Competitive Security Assessment

Pinnako

Jun 14th, 2023

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.
- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.
- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.
- Verify the code base is compliant with the most up-to-date industry standards and security best practices.
- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

Project Detail

Project Name	Pinnako
Platform & Language	Solidity
Codebase	• https://github.com/pinnakoex/pinnakoes_contract • audit commit - b95ec9c57103b89ef9044c4e17ca6d529cbf739d • final commit - b6f744624aab9dc89ad628eb3e9c8a112d30b87c
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

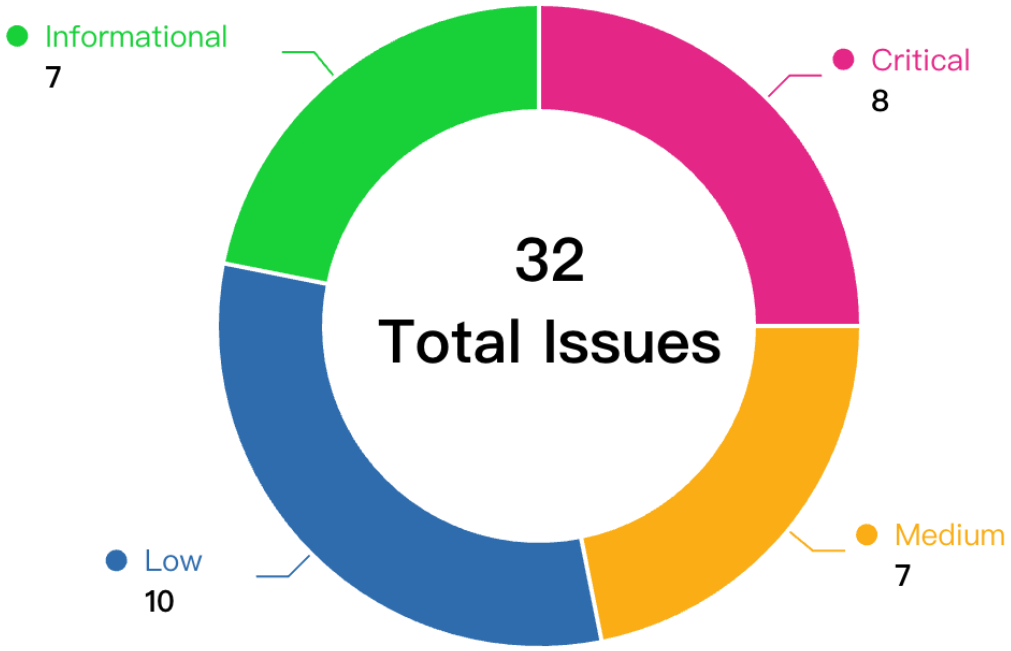
Code Vulnerability Review Summary

Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	8	0	0	7	1	0
Medium	7	0	2	5	0	0
Low	10	0	0	10	0	0
Informational	7	0	1	6	0	0

Audit Scope

File	Commit Hash
contracts/core/OrderBook.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/PositionRouter.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/Vault.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/VaultUtils.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/Router.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/RouterSign.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/BasePositionManager.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/LpManager.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/PositionManager.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/VaultStorage.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/interfaces/IVaultUtils.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/VaultMSData.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/interfaces/IVault.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/interfaces/IOrderBook.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/interfaces/IPositionRouter.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/interfaces/IVaultStorage.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/interfaces/IFeeRouter.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/interfaces/IRouter.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/interfaces/IBasePositionManager.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d
contracts/core/interfaces/ILpManager.sol	b95ec9c57103b89ef9044c4e17ca6d529cbf739d

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
PKO-1	Incorrect formula used in function `VaultUtils.getPositionNextAveragePrice()`	Logical	Critical	Fixed	TrungOre
PKO-2	Logic error in `VaultPriceFeedV3Fast` contract `getPythPrice` function	Logical	Critical	Fixed	w2ning
PKO-3	Logical error:Inconsistent token address used when adding/removing eth liquidity in `LpManager.sol`	Logical	Critical	Fixed	w2ning, TrungOre
PKO-4	Potential Liquidity Exhaustion Risk	Logical	Critical	Fixed	Kong7ych3

PKO-5	The average price must not be updated when function `Vault._decreasePosition()` is called	Logical	Critical	Fixed	TrungOre
PKO-6	The implementation of the `premiumFee` in the `Vault._collectMarginFees` function can potentially result in a loss of funds for users.	Logical	Critical	Mitigated	TrungOre
PKO-7	Underflow error when calling `VaultUtils.getNextAveragePrice()` can prevent users from dercreasing the position	Integer Overflow and Underflow	Critical	Fixed	TrungOre
PKO-8	missing _mintOut protection can lead to user lose funds.	Logical	Critical	Fixed	jayphbee
PKO-9	Incorrect daily seconds used.	Logical	Medium	Fixed	jayphbee
PKO-10	Open a position without updating the price to increase the winning rate	Race condition	Medium	Fixed	Kong7ych3
PKO-11	Some tokens will be lost when function `Router.directPoolDeposit()` is called	Logical	Medium	Fixed	TrungOre
PKO-12	User can' t be guaranteed to decrease his position.	Privilege Related	Medium	Fixed	jayphbee
PKO-13	User's position can become liquidatable because system parameter change.	Privilege Related	Medium	Acknowledged	jayphbee
PKO-14	Weak Sources of Randomness in `randomSource::_seed`	Weak Sources of Randomness	Medium	Acknowledged	w2ning
PKO-15	updateRate not performed after buyUSD operation	Logical	Medium	Fixed	Kong7ych3, TrungOre
PKO-16	Any `msg.value` exceeding `_executionFee` should be return	Logical	Low	Fixed	Kong7ych3
PKO-17	Global variable `psbtLogic` can never be changed in `PSBT` contract	Logical	Low	Fixed	w2ning
PKO-18	Logic error in `Array` contract `get` function	Logical	Low	Fixed	w2ning

PKO-19	Logical error: `tradingTax` miss a set function	Logical	Low	Fixed	w2ning
PKO-20	Missing limit in `RouterSign::initialization`	Code Style	Low	Fixed	Kong7ych3
PKO-21	Precision issue:Multiplication before division	Logical	Low	Fixed	w2ning, jayphbee
PKO-22	Risk of owner excessive privilege	Privilege Related	Low	Fixed	Kong7ych3
PKO-23	addLiquidity should restrict msg.value == 0 when _token is not native.	Logical	Low	Fixed	jayphbee
PKO-24	createIncreasePosition and createDecreasePosition should have strict msg.value check.	Logical	Low	Fixed	jayphbee
PKO-25	setMaxGlobalSizes function lack array length equality check for its parameters.	Logical	Low	Fixed	jayphbee
PKO-26	BasePositionManager - no params value check in `setMaxGlobalSizes` function	Code style	Informational	Fixed	zeroxvee
PKO-27	Consider creating structs for return types on `OrderBook.getIncreaseOrder`, `OrderBook.getDecreaseOrder`, and `OrderBook.getSwapOrder`	Code Style	Informational	Fixed	0xgm
PKO-28	Gas Optimization: Unused function	Gas Optimization	Informational	Fixed	jayphbee
PKO-29	Gas Optimization:SafeMath is unnecessary after solidity 0.8.0	Gas Optimization	Informational	Acknowledged	jayphbee
PKO-30	LpManager - redundant checks in `setLP()` and `delLP()` functions	Gas Optimization	Informational	Fixed	zeroxvee
PKO-31	Missing event record	Code Style	Informational	Fixed	Kong7ych3
PKO-32	Redundant payable tag	Code Style	Informational	Fixed	Kong7ych3, TrungOre

PKO-1:Incorrect formula used in function `VaultUtils.getPositionNextAveragePrice()`

Category	Severity	Status	Contributor
Logical	Critical	Fixed	TrungOre

Code Reference

- code/contracts/core/VaultUtils.sol#L378-L388
- code/contracts/core/VaultUtils.sol#L401-L408

```
378:    function getNextAveragePrice(uint256 _size, uint256 _averagePrice,  uint256 _nextPrice, uint
256 _sizeDelta, bool _isIncrease) public pure override returns (uint256) {
379:        if (_size == 0) return _nextPrice;
380:        if (_isIncrease){
381:            uint256 nextSize = _size.add(_sizeDelta) ;
382:            return nextSize > 0 ? (_averagePrice.mul(_size)).add(_sizeDelta.mul(_nextPrice)).div
(nextSize) : 0;
383:        }
384:        else{
385:            uint256 _latestSize = _size > _sizeDelta ? _size.sub(_sizeDelta) : 0;
386:            return _latestSize > 0 ? (_averagePrice.mul(_size).sub(_sizeDelta.mul(_nextPrice))).
div(_latestSize): 0;
387:        }
388:    }

401:    function getPositionNextAveragePrice(uint256 _size, uint256 _averagePrice, uint256 _nextPric
e, uint256 _sizeDelta, bool _isIncrease) public override pure returns (uint256) {
402:        if (_isIncrease)
403:            return (_size.mul(_averagePrice)).add(_sizeDelta.mul(_nextPrice)).div(_size.add(_siz
eDelta));
404:        else{
405:            require(_size >= _sizeDelta, "invalid size delta");
406:            return (_size.mul(_averagePrice)).sub(_sizeDelta.mul(_nextPrice)).div(_size.sub(_siz
eDelta));
407:        }
408:    }
```

Description

TrungOre : To facilitate description, we use the notation `position = (size, averagePrice)` to represent each position, where `size` and `averagePrice` are two parameters.

Let's assume that the sender already possesses a position `(s1, p1)` and intends to increase the position's size by `s2` at price `p2`. The function VaultUtils.getPositionNextAveragePrice() aims to generate a new price `p_n` such that, regardless of the spot price `p3` at any given time, the profit/loss resulting from two separate positions `(p1, s1)` and `(p2, s2)` is equal to the profit/loss generated by the position `(p_n, s1 + s2)`. In other words, the following equation must hold true for every spot price `p3`:

```
    (p3 - p1) * s1 / p1 + (p3 - p2) * s2 / p2 = (p3 - p_n) * (s1 + s2) / p_n
<=> p_n = p1 * p2 * (s1 + s2) / (s1 * p2 + s2 * p1)
```

The new average price derived from the above equation differs from the one depicted in `VaultUtils.getPositionNextAveragePrice()`, which is `p_n = (p1 * s1 + p2 * s2) / (s1 + s2)`. Specifically, `p_n` is smaller than the value returned by the `VaultUtils.getPositionNextAveragePrice()` function. As a result, this discrepancy leads to higher-than-expected profits (if the position is profitable) and reduced losses (if the position incurs a loss).

A similar situation arises in the `VaultUtils.getNextAveragePrice()` function when calculating the average price for all positions in the vault. This discrepancy has a more significant impact when calculating profits/losses, especially when calling the `LpManager.getAum()` function, which results in incorrect calculations for the shares minted/burned for the liquidity provider.

Recommendation

TrungOre : The formula for calculating the next average price should be modified as follows:

```
p_n = p1 * p2 * (s1 + s2) / (s1 * p2 + s2 * p1)
```

Client Response

Fixed, Equation accepted to calculate average price.

PKO-2:Logic error in `VaultPriceFeedV3Fast` contract `getPythPrice` function

Category	Severity	Status	Contributor
Logical	Critical	Fixed	w2ning

Code Reference

- code/contracts/oracle/VaultPriceFeedV3Fast.sol#L364

```
364:                read_state = false;
```

Description

w2ning : If (price < 10 || upd_time < 10), variable `read_state` assigned as false, but finally read_state would be changed as true.

Consider below POC contract

```
function getPythPrice(address _token) public view returns(uint256, bool, uint256){
    if (address(pyth) == address(0)) {
        return (0, false, 0);
    }

    if (tokenPythKEY[_token] == bytes32(0)) {
        return (0, false, 1);
    }

    uint256 price = 0;
    bool read_state = false;
    uint256 upd_time = 5;
    try pyth.getPriceUnsafe(tokenPythKEY[_token]) returns (PythStructs.Price memory _pyPrice ) {
        uint256 it_price = uint256(int256(_pyPrice.price));
        if (it_price < 1) {
            return (0, false, 2);
        }
        // 3040137682421, 825317579, -8,
        upd_time = uint256(_pyPrice.publishTime);
        uint256 time_interval = uint256(block.timestamp).sub(upd_time);
        if (time_interval > priceSafetyGap) {
            return (0, false, 3);
        }
        int256 _expo= int256(_pyPrice.expo);
        if (_expo >= 0) {
            return (0, false, 4);
        }
        price = uint256(it_price).mul(PRICE_PRECISION).div(10 ** uint256(-_expo));
        if (price < 10 || upd_time < 10) {
            // Variable read_state assigned as false
            read_state = false;
            upd_time = 7;
        }
        // But finally read_state would be changed as true
        read_state = true;
    } catch {
        upd_time = 6;
    }

    return (price, read_state, upd_time);
}
```

Recommendation

w2ning : If (price < 10 || upd_time < 10), return variables directly

Consider below fix in the `VaultPriceFeedV3Fast.getPythPrice()` function

```
        try pyth.getPriceUnsafe(tokenPythKEY[_token]) returns (PythStructs.Price memory _pyPrice ) {
            uint256 it_price = uint256(int256(_pyPrice.price));
            if (it_price < 1) {
                return (0, false, 2);
            }
            // 3040137682421, 825317579, -8,
            upd_time = uint256(_pyPrice.publishTime);
            uint256 time_interval = uint256(block.timestamp).sub(upd_time);
            if (time_interval > priceSafetyGap) {
                return (0, false, 3);
            }
            int256 _expo= int256(_pyPrice.expo);
            if (_expo >= 0) {
                return (0, false, 4);
            }
            price = uint256(it_price).mul(PRICE_PRECISION).div(10 ** uint256(-_expo));
            if (price < 10 || upd_time < 10) {
                read_state = false;
                upd_time = 7;
                // fix: Return variables directly
                return (price, read_state, upd_time);
            }
            read_state = true;
        } catch {
            upd_time = 6;
        }

        return (price, read_state, upd_time);
```

Client Response

Fixed, Solved by using new priceFeed method(VaultPriceFeed.sol)

PKO-3:Logical error:Inconsistent token address used when adding/removing eth liquidity in `LpManager.sol`

Category	Severity	Status	Contributor
Logical	Critical	Fixed	w2ning, TrungOre

Code Reference

- code/contracts/core/LpManager.sol#L88
- code/contracts/core/LpManager.sol#L101-L111

```
88:         uint256 usdAmount = IVault(lpVault[_lp]).buyUSD(_token);

101:         address _tokenOut = _tokenOutOri==address(0) ? weth : _tokenOutOri;
102:         require(IVault(lpVault[_lp]).isFundingToken(_tokenOut), "[LpManager] not supported lp to
ken");
103:         address _account = msg.sender;
104:         IERC20(_lp).safeTransferFrom(_account, address(this), _lpAmount );
105:
106:         // calculate aum before sellUSD
107:         uint256 aumInUSD = getAumInUSD(_lp, false);
108:         uint256 lpSupply = IERC20(_lp).totalSupply();
109:         uint256 usdAmount = _lpAmount.mul(aumInUSD).div(lpSupply); //30b
110:         IMintable(_lp).burn(_lpAmount);
111:         uint256 amountOut = IVault(lpVault[_lp]).sellUSD(_tokenOut, address(this), usdAmount);
```

Description

w2ning : When `_FundToken` has been changed to weth, the following logic still uses input parameters `_token`.
This may lead to price calculation errors and potential economic losses

```
    function addLiquidity(address _lp, address _token, uint256 _amount, uint256 _minlp) external ove
rride payable nonReentrant returns (uint256) {
        require(isLpToken(_lp), "[LpManager] Not supported lp token" );
        require(IVault(lpVault[_lp]).isFundingToken(_token), "[LpManager] not supported lp token");
        {
            address _fundToken = _token;
            uint256 _fundAmount = _amount;
            if (_token == address(0)){
               // Here _FundToken has been changed to weth
                _fundToken = weth;
                _fundAmount = msg.value;
                IWETH(weth).deposit{value: msg.value}();
            }else{
                IERC20(_fundToken).safeTransferFrom(msg.sender, address(this), _fundAmount);
            }
            require(_fundAmount > 0, "[LpManager] invalid amount");
            IERC20(_fundToken).safeTransfer(lpVault[_lp], _fundAmount);
        }

        // calculate aum before buyUSD
        uint256 aumInUSD = getAumInUSD(_lp, true);
        uint256 lpSupply = IERC20(_lp).totalSupply();

        // Here still use the input parameter _token
        uint256 usdAmount = IVault(lpVault[_lp]).buyUSD(_token);
        uint256 mintAmount = aumInUSD == 0 ? usdAmount : usdAmount.mul(lpSupply).div(aumInUSD);
        require(mintAmount >= _minlp, "[LpManager] min output not satisfied");
        IMintable(_lp).mint(msg.sender, mintAmount);
        IPSBT(psbt).updateAddLiqScoreForAccount(msg.sender, lpVault[_lp], usdAmount.div(VaultMSData.
USDX_DECIMALS).mul(VaultMSData.PRICE_PRECISION), 0);

        // Here still use the input parameter _token
        emit AddLiquidity(msg.sender, _token, _amount, aumInUSD, lpSupply, usdAmount, mintAmount);
        return mintAmount;
    }
```

TrungOre : To add liquidity with ETH, we can utilize the function `LpManager.addLiquidity()` by setting the

parameter `_token` as `address(0)`. Subsequently, this function will trigger an external call `IVault(lpVault[_lp]).buyUSD(_token = address(0))`. Here are the corresponding code references:

```
uint256 usdAmount = IVault(lpVault[_lp]).buyUSD(_token);
```

On the other hand, when we intend to remove ETH liquidity from the vault, we can utilize the function `LpManager.removeLiquidity()`. However, unlike `LpManager.addLiquidity()`, this function passes the address of WETH instead of `address(0)` to the external call `IVault(lpVault[_lp]).sellUSD()`. The relevant code block can be found here:

```
 address _tokenOut = _tokenOutOri==address(0) ? weth : _tokenOutOri;
        require(IVault(lpVault[_lp]).isFundingToken(_tokenOut), "[LpManager] not supported lp token");
        address _account = msg.sender;
        IERC20(_lp).safeTransferFrom(_account, address(this), _lpAmount );

        // calculate aum before sellUSD
        uint256 aumInUSD = getAumInUSD(_lp, false);
        uint256 lpSupply = IERC20(_lp).totalSupply();
        uint256 usdAmount = _lpAmount.mul(aumInUSD).div(lpSupply); //30b
        IMintable(_lp).burn(_lpAmount);
        uint256 amountOut = IVault(lpVault[_lp]).sellUSD(_tokenOut, address(this), usdAmount);
```

Impact:

- Liquidity added to the vault using ETH cannot be withdrawn.
- Users who add liquidity using ETH have the ability to remove liquidity from other users who added liquidity using WETH. This can lead to an incorrect state of the WETH market within the vault.

Recommendation

w2ning : Variable `_token` should be change to `_FundToken`

Consider below fix in the `LpManager.addLiquidity()` function

```
    function addLiquidity(address _lp, address _token, uint256 _amount, uint256 _minlp) external ove
rride payable nonReentrant returns (uint256) {
        require(isLpToken(_lp), "[LpManager] Not supported lp token" );
        require(IVault(lpVault[_lp]).isFundingToken(_token), "[LpManager] not supported lp token");
        {
            address _fundToken = _token;
            uint256 _fundAmount = _amount;
            if (_token == address(0)){
                _fundToken = weth;
                _fundAmount = msg.value;
                IWETH(weth).deposit{value: msg.value}();
            }else{
                IERC20(_fundToken).safeTransferFrom(msg.sender, address(this), _fundAmount);
            }
            require(_fundAmount > 0, "[LpManager] invalid amount");
            IERC20(_fundToken).safeTransfer(lpVault[_lp], _fundAmount);
        }

        // calculate aum before buyUSD
        uint256 aumInUSD = getAumInUSD(_lp, true);
        uint256 lpSupply = IERC20(_lp).totalSupply();

        // fix: Change _token to _fundToken
        uint256 usdAmount = IVault(lpVault[_lp]).buyUSD(_fundToken);
        uint256 mintAmount = aumInUSD == 0 ? usdAmount : usdAmount.mul(lpSupply).div(aumInUSD);
        require(mintAmount >= _minlp, "[LpManager] min output not satisfied");
        IMintable(_lp).mint(msg.sender, mintAmount);
        IPSBT(psbt).updateAddLiqScoreForAccount(msg.sender, lpVault[_lp], usdAmount.div(VaultMSData.
USDX_DECIMALS).mul(VaultMSData.PRICE_PRECISION), 0);

        // fix: Change _token to _fundToken
        emit AddLiquidity(msg.sender, _fundToken, _amount, aumInUSD, lpSupply, usdAmount, mintAmoun
t);
        return mintAmount;
    }
```

TrungOre : Use the address of weth when executing the external call `IVault(lpVault[_lp]).buyUSD(_token = weth)` when adding liquidity using eth

Client Response

Fixed, Updated in PlpManager.sol

PKO-4:Potential Liquidity Exhaustion Risk

Category	Severity	Status	Contributor
Logical	Critical	Fixed	Kong7ych3

Code Reference

- code/contracts/core/Vault.sol#L524

```
524:            _decreasePoolAmount(position.collateralToken, tokenAmount);
```

Description

Kong7ych3 : In the protocol, when the user increases the long/short position, his funds will be transferred to the corresponding vault and the poolAmount will be updated. When the user closes the position, the user's profit will be calculated according to the current price. If the user makes a profit, the vault will transfer the user's collateral and profit to the user, and the profit and collateral will be deducted from poolAmount. This means that the user's actual counterparty is the vault, not other users with opposite positions. If a user makes a huge profit, most of the funds in the vault will be paid to the user, and the vault will not distinguish whether the paid funds are long funds or short funds. This will result in the vault not having sufficient funds to transfer to other users who need to reduce their positions. This issue will cause users who withdraw their funds at a later time are more likely to suffer losses.

Recommendation

Kong7ych3 : It is recommended to separate the short and long funds into accounts, and the funds should be deducted from the pool of the opposite position when settling the user's profit. In order to avoid the principal of users with the same position being transferred out. Or establish a reserve treasury, and use the reserve funds in the treasury to pay when the funds in the vault cannot be paid.

Client Response

Fixed, Fix:

- Adjust the order of reserve amount calculation
- Compare user profit with reserved amount (which means user has maximium profit)
- Add treasury later

PKO-5:The average price must not be updated when function `Vault._decreasePosition()` is called

Category	Severity	Status	Contributor
Logical	Critical	Fixed	TrungOre

Code Reference

- code/contracts/core/Vault.sol#L266-L270

```
266:    function _decreasePosition(bytes32 key, uint256 _collateralDelta, uint256 _sizeDelta, addres
s _receiver) private returns (uint256) {
267:        // bytes32 key = vaultUtils.getPositionKey(_account, _collateralToken, _indexToken, _isL
ong, 0);
268:        VaultMSData.Position storage position = positions[key];
269:        vaultUtils.validateDecreasePosition(position,_sizeDelta, _collateralDelta);
270:        _updateGlobalSize(position.isLong, position.indexToken, _sizeDelta, position.averagePric
e, false);
```

Description

TrungOre : When users intend to decrease their position, the `Vault._decreasePosition()` function is called, which subsequently executes an internal call to `Vault._updateGlobalSize()` after validating the decrease position request.

```
/// link: code/contracts/core/Vault.sol#L266-L270
function _decreasePosition(bytes32 key, uint256 _collateralDelta, uint256 _sizeDelta, address _recei
ver) private returns (uint256) {
    VaultMSData.Position storage position = positions[key];
    vaultUtils.validateDecreasePosition(position,_sizeDelta, _collateralDelta);

    /// $audit update global size + average price here
    _updateGlobalSize(position.isLong, position.indexToken, _sizeDelta, position.averagePrice, fals
e);
```

In the private function `Vault._updateGlobalSize()`, the update for `ttREC.long(short)AveragePrice` is performed with the `_increase` parameter set to false. This implies that the next global average price will be calculated using the formula:

```
nextAveragePrice = (avgPrice * _size - _sizeDelta * nextPrice) / (_size - _sizeDelta)
```

following the implementation of the `VaultUtils.getNextAveragePrice()` function.

However, the `Vault.getNextAveragePrice()` function should only be called when merging the prices of two positions with different prices during position increases. In contrast, the purpose of `Vault.decreasePosition()` is

not to merge these positions. This inconsistency can have a significant impact on profit/loss calculations when considering the delta of all positions in the vault.

Let's consider an example where the vault has only one long position with `size = 10` and `averagePrice = 5`. At time `t` with a spot price of `p = 10`, the position's owner decides to decrease the position by `_sizeDelta = 1`. In this case, the `ttREC.longAveragePrice` will become:

```
ttREC.longAveragePrice = (10 * 5 - 1 * 10) / (10 - 1) = 40/9
```

At a spot price of `20`, the profit calculated using the updated `longAveragePrice` will be `(20 - 40/9) * 9 / (40/9) = 31.5`, which obviously makes no sense. This is because with the original position before the decrease, at a spot price of `20`, it only generates a profit of `(20 - 5) * 10 / 5 = 30`, which is less than `31.5`.

Impact: This incorrect logic can be exploited by attackers to manipulate the delta generated from all positions in the vault, thereby affecting the `Vault.getAum()` function's returned value in favor of their activities, such as minting or burning shares.

Recommendation

TrungOre : Consider not to update the `ttREC.long(short)AveragePrice` when decreasing the position.

Client Response

Fixed, Solved by using new price feed scheme. (VaultPriceFeed.sol)

PKO-6:The implementation of the `premiumFee` in the `Vault._collectMarginFees` function can potentially result in a loss of funds for users.

Category	Severity	Status	Contributor
Logical	Critical	Mitigated	TrungOre

Code Reference

- code/contracts/core/Vault.sol#L589-L596

```
589:         if (_premiumFee > 0){
590:             _validate(_position.collateral >= uint256(_premiumFee), 29);
591:             _increaseGuaranteedUsd(_position.collateralToken, uint256(_premiumFee));//increase -
> aum ↑
592:             _position.collateral = _position.collateral.sub(uint256(_premiumFee));
593:         }else if (_premiumFee <0) {
594:             _decreaseGuaranteedUsd(_position.collateralToken, uint256(-_premiumFee));//decrease
-> aum ↓
595:             _position.collateral = _position.collateral.add(uint256(-_premiumFee));
596:         }
```

Description

TrungOre : The premium fee can be positive or negative to indicate whether the position should pay a fee for the smaller position size or receive a fee from the larger position size.

An issue arises when a transaction with `premiumFee < 0` is called before a transaction with `premiumFee > 0`. When the `Vault._collectMarginFees` function is invoked with a negative `_premiumFee` obtained from the `vaultUtils.getPremiumFee` function, it temporarily decreases the value of the vault's `guaranteedUsd`

```
}else if (_premiumFee <0) {
    _decreaseGuaranteedUsd(_position.collateralToken, uint256(-_premiumFee));//decrease -> aum ↓
    _position.collateral = _position.collateral.add(uint256(-_premiumFee));
}
```

This temporary decrease in `guaranteedUsd` causes the asset under management (AUM) of the vault to temporarily increase, as `AUM = poolAmount - guaranteedUsd - delta`. This increase in AUM can benefit the liquidity provider, as they can call `LpManager.removeLiquidity()` before the transaction with `_premiumFee > 0` is

executed, enabling them to take a portion of the premium fee. However, this premium fee should only be collected by the positions with the smaller size, not the liquidity provider. Consequently, this results in a loss for the traders.

Let's consider an example with two positions in the vault:

- A long position from Alice with `size = 100` and `collateral = 10`.
- A short position from Bob with `size = 50` and `collateral = 5`.

Initially, we have `poolAmount = guaranteedUsd = 10 + 5 = 15`. Suppose there is only one liquidity provider who increases the `poolAmount` of the vault by `10` and receives `10` LP tokens. Now, we have `poolAmount = 15 + 10 = 25`.

According to the logic of the premium fee, Alice should pay a fee to Bob. Let's assume Alice needs to pay 2 USD to Bob. When the `Vault._collectMarginFees()` function is called by Bob:

- `guaranteedUsd = guaranteedUsd - premiumFee = 15 - 2 = 13`
- Bob's `position.collateral = 5 + 2 = 7`

Subsequently, the liquidity provider calls `LpManager.removeLiquidity()` to withdraw all of their liquidity (assuming no profit/loss for the traders). They will receive `aum = 25 - 13 = 12` (the liquidity provider receives more than 2 USD even when there is no profit/loss from the traders). This results in `poolAmount = 25 - 12 = 13`.

Now, if `Vault._collectMarginFees()` is called by Alice:

- `guaranteedUsd = 13 + 2 = 15`
- Alice's `position.collateral = 10 - 2 = 8`

It becomes evident that the sum of Alice's `position.collateral` and Bob's `position.collateral` is `8 + 7 = 15`, which is greater than `13 = poolAmount`. This state is incorrect.

Recommendation

TrungOre : One potential solution to address this issue is to ensure that the premium fee paid by the larger position size is equal to the premium fee claimed by the smaller position size. This can be achieved by removing the `negIndexMaxPointsPerSec` parameter. Alternatively, if the `longRate` or `shortRate` receives the value of `negIndexMaxPointsPerSec`, we can recalculate the opposite rate to maintain consistency.

Client Response

Mitigated, Adjustment of guaranteedUsd by premium fee is deleted.

PKO-7:Underflow error when calling `VaultUtils.getNextAveragePrice()` can prevent users from dercreasing the position

Category	Severity	Status	Contributor
Integer Overflow and Underflow	Critical	Fixed	TrungOre

Code Reference

- code/contracts/core/VaultUtils.sol#L385-L386

```
385:            uint256 _latestSize = _size > _sizeDelta ? _size.sub(_sizeDelta) : 0;
386:            return _latestSize > 0 ? (_averagePrice.mul(_size).sub(_sizeDelta.mul(_nextPrice))).
div(_latestSize): 0;
```

Description

TrungOre : The function `VaultUtils.getNextAveragePrice()` does not include a check if `_averagePrice * _size` is greater than `_sizeDelta * _nextPrice` before performing the subtraction when the parameter `_isIncrease` is set to `false`.

```
/// link: code/contracts/core/VaultUtils.sol#L385-L386
uint256 _latestSize = _size > _sizeDelta ? _size.sub(_sizeDelta) : 0;
return _latestSize > 0 ? (_averagePrice.mul(_size).sub(_sizeDelta.mul(_nextPrice))).div(_latestSiz
e): 0;
```

This absence of a check can lead to an underflow error during the subtraction, causing the entire function to revert. Consequently, this error affects functions such as `Vault.decreasePosition()` and `Vault.liquidatePosition()`.

Impact: Users may be unable to decrease their position when the spot price deviates significantly from the average price.

Recommendation

TrungOre : It is necessary to include a check to verify if `_averagePrice * _size` is greater than `_sizeDelta * _nextPrice` before proceeding with the subtraction operation in the code.

Client Response

Fixed, Comparision added.

PKO-8:missing _mintOut protection can lead to user lose funds.

Category	Severity	Status	Contributor
Logical	Critical	Fixed	jayphbee

Code Reference

- code/contracts/core/RouterSign.sol#L74
- code/contracts/core/RouterSign.sol#L88
- code/contracts/core/RouterSign.sol#L135
- code/contracts/core/RouterSign.sol#L149

```
74:             uint256 amountOut = _swap(_path, 0, address(this));

88:             uint256 amountOut = _swap(_path, 0, address(this));

135:             uint256 amountOut = _swap(_path, 0, address(this));

149:             uint256 amountOut = _swap(_path, 0, address(this));
```

Description

jayphbee : When user calling `increasePosition`

```
    function increasePosition(address[] memory _path, address _indexToken, uint256 _amountIn, uint25
6 _sizeDelta, bool _isLong, uint256 _price,
            uint256[] memory , uint256 ,  address, bytes memory ) external payable nonReentrant{
        require(freeTrade, "FreeTrade Not Open");
        ...
        if (_path.length > 1 && _amountIn > 0) {
            uint256 amountOut = _swap(_path, 0, address(this));
            IERC20(_path[_path.length - 1]).safeTransfer(vault, amountOut);
        }
        _increasePosition(_path[_path.length - 1], _indexToken, _sizeDelta, _isLong, _price);
    }
```

the `amountOut` is calculated as

```
uint256 amountOut = _swap(_path, 0, address(this));
```

The `_minOut` parameter for `_swap` is passed as 0. The `_swap` function calls `_vaultSwap` and the `_vaultSwap` function calls `IVault(vault).swap` and finally calls the private `_swap` function.

```
    function _swap(address _tokenIn,  address _tokenOut, address _receiver ) private returns (uint25
6) {
        ...
        uint256 _amountInUsd = tokenToUsdMin(_tokenIn, amountIn);
        uint256 _amountOut = usdToTokenMin(_tokenOut, _amountInUsd);
        ...
    }
```

The `_amountIn` of `_tokenIn` is coverted to usd by calling `tokenToUsdMin`. And then these usd is converted to `_amountOut`.

```
    function tokenToUsdMin(address _token, uint256 _tokenAmount) public view override returns (uint2
56) {
        uint256 price = getMinPrice(_token);
        return _tokenAmount.mul(price).div(10**IMintable(_token).decimals());
    }
    function usdToTokenMin(address _token, uint256 _usdAmount) public override view returns (uint25
6) {
        return _usdAmount > 0 ? usdToToken(_token, _usdAmount, getMaxPrice(_token)) : 0;
    }
    function usdToToken( address _token, uint256 _usdAmount, uint256 _price ) public view returns (u
int256) {
        // if (_usdAmount == 0)  return 0;
        uint256 decimals = IMintable(_token).decimals();
        require(decimals > 0, "invalid decimal");
        return _usdAmount.mul(10**decimals).div(_price);
    }
```

We can see that the `amountOut` is totally determined by price of `_tokenIn` and `_tokenOut`. The impact is that if the `tokenOut` price increase sharply after user submmit the `increasePosition` transaction, user will get less `amountOut` than expected.

Recommendation

jayphbee : Whenever there is a swap, use `_minOut` as parameter to protect user from sharp price fluctuation. change

```
uint256 amountOut = _swap(_path, 0, address(this));
```

to

```
uint256 amountOut = _swap(_path, _minOut, address(this));
```

Client Response

Fixed, min collateral parameter added.

PKO-9:Incorrect daily seconds used.

Category	Severity	Status	Contributor
Logical	Medium	Fixed	jayphbee

Code Reference

- code/contracts/core/Router.sol#L131
- code/contracts/core/Router.sol#L154
- code/contracts/core/Router.sol#L165
- code/contracts/core/Router.sol#L178

```
131:                if (maxSwapAmountPerDay > 0 && swapDailyRecord[block.timestamp/3600].add(addAmou
nt) > maxSwapAmountPerDay){

154:        swapDailyRecord[block.timestamp/3600] = swapDailyRecord[block.timestamp/3600].add(_amoun
tIn.mul(price).div(10 ** decimals));

165:        swapDailyRecord[block.timestamp/3600] = swapDailyRecord[block.timestamp/3600].add(msg.va
lue.mul(price).div(10 ** decimals));

178:        swapDailyRecord[block.timestamp/3600] = swapDailyRecord[block.timestamp/3600].add(_amoun
tIn.mul(price).div(10 ** decimals));
```

Description

jayphbee : 1 day has 24 * 60 * 60 = 86400 seconds. There are multiple places where use 3600 as daily seconds to accumulate the daily swap record. The impact is that daily swap amount can be 24 times bigger than `maxSwapAmountPerDay`.

```
                if (maxSwapAmountPerDay > 0 && swapDailyRecord[block.timestamp/3600].add(addAmount)
> maxSwapAmountPerDay){
                    return false;
                }
```

Recommendation

jayphbee : correct daily seconds

Client Response

Fixed, 3600->86400 changed

PKO-10:Open a position without updating the price to increase the winning rate

Category	Severity	Status	Contributor
Race condition	Medium	Fixed	Kong7ych3

Code Reference

- code/contracts/core/Vault.sol#L189
- code/contracts/core/Vault.sol#L259

```
189:    function increasePosition(address _account, address _collateralToken, address _indexToken, u
int256 _sizeDelta, bool _isLong) external override nonReentrant {

259:    function decreasePosition(address _account, address _collateralToken, address _indexToken, u
int256 _collateralDelta, uint256 _sizeDelta, bool _isLong, address _receiver
```

Description

Kong7ych3 : In the agreement, when the user increases/decreases the position, the price will be obtained through getMinPrice/getMaxPrice. Prices are sourced from Chainlink, Pyth, and self-feeding price services. Theoretically these price feeds cannot be manipulated, but unfortunately when service providers submit new prices, all users can always find these transactions in the mempool. Therefore, whenever the price is updated, malicious users can use MEV to open a position before the price update, and then close the position after the price update. All operations are completed in the same block. As long as the profit from price fluctuations exceeds its cost, malicious users can continue to make profits in this way.

It should be noted that Pyth is used in the oracle, and there is an updatePriceFeeds interface in Pyth, which allows users to submit legitimate price updates themselves. Therefore, malicious users can more easily profit by using this method.

Recommendation

Kong7ych3 : It is recommended to prohibit users from closing positions immediately after opening positions in the same block, and reduce the priceSafetyGap in the oracle.

Client Response

Fixed,

- Method1: only our robot to do trade with PositionRouter
- Method2: We have modified a short-term tax. User cannot take out any profit within the tax-time.

PKO-11:Some tokens will be lost when function `Router.directPoolDeposit()` is called

Category	Severity	Status	Contributor
Logical	Medium	Fixed	TrungOre

Code Reference

- code/contracts/core/Vault.sol#L385-L391

```
385:    function directPoolDeposit(address _token) external override nonReentrant {
386:        _validate(fundingTokens.contains(_token), 14);
387:        uint256 tokenAmount = _transferIn(_token);
388:        _validate(tokenAmount > 0, 15);
389:        // _increasePoolAmount(_token, tokenAmount);
390:        emit DirectPoolDeposit(_token, tokenAmount);
391:    }
```

Description

TrungOre : The function `Router.directPoolDeposit()` mandates that the sender transfers `_amount` tokens from their wallet to the vault and subsequently executes the external call to `Vault.directPoolDeposit()`. However, the `Vault.directPoolDeposit()` function does not affect the state of the vault, such as `poolAmount` or `guaranteedUsd`. It solely updates the `reservedAmount` of the token, which does not yield any profits for traders or liquidity providers. Consequently, this amount will be wasted and lost within the contracts.

Recommendation

TrungOre : Consider to increase the `poolAmount` of the vault in the function `Vault.directPoolDeposit()`

Client Response

Fixed, Funding-token check added.

PKO-12:User can' t be guaranteed to decrease his position.

Category	Severity	Status	Contributor
Privilege Related	Medium	Fixed	jayphbee

Code Reference

- code/contracts/core/RouterSign.sol#L96
- code/contracts/core/RouterSign.sol#L103
- code/contracts/core/RouterSign.sol#L109
- code/contracts/core/RouterSign.sol#L118

```
96:         require(freeTrade, "FreeTrade Not Open");

103:         require(freeTrade, "FreeTrade Not Open");

109:         require(freeTrade, "FreeTrade Not Open");

118:         require(freeTrade, "FreeTrade Not Open");
```

Description

jayphbee : The `freeTrade` flag is used to control when the free trade is open and the owner of `RouterSign` contract can set it. If the `freeTrade` is `false`, user can't increase or decrease his position.

There is a scenery that the owner set the `freeTrade` flag to be `true` and a user calls `increasePosition` to open a position, but for some reason the owner set the `freeTrade` to `false`. At this time the market suffers huge price flucuation and the use wants to descreae his position to lower the liquidation risk but `freeTrade` flag is false user can't decrease his position in time.

The impact is that user can lose money due to he is unable to decrease his position in time.

Recommendation

jayphbee : User should have the abliity to decrease his position to lower the liquidation risk at any time. I would suggest remove the

```
require(freeTrade, "FreeTrade Not Open");
```

in `decreasePosition, decreasePositionETH, decreasePositionAndSwap, decreasePositionAndSwapETH` functions.

Client Response

Fixed, Free-trade flag is removed.

PKO-13:User's position can become liquidatable because system parameter change.

Category	Severity	Status	Contributor
Privilege Related	Medium	Acknowledged	jayphbee

Code Reference

- code/contracts/core/VaultUtils.sol#L99
- code/contracts/core/VaultUtils.sol#L116
- code/contracts/core/VaultUtils.sol#L130
- code/contracts/core/VaultUtils.sol#L141

```
99:    function setPremiumRate(uint256 _premiumBasisPoints, int256 _posIndexMaxPoints, int256 _negIn
dexMaxPoints, uint256 _maxPremiumBasisErrorUSD) external onlyOwner{

116:    function setFundingRate(uint256 _fundingRateFactor, uint256 _stableFundingRateFactor) extern
al onlyOwner{

130:    function setTaxRate(uint256 _taxMax, uint256 _taxTime) external onlyOwner{

141:    function setFees(
```

Description

jayphbee : In the `VaultUtils` contract, there are functions(`setPremiumRate`, `setFundingRate`, `setTaxRate` and `setFee`) that can be used to change the system parameters, some of which can effect user's position liquidation price , like the `premiumBasisPointsPerSec`, `taxMax` and `fundingRateFactor`.

```
    function getLiqPrice(bytes32 _key) public view override returns (int256){
        VaultMSData.Position memory position = vault.getPositionStructByKey(_key);
        if (position.size < 1) return 0;

        VaultMSData.TradingFee memory colTF = vault.getTradingFee(position.collateralToken);
        VaultMSData.TradingFee memory idxTF = vault.getTradingFee(position.indexToken);

        uint256 marginFees = getFundingFee(position, colTF).add(getPositionFee(position, 0, idxTF));
        int256 _premiumFee = getPremiumFee(position, idxTF);

        uint256 colRemain = position.collateral.sub(marginFees);
        colRemain = _premiumFee >= 0 ?position.collateral.sub(uint256(_premiumFee)) : position.colla
teral.add(uint256(-_premiumFee)) ;
        // (bool hasProfit, uint256 delta) = getDelta(position.indexToken, position.size, position.a
veragePrice, position.isLong, position.lastUpdateTime, position.collateral);
        // colRemain = hasProfit ? position.collateral.sub(delta) : position.collateral.add(delta);

        uint256 acceptPriceGap = colRemain.mul(position.averagePrice).div(position.size);
        return position.isLong ? int256(position.averagePrice) - int256(acceptPriceGap) : int256(pos
ition.averagePrice.add(acceptPriceGap));
    }
```

The impact is that if there's no time delay for these parameter changes take effective, some of use's position can become liquidatable immediately, thus user can lose money.

Recommendation

jayphbee : Introduce time delay for these system wide parameter changes, so that user can have time to react to this changes.

Client Response

Acknowledged, We will announce the change on the website first.

PKO-14:Weak Sources of Randomness in `randomSource::_seed`

Category	Severity	Status	Contributor
Weak Sources of Randomness	**Medium**	**Acknowledged**	**w2ning**

Code Reference

- code/contracts/utils/randomSource.sol#L17

```
17:         return  _seed() % _modulus;
```

Description

w2ning : Weak sources of randomness due to a modulo on block.timestamp, now or blockhash. These can be influenced by miners to some extent so they should be avoided.

```
    function _seed() internal returns (uint256) {
        uint256 res = uint256(keccak256(abi.encodePacked(block.timestamp,
                                                         randNonce.mul(13),
                                                         msg.sender,
                                                         randNonce)));

        randNonce = randNonce.add(res.div(75017)).mod(MAX_NONCE);
        return res;
    }
```

Recommendation

w2ning : Do not use block.timestamp, now or blockhash as a source of randomness. Using random numbers provided by chainlink

Client Response

Acknowledged, The randomness is not used in this project. We will use random service provided chainlink in the feature if needed.

PKO-15:updateRate not performed after buyUSD operation

Category	Severity	Status	Contributor
Logical	Medium	Fixed	Kong7ych3, TrungOre

Code Reference

- code/contracts/core/Vault.sol#L153
- code/contracts/core/Vault.sol#L153-L163

```
153:    function buyUSD(address _token) external override nonReentrant onlyManager returns (uint256)
{

153:    function buyUSD(address _token) external override nonReentrant onlyManager returns (uint256)
{
154:        _validate(fundingTokens.contains(_token), 16);
155:        uint256 tokenAmount = _transferIn(_token);
156:        _validate(tokenAmount > 0, 17);
157:        updateRate(_token);
158:        uint256 feeBasisPoints = vaultUtils.getBuyLpFeeBasisPoints(_token, tokenToUsdMin(_token,
tokenAmount));
159:        uint256 amountAfterFees = _collectSwapFees(_token, tokenAmount, feeBasisPoints);
160:        //- fee transfered out inside _collectSwapFees
161:        _increasePoolAmount(_token, amountAfterFees);
162:        return tokenToUsdMin(_token, amountAfterFees);
163:    }
```

Description

Kong7ych3 : In the vault contract, the updateRate will be updated before the buyUSD operation, but the updateRate will not be updated again after the buyUSD operation is completed. This will cause tradingFee cannot be updated normally.
TrungOre : The function `updateRate()` serves the purpose of accumulating the funding fee and premium fee, and determining the latest funding rate and long/short rate. The funding rate is calculated based on the ratio of `reservedAmount` to `poolAmount`.

```
    function getLatestFundingRatePerSec(address _token) public view override returns (uint256){
        VaultMSData.TokenBase memory tB = vault.getTokenBase(_token);
        if (tB.poolAmount == 0) return 0;
        // tradingFee.fundingRatePerHour
        uint256 _fundingUtil = tB.reservedAmount.mul(VaultMSData.PRC_RATE_PRECISION).div(tB.poolAmou
nt);
        return hRateToSecRate(fundingRateFactor.mul(_fundingUtil)).div(VaultMSData.PRC_RATE_PRECISIO
N);
    }
```

It is important to note that any modifications involving `reservedAmount` or `poolAmount` necessitate the recalculation of the funding rate. However, in the function `Vault.buyUSD()`, towards the end of the function, the poolAmount of the `_token` is increased by `amountAfterFees`, but no `updateRate(_token)` is called afterwards.

```
        _increasePoolAmount(_token, amountAfterFees);
```

As a result, this omission causes the funding rate to become outdated, leading to incorrect fee calculations.

Recommendation

Kong7ych3 : It is recommended to updateRate after the buyUSD operation is completed.
TrungOre : Call `updateRate()` at the end of the function `Vault.buyUSD()`

Client Response

Fixed, updateRate() added.

PKO-16:Any `msg.value` exceeding `_executionFee` should be return

Category	Severity	Status	Contributor
Logical	Low	Fixed	Kong7ych3

Code Reference

- code/contracts/core/PositionRouter.sol#L361

```
361:         require(msg.value >= _executionFee, "PositionRouter: invalid msg.value");
```

Description

Kong7ych3 : In the PositionRouter contract, the user can request to increase the position through the createIncreasePosition function, which will check that the user's msg.value must be greater than or equal to _executionFee. But the refund logic is not implemented. If the user's msg.value is greater than _executionFee, the extra native tokens will not be withdrawn.

Recommendation

Kong7ych3 : It is recommended to check that msg.value must be equal to _executionFee. Or add a refund function, when the user's msg.value is greater than _executionFee, the excess funds should be refunded to the user.

Client Response

Fixed, change: [fee >= requirement] into[fee == requirement]

PKO-17:Global variable `psbtLogic` can never be changed in `PSBT` contract

Category	Severity	Status	Contributor
Logical	Low	Fixed	w2ning

Code Reference

- code/contracts/DID/PSBT.sol#L39

```
39:    address public psbtLogic;
```

Description

w2ning : Global variable `psbtLogic` can never be changed in `PSBT` contract

```
    address public psbtLogic;

    constructor( address _NFTUtils) {
        require(_NFTUtils != address(0), "empty NFTUtils address");
        contMap["nftutil"] = _NFTUtils;
        uint256 cur_time = block.timestamp;
        string memory defRC =  INFTUtils(contMap["nftutil"]).genReferralCode(0);
        if (refCodeOwner[defRC]!= address(0))
            defRC = string(abi.encodePacked(defRC, cur_time));
        PSBTData.PSBTStr memory _PSBTStr = PSBTData.PSBTStr(address(this), "PSBT OFFICIAL", defRC, c
ur_time, 1);
        _tokens.push(_PSBTStr);
        addressToTokenID[address(this)] = 0;
        refCodeOwner[defRC] = address(this);
        _balances[address(this)] = 1;
        //set default:
        scorePara[1] = 20;//score_tradeOwn
        scorePara[2] = 4;//score_tradeOwn
        scorePara[3] = 15;//score_swapOwn
        scorePara[4] = 3;//score_swapChd
        scorePara[5] = 10;//score_addLiqOwn
        scorePara[6] = 2;//score_addLiqChd
        scorePara[8] = 2;//invite create Account
        scorePara[101] = 20;//score_tradeOwn
        scorePara[102] = 4;//score_tradeOwn
        scorePara[103] = 15;//score_swapOwn
        scorePara[104] = 3;//score_swapChd
        scorePara[105] = 10;//score_addLiqOwn
        scorePara[106] = 2;//score_addLiqChd
        scorePara[108] = 10;//invite create Account
    }
```

Recommendation

w2ning : Assign a value to the variable in the constructor And add a function to modify the variable

Consider below fix in the `PSBT.constructor()` function

```
    constructor( address _NFTUtils, address _psbtLogic) {
        psbtLogic = _psbtLogic;
    }
    function setpsbtLogic(address _psbtLogic) onlyOwner{
        psbtLogic = _psbtLogic;
}
```

Client Response

Fixed, Changed PSBT -> PID.sol

PKO-18:Logic error in `Array` contract `get` function

Category	Severity	Status	Contributor
Logical	Low	Fixed	w2ning

Code Reference

- code/contracts/utils/Array.sol#L11

```
11:         return bytes32(0);
```

Description

w2ning : When the index exceeds the array length, it should be directly revert.

```
function get(bytes32[] memory arr, uint256 index) internal pure returns (bytes32) {
    if (index < arr.length) {
        return arr[index];
    }

    return bytes32(0);
}
```

Recommendation

w2ning : Revert when the index exceeds the array length

Consider below fix in the `Array.get()` function

```
function get(bytes32[] memory arr, uint256 index) internal pure returns (bytes32) {

    require(index < arr.length, "Exceed the range of arry")

    return arr[index];

}
```

Client Response

Fixed, Add getSafe( ) function along side get( ). (this function is not used in Pinnako so far)

PKO-19:Logical error: `tradingTax` miss a set function

Category	Severity	Status	Contributor
Logical	Low	Fixed	w2ning

Code Reference

- code/contracts/core/VaultUtils.sol#L46

```
46:    mapping(address => VaultMSData.TradingTax) tradingTax;
```

Description

w2ning : The global variable `tradingTax` of contract cannot be modified.

```
    mapping(address => VaultMSData.TradingTax) tradingTax;

    function getTradingTax(address _token) public override view returns (VaultMSData.TradingTax memo
ry){
        return tradingTax[_token];
    }
```

Recommendation

w2ning : Add a function to change the global variable

Consider below fix in the `VaultUtils` contract

```
    function setTradingTax(address _token, VaultMSData.TradingTax memory _tradingTax) external onlyO
wner {
        tradingTax[_token] = _tradingTax;
    }
```

Client Response

Fixed, tradingTax Struct deleted. We use global tax instead.

PKO-20:Missing limit in `RouterSign::initialization`

Category	Severity	Status	Contributor
Code Style	Low	Fixed	Kong7ych3

Code Reference

- code/contracts/core/RouterSign.sol#L36-L41

```
36:     function initialize(address _vault, address _weth, address _priceFeed, address _psbt) externa
l onlyOwner {
37:         vault = _vault;
38:         weth = _weth;
39:         priceFeed = _priceFeed;
40:         PSBT = _psbt;
41:     }
```

Description

Kong7ych3 : In the RouterSign contract, the owner can initialize key parameters such as vault, priceFeed, and PSBT in the contract through the initialize function, but there is no restriction on calling the initialize function repeatedly, and the contract already has setPSBT, setPriceFeed, and setVault functions to set these parameters. So this will cause double initialization issue.

Recommendation

Kong7ych3 : It is recommended to prohibit repeated initialize operations.

Consider below fix in the `RouterSign.initialize()` function

```
require(!isInitialized, "RouterSign: already initialized");
isInitialized = true;
```

Client Response

Fixed, We have deleted redundant set address functions and only keep initailize function.

PKO-21:Precision issue:Multiplication before division

Category	Severity	Status	Contributor
Logical	Low	Fixed	w2ning, jayphbee

Code Reference

- code/contracts/core/LpManager.sol#L92
- code/contracts/core/LpManager.sol#L121
- code/contracts/DID/PSBT.sol#L265
- code/contracts/DID/PSBT.sol#L267
- code/contracts/DID/PSBT.sol#L269

```
92:         IPSBT(psbt).updateAddLiqScoreForAccount(msg.sender, lpVault[_lp], usdAmount.div(VaultMSDa
ta.USDX_DECIMALS).mul(VaultMSData.PRICE_PRECISION), 0);

121:         IPSBT(psbt).updateAddLiqScoreForAccount(_account, lpVault[_lp], usdAmount.div(VaultMSDat
a.USDX_DECIMALS).mul(VaultMSData.PRICE_PRECISION), 100);

265:     function updateScoreForAccount(address _account, address /*_vault*/, uint256 _amount, uint25
6 _reasonCode) external onlyScoreUpdater override {

267:         updateScore(_account, _account, _amount.div(1000).mul(scorePara[_reasonCode]).div(PSBTDa
ta.USD_TO_SCORE_PRECISION),_reasonCode);

269:             updateScore(_par[0], _account, _amount.div(1000).mul(scorePara[1000 + _reasonCode]).
div(PSBTData.USD_TO_SCORE_PRECISION), 1000 + _reasonCode);
```

Description

w2ning : Performing division before multiplication can lead to precision loss.

```
    function updateAddLiqScoreForAccount(address _account, address /*_vault*/, uint256 _amount, uint
256 _refCode) external onlyScoreUpdater override {
        (address[] memory _par,  ) = getReferralForAccount(_account);
        incrementAddUint(_account, PSBTData.ACCUM_ADDLIQUIDITY, _amount);

        // Performing division before multiplication can lead to precision loss.
        updateScore(_account,  _account, _amount.div(1000).mul(scorePara[5 + _refCode]).div(PSBTDat
a.USD_TO_SCORE_PRECISION), 3 + _refCode);
        if (_par.length == 1)
            updateScore(_par[0], _account, _amount.div(1000).mul(scorePara[6 + _refCode]).div(PSBTDa
ta.USD_TO_SCORE_PRECISION), 13 + _refCode);
    }
```

jayphbee : User's liquidity score is updated by calling `IPSBT(psbt).updateAddLiqScoreForAccount`. The score is calculated as

```
usdAmount.div(VaultMSData.USDX_DECIMALS).mul(VaultMSData.PRICE_PRECISION)
```

The impact is that user could get less score than expected because precision lose by div then mul.

Recommendation

w2ning : Performing multiplication before division can sometimes avoid loss of precision.
jayphbee : mul before div.

```
IPSBT(psbt).updateAddLiqScoreForAccount(msg.sender, lpVault[_lp], usdAmount.mul(VaultMSData.PRICE_PR
ECISION).div(VaultMSData.USDX_DECIMALS), 0);
```

Client Response

Fixed, Move div(1000) after mul.

PKO-22:Risk of owner excessive privilege

Category	Severity	Status	Contributor
Privilege Related	Low	Fixed	Kong7ych3

Code Reference

- code/contracts/core/RouterSign.sol#L43-L53
- code/contracts/core/Vault.sol#L144

```
43:     function setPSBT(address _psbt) external onlyOwner {
44:         PSBT = _psbt;
45:     }
46:
47:     function setPriceFeed(address _priceFeed) external onlyOwner {
48:         priceFeed = _priceFeed;
49:     }
50:
51:     function setVault(address _vault) external onlyOwner {
52:         vault = _vault;
53:     }

144:     function upgradeVault(address _newVault, address _token, uint256 _amount) external onlyOwner
{
```

Description

Kong7ych3 : In the protocol, the owner role can modify sensitive parameters. For example, the owner role can modify the priceFeed and vault parameters in the RouterSign contract through the setPriceFeed and setVault functions. These parameters are related to the security of user funds. And in the Vault contract, the owner role can transfer the funds in the vault to the specified address through the upgradeVault function. These will lead to the risk of owner excessive privilege.

Recommendation

Kong7ych3 : In the short term, in order to ensure the stable operation of the protocol in the early stage, owner ownership can be transferred to the timelock contract. The project team can use multisig contracts to control the timelock contract to avoid single-point risks, and the authority to suspend the protocol can be handed over to the EOA control of the project team , in case of emergency. In the long run, when the protocol is running stably, it is more appropriate to hand over its ownership to community governance. It can increase users' trust in the protocol and avoid the risk of excessive privilege.

Client Response

Fixed, Yes, we have timelock and multi-sig wallet

PKO-23:addLiquidity should restrict msg.value == 0 when _token is not native.

Category	Severity	Status	Contributor
Logical	Low	Fixed	jayphbee

Code Reference

- code/contracts/core/LpManager.sol#L78-L80

```
78:             }else{
79:                 IERC20(_fundToken).safeTransferFrom(msg.sender, address(this), _fundAmount);
80:             }
```

Description

jayphbee : User can add liquidity with native and non-native token by calling `addLiquidity` function. When `_token` is not native, there should be a check to prevent user from unexpectedly sneding ether.

Recommendation

jayphbee : Add msg.value == 0 check.

```
            if (_token == address(0)){
                _fundToken = weth;
                _fundAmount = msg.value;
                IWETH(weth).deposit{value: msg.value}();
            }else{
                require(msg.value == 0, "Invalid ether send");
                IERC20(_fundToken).safeTransferFrom(msg.sender, address(this), _fundAmount);
            }
```

Client Response

Fixed, functions set as non-payable

PKO-24:createIncreasePosition and createDecreasePosition should have strict msg.value check.

Category	Severity	Status	Contributor
Logical	Low	Fixed	jayphbee

Code Reference

- code/contracts/core/PositionRouter.sol#L361
- code/contracts/core/PositionRouter.sol#L432

```
361:        require(msg.value >= _executionFee, "PositionRouter: invalid msg.value");

432:        require(msg.value >= _executionFee, "PositionRouter: invalid msg.value");
```

Description

jayphbee : `createIncreasePosition` and `createDecreasePosition` use `_executionFee` to cover the execution fee. User should not overpay this fee. But they can unexpectedly pay more than expected because

```
require(msg.value >= _executionFee, "PositionRouter: invalid msg.value");
```

Recommendation

jayphbee : restrict `msg.value == _executionFee`

Client Response

Fixed, change: [fee >= requirement] into [fee == requirement]

PKO-25:setMaxGlobalSizes function lack array length equality check for its parameters.

Category	Severity	Status	Contributor
Logical	Low	Fixed	jayphbee

Code Reference

- code/contracts/core/BasePositionManager.sol#L83-L95

```
83:     function setMaxGlobalSizes(
84:         address[] memory _tokens,
85:         uint256[] memory _longSizes,
86:         uint256[] memory _shortSizes
87:     ) external onlyOwner {
88:         for (uint256 i = 0; i < _tokens.length; i++) {
89:             address token = _tokens[i];
90:             maxGlobalLongSizes[token] = _longSizes[i];
91:             maxGlobalShortSizes[token] = _shortSizes[i];
92:         }
93: 
94:         emit SetMaxGlobalSizes(_tokens, _longSizes, _shortSizes);
95:     }
```

Description

jayphbee : The `setMaxGlobalSizes` function have no array length check for its parameters.

```
function setMaxGlobalSizes(
        address[] memory _tokens,
        uint256[] memory _longSizes,
        uint256[] memory _shortSizes
    ) external onlyOwner { // @audit-issue [L-01] length equality check
        for (uint256 i = 0; i < _tokens.length; i++) {
            address token = _tokens[i];
            maxGlobalLongSizes[token] = _longSizes[i];
            maxGlobalShortSizes[token] = _shortSizes[i];
        }

        emit SetMaxGlobalSizes(_tokens, _longSizes, _shortSizes);
    }
```

Recommendation

jayphbee : Add length equality check.

```
    function setMaxGlobalSizes(
        address[] memory _tokens,
        uint256[] memory _longSizes,
        uint256[] memory _shortSizes
    ) external onlyOwner { // @audit-issue [L-01] length equality check
        require(_tokens.length == _longSizes.length && _tokens.length == _shortSizes.length, "Invali
d array length");
        ...
    }
```

Client Response

Fixed, Function deleted. Trading Size only kept in vaultUtils.sol

PKO-26:BasePositionManager - no params value check in `setMaxGlobalSizes` function

Category	Severity	Status	Contributor
Code style	Informational	Fixed	zeroxvee

Code Reference

- code/contracts/core/BasePositionManager.sol#L83

```
83:    function setMaxGlobalSizes(
```

Description

zeroxvee : Function, `setMaxGlobalSizesIt` would be safer to add a condition that checks if the lengths of `_tokens`, `_longSizes`, and `_shortSizes` are equal, in order to avoid array out of bound errors.

Recommendation

zeroxvee : Consider adding this line of code

```
{require(_tokens.length == _longSizes.length, 'BasePositionManager: array length mismatch');
```

Client Response

Fixed, Function deleted. Trading Size only kept in vaultUtils

PKO-27:Consider creating structs for return types on `OrderBook.getIncreaseOrder`, `OrderBook.getDecreaseOrder`, and `OrderBook.getSwapOrder`

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	0xgm

Code Reference

- code/contracts/core/OrderBook.sol#L267-L290
- code/contracts/core/OrderBook.sol#L554-L608

```
267:     function getSwapOrder(address _account, uint256 _orderIndex) override public view returns (
268:         address path0,
269:         address path1,
270:         address path2,
271:         uint256 amountIn,
272:         uint256 minOut,
273:         uint256 triggerRatio,
274:         bool triggerAboveThreshold,
275:         bool shouldUnwrap,
276:         uint256 executionFee
277:     ) {
278:         SwapOrder memory order = swapOrders[_account][_orderIndex];
279:         return (
280:             order.path.length > 0 ? order.path[0] : address(0),
281:             order.path.length > 1 ? order.path[1] : address(0),
282:             order.path.length > 2 ? order.path[2] : address(0),
283:             order.amountIn,
284:             order.minOut,
285:             order.triggerRatio,
286:             order.triggerAboveThreshold,
287:             order.shouldUnwrap,
288:             order.executionFee
289:         );
290:     }

554:     function createIncreaseOrder(
555:         address[] memory _path,
556:         uint256 _amountIn,
557:         address _indexToken,
558:         uint256 _minOut,
559:         uint256 _sizeDelta,
560:         address _collateralToken,
561:         bool _isLong,
562:         uint256 _triggerPrice,
563:         bool _triggerAboveThreshold,
564:         uint256 _executionFee,
565:         bool _shouldWrap
566:     ) external payable nonReentrant {
567:
568:         // always need this call because of mandatory executionFee user has to transfer in ETH
```

```
569:         _transferInETH();
570:         require(_path.length == 1 || _path.length == 2, "invalid path");
571:         require(_executionFee >= minExecutionFee, "OrderBook: insufficient execution fee");
572:         if (_shouldWrap) {
573:             require(_path[0] == weth, "OrderBook: only weth could be wrapped");
574:             require(msg.value == _executionFee.add(_amountIn), "OrderBook: incorrect value trans
ferred");
575:         } else {
576:             require(msg.value == _executionFee, "OrderBook: incorrect execution fee transferre
d");
577:             IRouter(router).pluginTransfer(_path[0], msg.sender, address(this), _amountIn);
578:         }
579:
580:         address _purchaseToken = _path[_path.length - 1];
581:         uint256 _purchaseTokenAmount;
582:         if (_path.length > 1) {
583:             require(_path[0] != _purchaseToken, "OrderBook: invalid _path");
584:             IERC20(_path[0]).safeTransfer(vault, _amountIn);
585:             _purchaseTokenAmount = _swap(_path, _minOut, address(this));
586:         } else {
587:             _purchaseTokenAmount = _amountIn;
588:         }
589:
590:         {
591:             uint256 _purchaseTokenAmountUsd = IVault(vault).tokenToUsdMin(_purchaseToken, _purch
aseTokenAmount);
592:             require(_purchaseTokenAmountUsd >= minPurchaseTokenAmountUsd, "OrderBook: insufficie
nt collateral");
593:         }
594:
595:         _createIncreaseOrder(
596:             msg.sender,
597:             _purchaseToken,
598:             _purchaseTokenAmount,
599:             _collateralToken,
600:             _indexToken,
601:             _sizeDelta,
602:             _isLong,
603:             _triggerPrice,
604:             _triggerAboveThreshold,
605:             _executionFee
```

```
606:         );
607:
608:     }
```

Description

0xgm : Improve code readability by converting return types for the following three functions that get orders of three types (increase, decrease, swap) from the OrderBook.

```
contract OrderBook {
function getIncreaseOrder(
        address _account,
        uint256 _orderIndex
    )
        public
        view
        override
        returns (
            address purchaseToken,
            uint256 purchaseTokenAmount,
            address collateralToken,
            address indexToken,
            uint256 sizeDelta,
            bool isLong,
            uint256 triggerPrice,
            bool triggerAboveThreshold,
            uint256 executionFee
        )
    {
        IncreaseOrder memory order = increaseOrders[_account][_orderIndex];
        return (
            order.purchaseToken,
            order.purchaseTokenAmount,
            order.collateralToken,
            order.indexToken,
            order.sizeDelta,
            order.isLong,
            order.triggerPrice,
            order.triggerAboveThreshold,
            order.executionFee
        );
    }}

// ... repeated for getDecreaseOrder() and getSwapOrder(), excluded for brevity
```

Recommendation

0xgm : Consider updating the return type into using the `Struct memory` and return the `order` which will match the type in the `getIncreaseOrder()`, `getDecreaseOrder()`, and `getSwapOrder()` functions to improve readability:

```
function getIncreaseOrder(
        address _account,
        uint256 _orderIndex
    ) public view override returns (IncreaseOrder memory)
    {
        IncreaseOrder memory order = increaseOrders[_account][_orderIndex];
        return order;
    }

    function getDecreaseOrder(
        address _account,
        uint256 _orderIndex
    ) public view override returns (DecreaseOrder memory)
    {
        DecreaseOrder memory order = decreaseOrders[_account][_orderIndex];
        return order;
    }

function getSwapOrder(
        address _account,
        uint256 _orderIndex
    ) public view override returns (SwapOrder memory)
    {
        SwapOrder memory order = swapOrders[_account][_orderIndex];
        return order;
    }
}
```

Client Response

Fixed, Changed to struct.

PKO-28:Gas Optimization: Unused function

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	jayphbee

Code Reference

- code/contracts/core/Router.sol#L299-L303

```
299:    function isContract(address addr) private view returns (bool) {
300:        uint size;
301:        assembly { size := extcodesize(addr) }
302:        return size > 0;
303:    }
```

Description

jayphbee : `isContract` is unused in the Router contract.

Recommendation

jayphbee : Remove the unused `isContract` function.

Client Response

Fixed, Function deleted.

PKO-29:Gas Optimization:SafeMath is unnecessary after solidity 0.8.0

Category	Severity	Status	Contributor
Gas Optimization	Informational	Acknowledged	jayphbee

Code Reference

- code/contracts/core/PositionManager.sol#L12
- code/contracts/core/PositionRouter.sol#L13
- code/contracts/core/OrderBook.sol#L17
- code/contracts/core/RouterSign.sol#L18
- code/contracts/core/Router.sol#L18
- code/contracts/core/Vault.sol#L18
- code/contracts/core/VaultStorage.sol#L18
- code/contracts/core/VaultUtils.sol#L18
- code/contracts/core/BasePositionManager.sol#L20
- code/contracts/core/LpManager.sol#L22

```
12:    using SafeMath for uint256; //BLKMDF

13:    using SafeMath for uint256;

17:    using SafeMath for uint256;

18:    using SafeMath for uint256;

18:    using SafeMath for uint256;

18:import "./interfaces/IFeeRouter.sol";

18:    using EnumerableValues for EnumerableSet.Bytes32Set;

18:    using SafeMath for uint256;

20:    using SafeMath for uint256;

22:    using SafeMath for uint256;
```

Description

jayphbee : After solidity 0.8.0 SafeMath libaray is unnessary.

Recommendation

jayphbee : Remove SafeMath usage in the codebase.

Client Response

Acknowledged, Code kept to avoid some compile-error.

PKO-30:LpManager - redundant checks in `setLP()` and `delLP()` functions

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	zeroxvee

Code Reference

- code/contracts/core/LpManager.sol#L49
- code/contracts/core/LpManager.sol#L56

```
49:    function setLP(address _lptoken, address _vault) external onlyOwner {

56:    function delLP(address _lptoken) external onlyOwner {
```

Description

zeroxvee : In setLP() and delLP(), you are checking if lpTokens.contains(_lptoken) and then adding or removing the _lptoken. OpenZeppelin's EnumerableSet library automatically checks if the element is in the set when adding or removing, so you can remove those checks for gas savings.

Recommendation

zeroxvee : consider removing the redundant checks

```
function setLP(address _lptoken, address _vault) external onlyOwner {lpVault[_lptoken] = _vault;}

function delLP(address _lptoken) external onlyOwner {  lpVault[_lptoken] = address(0);}
```

Client Response

Fixed, Modified.

PKO-31:Missing event record

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	Kong7ych3

Code Reference

- code/contracts/core/RouterSign.sol#L43-L53
- code/contracts/core/Router.sol#L48-L73
- code/contracts/core/LpManager.sol#L49
- code/contracts/core/LpManager.sol#L56
- code/contracts/core/LpManager.sol#L64
- code/contracts/core/VaultUtils.sol#L77-L167
- code/contracts/core/Vault.sol#L96-L142

```
43:     function setPSBT(address _psbt) external onlyOwner {
44:         PSBT = _psbt;
45:     }
46:
47:     function setPriceFeed(address _priceFeed) external onlyOwner {
48:         priceFeed = _priceFeed;
49:     }
50:
51:     function setVault(address _vault) external onlyOwner {
52:         vault = _vault;
53:     }
```

```
48:     function setPSBT(address _psbt) external onlyOwner {
49:         PSBT = _psbt;
50:     }
51:
52:     function setIsSwapOpenForPublic(bool status) external onlyOwner{
53:         isSwapOpenForPublic = status;
54:     }
55:
56:     function setMaxSwapRatio(address _token, uint256 _ratio) external onlyOwner{
57:         swapMaxRatio[_token] = _ratio;
58:     }
59:     function setMaxSwapAmountPerDay(uint256 _amount) external onlyOwner{
60:         maxSwapAmountPerDay = _amount;
61:     }
62:
63:     function setValidateContract(bool _valid) external onlyOwner {
64:         validateContract = _valid;
65:     }
66:
67:     function addPlugin(address _plugin) external override onlyOwner {
68:         plugins[_plugin] = true;
69:     }
70:
71:     function removePlugin(address _plugin) external onlyOwner {
72:         plugins[_plugin] = false;
73:     }
```

```
49:     function setLP(address _lptoken, address _vault) external onlyOwner {
```

```
56:     function delLP(address _lptoken) external onlyOwner {
```

```
64:     function setPSBT(address _psbt) external onlyOwner {

77:     function setMaxProfitRatio(uint256 _setRatio) external onlyOwner{
78:         require(_setRatio > VaultMSData.COM_RATE_PRECISION, "ratio small");
79:         maxProfitRatio = _setRatio;
80:     }
81:
82:     function setSpreadBasis(address _token, uint256 _spreadBasis, uint256 _maxSpreadBasis, uint25
6 _minSpreadCalUSD) external onlyOwner{
83:         require(_spreadBasis <= 10 * VaultMSData.COM_RATE_PRECISION, "ERROR38");
84:         require(_maxSpreadBasis <= MAX_FEE_BASIS_POINTS, "ERROR38");
85:         spreadBasis[_token] = _spreadBasis;
86:         maxSpreadBasis[_token] = _maxSpreadBasis;
87:         minSpreadCalUSD[_token] = _minSpreadCalUSD;
88:     }
89:
90:
91:     function setLiquidator(address _liquidator, bool _isActive) external override onlyOwner {
92:         isLiquidator[_liquidator] = _isActive;
93:     }
94:
95:     function setInPrivateLiquidationMode(bool _inPrivateLiquidationMode) external override onlyOw
ner {
96:         inPrivateLiquidationMode = _inPrivateLiquidationMode;
97:     }
98:
99:     function setPremiumRate(uint256 _premiumBasisPoints, int256 _posIndexMaxPoints, int256 _negIn
dexMaxPoints, uint256 _maxPremiumBasisErrorUSD) external onlyOwner{
100:         require(negIndexMaxPointsPerSec <= 0, "_negIndexMaxPoints be negative");
101:         require(_posIndexMaxPoints >= 0, "_posIndexMaxPoints be positive");
102:         _validate(_premiumBasisPoints <= VaultMSData.COM_RATE_PRECISION, 12);
103:         premiumBasisPointsPerHour = _premiumBasisPoints;
104:         premiumBasisPointsPerSec = hRateToSecRate(premiumBasisPointsPerHour);
105:
106:         negIndexMaxPointsPerHour = _negIndexMaxPoints;
107:         negIndexMaxPointsPerSec = hRateToSecRateInt(negIndexMaxPointsPerHour);
108:
109:         posIndexMaxPointsPerHour = _posIndexMaxPoints;
110:         posIndexMaxPointsPerSec = hRateToSecRateInt(posIndexMaxPointsPerHour);
111:
112:         maxPremiumBasisErrorUSD = _maxPremiumBasisErrorUSD;
113:         // vault.updateRate(address(0));
```

```
114:     }
115:
116:     function setFundingRate(uint256 _fundingRateFactor, uint256 _stableFundingRateFactor) extern
al onlyOwner{
117:         _validate(_fundingRateFactor <= VaultMSData.COM_RATE_PRECISION, 11);
118:         _validate(_stableFundingRateFactor <= VaultMSData.COM_RATE_PRECISION, 12);
119:         fundingRateFactor = _fundingRateFactor;
120:         stableFundingRateFactor = _stableFundingRateFactor;
121:         // vault.updateRate(address(0));
122:     }
123:
124:     function setMaxLeverage(uint256 _maxLeverage) public override onlyOwner{
125:         require(_maxLeverage > VaultMSData.COM_RATE_PRECISION, "ERROR2");
126:         require(_maxLeverage < 220 * VaultMSData.COM_RATE_PRECISION, "Max leverage reached");
127:         maxLeverage = _maxLeverage;
128:     }
129:
130:     function setTaxRate(uint256 _taxMax, uint256 _taxTime) external onlyOwner{
131:         require(_taxMax <= VaultMSData.PRC_RATE_PRECISION, "TAX MAX reached");
132:         if (_taxTime > 0){
133:             taxMax = _taxMax;
134:             taxDuration = _taxTime;
135:         }else{
136:             taxMax = 0;
137:             taxDuration = 0;
138:         }
139:     }
140:
141:     function setFees(
142:         uint256 _taxBasisPoints,
143:         uint256 _stableTaxBasisPoints,
144:         uint256 _mintBurnFeeBasisPoints,
145:         uint256 _swapFeeBasisPoints,
146:         uint256 _stableSwapFeeBasisPoints,
147:         uint256 _marginFeeBasisPoints,
148:         uint256 _liquidationFeeUsd,
149:         uint256 ,
150:         bool _hasDynamicFees
151:     ) external override onlyOwner {
152:         require(_taxBasisPoints <= MAX_FEE_BASIS_POINTS, "3");
153:         require(_stableTaxBasisPoints <= MAX_FEE_BASIS_POINTS, "ERROR4");
154:         require(_mintBurnFeeBasisPoints <= MAX_FEE_BASIS_POINTS, "ERROR5");
```

```
155:         require(_swapFeeBasisPoints <= MAX_FEE_BASIS_POINTS, "ERROR6");
156:         require(_stableSwapFeeBasisPoints <= MAX_FEE_BASIS_POINTS, "ERROR7");
157:         require(_marginFeeBasisPoints <= MAX_FEE_BASIS_POINTS, "ERROR8");
158:         require(_liquidationFeeUsd <= MAX_LIQUIDATION_FEE_USD, "ERROR9");
159:         taxBasisPoints = _taxBasisPoints;
160:         stableTaxBasisPoints = _stableTaxBasisPoints;
161:         mintBurnFeeBasisPoints = _mintBurnFeeBasisPoints;
162:         swapFeeBasisPoints = _swapFeeBasisPoints;
163:         stableSwapFeeBasisPoints = _stableSwapFeeBasisPoints;
164:         marginFeeBasisPoints = _marginFeeBasisPoints;
165:         liquidationFeeUsd = _liquidationFeeUsd;
166:         hasDynamicFees = _hasDynamicFees;
167:     }

 96:     function setAdd(address[] memory _addList) external onlyOwner{
 97:         vaultUtils = IVaultUtils(_addList[0]);
 98:         vaultStorage = _addList[1];
 99:         psbt = IPSBT(_addList[2]);
100:         priceFeed = _addList[3];
101:         feeRouter = _addList[4];
102:     }
103:     function setManager(address _manager, bool _isManager) external override onlyOwner{
104:         isManager[_manager] = _isManager;
105:     }
106:
107:     function setRouter(address _router, bool _status) external override onlyOwner{
108:         approvedRouters[_router] = _status;
109:     }
110:
111:     function setTokenConfig(address _token, uint256 _tokenWeight, bool _isStable, bool _isFundin
gToken, bool _isTradingToken) external override onlyOwner{
112:         if (_isTradingToken && !tradingTokens.contains(_token)) {
113:             tradingTokens.add(_token);
114:         }
115:         if (_isFundingToken && !fundingTokens.contains(_token)) {
116:             fundingTokens.add(_token);
117:         }
118:         VaultMSData.TokenBase storage tBase = tokenBase[_token];
119:
120:         if (_isFundingToken){
121:             totalTokenWeights = totalTokenWeights.add(_tokenWeight).sub(tBase.weight);
```

```
122:             tBase.weight = _tokenWeight;
123:         }
124:         else
125:             tBase.weight = 0;
126:
127:         tBase.isStable = _isStable;
128:         tBase.isFundable = _isFundingToken;
129:         getMaxPrice(_token);// validate price feed
130:     }
131:
132:     function clearTokenConfig(address _token, bool _del) external onlyOwner{
133:         if (tradingTokens.contains(_token)) {
134:             tradingTokens.remove(_token);
135:         }
136:         if (fundingTokens.contains(_token)) {
137:             totalTokenWeights = totalTokenWeights.sub(tokenBase[_token].weight);
138:             fundingTokens.remove(_token);
139:         }
140:         if (_del)
141:             delete tokenBase[_token];
142:     }
```

Description

Kong7ych3 : In the LpManager contract, the owner can modify sensitive parameters through setLP, delLP, and setPSBT, but the event is not recorded. In the Router contract, the owner did not record events when performing setPSBT, setIsSwapOpenForPublic, setMaxSwapRatio, setMaxSwapAmountPerDay, setValidateContract, addPlugin, and removePlugin operations. In the RouterSign contract, the owner can modify sensitive parameters through setPSBT, setPriceFeed, and setVault, but the event is not recorded. In the Vault contract, the owner can modify sensitive parameters through the setAdd, setManager, setRouter, setTokenConfig, and clearTokenConfig functions, but no event recording is performed. In the VaultUtils contract, the owner can modify sensitive parameters through the setMaxProfitRatio, setSpreadBasis, setLiquidator, setInPrivateLiquidationMode, setPremiumRate, setFundingRate, setMaxLeverage, setTaxRate, and setFees functions, but no event recording is performed.

Recommendation

Kong7ych3 : It is recommended to record events when modifying sensitive parameters for subsequent self-examination or community review.

Client Response

Fixed, Event added.

PKO-32:Redundant payable tag

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	Kong7ych3, TrungOre

Code Reference

- code/contracts/core/RouterSign.sol#L68
- code/contracts/core/RouterSign.sol#L95
- code/contracts/core/LpManager.sol#L98
- code/contracts/core/RouterSign.sol#L108
- code/contracts/core/RouterSign.sol#L129
- code/contracts/core/RouterSign.sol#L156
- code/contracts/core/RouterSign.sol#L169

```
68:            uint256[] memory , uint256 ,  address, bytes memory ) external payable nonReentrant{

95:                uint256[] memory , uint256 ,  address , bytes memory ) external payable nonReentr
ant  {

98:    function removeLiquidity(address _lp, uint256 _lpAmount, address _tokenOutOri, uint256 _minOu
t) external override payable nonReentrant returns (uint256) {

108:                uint256[] memory , uint256 ,  address , bytes memory ) external payable nonReent
rant {

129:            uint256[] memory _paras, uint256 _priceTimestamp,  address _updater, bytes memory _u
pdaterSignedMsg) external payable nonReentrant{

156:                uint256[] memory _paras, uint256 _priceTimestamp,  address _updater, bytes memor
y _updaterSignedMsg) external payable nonReentrant  {

169:                uint256[] memory _paras, uint256 _priceTimestamp,  address _updater, bytes memor
y _updaterSignedMsg) external payable nonReentrant {
```

Description

Kong7ych3 : In the RouterSign contract, the increasePosition, decreasePosition, decreasePositionAndSwap, increasePositionAndUpdate, decreasePositionAndUpdate, decreasePositionAndSwapUpdate functions do not need to receive native tokens, but have payable marks. This is redundant.

TrungOre : Certain functions, such as `LpManager.removeLiquidity()`, are labeled as `payable`, despite not involving ETH within their logic. This can potentially result in the loss of ETH in the contracts if users mistakenly invoke the function with attached ETH.

A similar situation arises with the functions `RouterSign.decreasePosition`, `RouterSign.decreasePositionETH`, ...

Recommendation

Kong7ych3 : It is recommended to remove redundant payable tags.

TrungOre : Remove the `payable` attribute from functions that do not have any logic related to handling ETH.

Client Response

Fixed, Redundant payable tag removed.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you ("Customer" or the "Company") in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3's prior written consent in each instance.

This report is not an "endorsement" or "disapproval" of any particular project or team. This report is not an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3's position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.