risk. Developers are encouraged to fix them.
- **Medium** issues are non-exploitable problems and not security vulnerabilities. They should be fixed unless there is a specific reason not to.
- **Major** issues are security vulnerabilities. They put a portion of users' sensitive information at risk, and often are not directly exploitable. All major issues should be fixed.
- **Critical** issues are directly exploitable security vulnerabilities. They put users' sensitive information at risk. All critical issues should be fixed.

Issue Status

- **Fixed:** The issue has been resolved.
- **Partially Fixed:** The issue has been partially resolved.
- **Acknowledged:** The issue has been acknowledged by the code owner, and the code owner confirms it's as designed, and decides to keep it.

Appendix 2

Disclaimer

This report is based on the scope of materials and documents provided, with a limited review at the time provided. Results may not be complete and do not include all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your own risk. A report does not imply an endorsement of any particular project or team, nor does it guarantee its security. These reports should not be relied upon in any way by any third party, including for the purpose of making any decision to buy or sell products, services, or any other assets. TO THE FULLEST EXTENT PERMITTED BY LAW, WE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, IN CONNECTION WITH THIS REPORT, ITS CONTENT, RELATED SERVICES AND PRODUCTS, AND YOUR USE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, NOT INFRINGEMENT.