Smart Contract Security Audit Report

Table Of Contents

1 Executive Summary

On 2024.02.19, the SlowMist security team received the DTX team's security audit application for DTX Protocol, developed the audit plan according to the agreement of both parties and the characteristics of the project, and finally issued the security audit report.

The SlowMist security team adopts the strategy of "white box lead, black, grey box assists" to conduct a complete security test on the project in the way closest to the real attack.

The test method information:

Test method	Description
Black box testing	Conduct security tests from an attacker's perspective externally.
Grey box testing	Conduct security testing on code modules through the scripting tool, observing the internal running status, mining weaknesses.
White box testing	Based on the open source code, non-open source code, to detect whether there are vulnerabilities in programs such as nodes, SDK, etc.

The vulnerability severity level information:

Level	Description
Critical	Critical severity vulnerabilities will have a significant impact on the security of the DeFi project, and it is strongly recommended to fix the critical vulnerabilities.
High	High severity vulnerabilities will affect the normal operation of the DeFi project. It is strongly recommended to fix high-risk vulnerabilities.
Medium	Medium severity vulnerability will affect the operation of the DeFi project. It is recommended to fix medium-risk vulnerabilities.
Low	Low severity vulnerabilities may affect the operation of the DeFi project in certain scenarios. It is suggested that the project team should evaluate and consider whether these vulnerabilities need to be fixed.
Weakness	There are safety risks theoretically, but it is extremely difficult to reproduce in engineering.
Suggestion	There are better practices for coding or architecture.

2 Audit Methodology

The security audit process of SlowMist security team for smart contract includes two steps:

- Smart contract codes are scanned/tested for commonly known and more specific vulnerabilities using automated analysis tools.
- Manual audit of the codes for security issues. The contracts are manually analyzed to look for any potential problems.

Following is the list of commonly known vulnerabilities that was considered during the audit of the smart contract:

Serial Number	Audit Class	Audit Subclass
1	Overflow Audit	-
2	Reentrancy Attack Audit	-
3	Replay Attack Audit	-
4	Flashloan Attack Audit	-
5	Race Conditions Audit	Reordering Attack Audit
6	Permission Vulnerability Audit	Access Control Audit
		Excessive Authority Audit
7	Security Design Audit	External Module Safe Use Audit
		Compiler Version Security Audit
		Hard-coded Address Security Audit
		Fallback Function Safe Use Audit
		Show Coding Security Audit
		Function Return Value Security Audit
		External Call Function Security Audit

Serial Number	Audit Class	Audit Subclass
7	Security Design Audit	Block data Dependence Security Audit
		tx.origin Authentication Security Audit
8	Denial of Service Audit	-
9	Gas Optimization Audit	-
10	Design Logic Audit	-
11	Variable Coverage Vulnerability Audit	-
12	"False Top-up" Vulnerability Audit	-
13	Scoping and Declarations Audit	-
14	Malicious Event Log Audit	-
15	Arithmetic Accuracy Deviation Audit	-
16	Uninitialized Storage Pointer Audit	-

3 Project Overview

3.1 Project Introduction

DTX Protocol is a P2Pool2P decentralized perpetual contract protocol.

The core focus of this audit was on the order book, which enables user interaction, and the position management module, which allows the keeper to operate. The oracle module is used to fetch the price of tokens from Pyth and FastPriceFeed. Liquidity management is mainly represented in the Market module and the LiquidityHandler module, where the user's liquidity is stored in the corresponding vault.

The FeeHandler module is used to calculate and update the fees that are charged during trading, such as the funding rate, the opening/closing rate, and the borrow fees.

3.2 Vulnerability Information

The following is the status of the vulnerabilities found in this audit:

NO	Title	Category	Level	Status
N1	Lack of authority control	Authority Control Vulnerability Audit	Critical	Fixed
N2	Liquidity incorrectly updated upon liquidation	Design Logic Audit	High	Fixed
N3	Redundant code	Others	Suggestion	Acknowledged
N4	Lack of checks in the direction of price triggers	Design Logic Audit	Medium	Fixed
N5	Lack of cost checks for the execution of orders	Design Logic Audit	High	Fixed
N6	call() should be used instead of transfer() and send()	Gas Optimization Audit	Suggestion	Fixed
N7	Potential issues with using chainlink to get prices	Others	Low	Fixed
N8	Rounding issues when transferring the fee	Arithmetic Accuracy Deviation Vulnerability	High	Fixed
N9	Lack of deduction of fees receivable in liquidation check	Design Logic Audit	Medium	Ignored
N10	Lack of deducting the fund fee to take-profit checks	Design Logic Audit	Medium	Ignored
N11	Risk of excessive authority	Authority Control Vulnerability Audit	Medium	Acknowledged
N12	Lack of long/short position spread judgement	Design Logic Audit	High	Ignored
N13	Missing event record	Others	Suggestion	Fixed
N14	Lack of liquidity checking mechanism	Design Logic Audit	High	Fixed

NO	Title	Category	Level	Status
	in case of position reduction			

4 Code Overview

4.1 Contracts Description

Audit Version:

https://github.com/dtx-trade/dtx-contracts

commit: df761be1a459fca8c6bb3a7deecb9a9164727214

Fixed Version:

https://github.com/dtx-trade/dtx-contracts

commit: f5ce7fe48893a4575732608bb94f24f1aa77bfe1

The main network address of the contract is as follows:

The code was not deployed to the mainnet.

4.2 Visibility Description

The SlowMist Security team analyzed the visibility of major contracts during the audit, the result as follows:

Faucet			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
faucet	Public	Can Modify State	-

FeeHandler			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM

FeeHandler			
updateFundingAndBorrowingState	External	Can Modify State	onlyPositionManager
incrementClaimableFundingAmount	External	Can Modify State	onlyPositionManager
setIsClaimFundingFees	External	Can Modify State	-
isClaimFundingFees	External	-	-
claimFundingFees	External	Can Modify State	onlySystem
getClaimFundingFees	External	-	-
getMarketClaimFundingFees	External	-	-
applyDeltaToOpenInterest	External	Can Modify State	onlyPositionManager
getMarginFee	Public	-	-
getLiquidationFee	External	-	-
getFundingAmountPerSize	Public	-	-
getClaimableFundingAmountPerSize	Public	-	-
getBorrowingRate	Public	-	-
getCumulativeBorrowingRate	Public	-	-
getCumulativeBorrowingRateDuration	Public	-	-
getNextCumulativeBorrowingRate	Public	-	-
getBorrowingFees	External	-	-
getFundingFees	External	-	-
getFundingAmount	Public	-	-
getLastFundingRate	Public	-	-
getNextFundingRate	Public	-	-
getNextFundingRateState	Public	-	-

FeeHandler			
getLastFundingRateDuration	Public	-	-
getFundingAmountPerSizeDelta	Public	-	-
getVaultOpenInterest	Public	-	-
getMarketOpenInterest	Public	-	-
_updateFundingState	Private	Can Modify State	-
_updateCumulativeBorrowingRate	Private	Can Modify State	-
_applyDeltaToFundingPerSize	Private	Can Modify State	-
_applyDeltaToClaimableFundingPerSize	Private	Can Modify State	-
_setFundingRate	Private	Can Modify State	-
toBasePrice	Public	-	-
treasuryFactor	External	-	-
getFeeConfig	Public	-	-

Hayek			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
transferFrom	External	Can Modify State	onlySystem

IAM			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
_validateRole	Internal	-	-
owner	External	-	-

Keeper			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
listPendingOrdersPaginated	External	-	-
listMarketOrdersPaginated	External	-	-
listVaultOrdersPaginated	External	-	-
setPricesAndExecOrder	External	Can Modify State	onlyKeeper
execVaultOrders	External	Can Modify State	onlyKeeper
_executeIncreaseOrder	Private	Can Modify State	-
_executeDecreaseOrder	Private	Can Modify State	-
canExecuteWithPrice	Public	-	-
canLiquidateWithPrice	Public	-	-
setPricesAndLiquidatePosition	External	Can Modify State	onlyKeeper
_setVaultPrice	Private	Can Modify State	-

LiquidityHandler			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
updateOccupiedLiquidity	External	Can Modify State	onlyPositionManager
getLiquidityConfig	External	-	-
hasLiquidity	External	-	-
getAvailableLiquidity	Public	-	-
getVaultUsedLiquidity	External	-	-
getVaultAvailableLiquidity	Public	-	-

MarketRouter			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
setSysAddresses	External	Can Modify State	onlyAdmin
setMarketConfigsByVault	External	Can Modify State	onlyAdmin
claimFundingFees	External	Can Modify State	nonReentrant
addMarket	Public	Can Modify State	onlyAdmin
addMarkets	External	Can Modify State	onlyAdmin
updateMarkets	External	Can Modify State	onlyAdmin
updateIndexTokenMetas	External	Can Modify State	onlyAdmin
allAssets	External	-	-
allVaults	External	-	-
allIndexTokens	External	-	-
getMarket	External	-	-
getMarket	External	-	-
getIndexTokensByVault	External	-	-
getIndexTokenMeta	External	-	-
oracle	External	-	-
hayek	External	-	-
positionManager	External	-	-
orderBook	External	-	-
nft	External	-	-
feeData	External	-	-

MarketRouter			
marketData	External	-	-
orderData	External	-	-
positionData	External	-	-
marketVault	External	-	-
treasury	External	-	-
configData	External	-	-

NFT			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM ERC721
mint	Public	Can Modify State	-
_mint	Internal	Can Modify State	-
_genTokenId	Internal	Can Modify State	-
_removeTokenId	Internal	Can Modify State	-
_addTokenId	Internal	Can Modify State	-
_update	Internal	Can Modify State	-
userTokenIds	External	-	-

OrderHandler			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
validateIncreaseOrder	External	-	-
validateDecreaseOrder	External	-	-

OrderHandler			
validateDecreaseRequest	External	-	-
validateUpdateOrder	External	-	-
validateCancelOrders	External	-	-
validateCancelOrder	External	-	-
canExecuteWithPrice	External	-	-
canExecuteWithPrice	External	-	-
getOrderConfig	Public	-	-

PositionManager			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
increasePosition	External	Can Modify State	-
decreasePosition	External	Can Modify State	-
previewIncreasePosition	External	-	-
previewDecreasePosition	External	-	-
liquidatePosition	External	Can Modify State	onlyLiquidator
settleVaultProfit	Public	Can Modify State	onlyPositionManager
getPosition	Public	-	-
getPositionByKey	External	-	-
getPositionsKeys	External	-	-
getPositionLength	External	-	-
validateLiquidation	External	-	-

PositionHandler			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
validatePosition	Public	-	-
getPositionConfig	External	-	-
isExistsPosition	Public	-	-
getPositionFee	Public	-	-
_transferLiquidationFee	Private	Can Modify State	-
_transferLiquidationTPFee	Private	Can Modify State	-
validateLiquidation	Public	-	-
liquidatePosition	External	Can Modify State	-
getLastPosition	External	-	-
getDecreasePosition	External	-	-
updateGlobalPosition	External	Can Modify State	onlyPositionManager

Reader			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
getPositionsByAccount	Public	-	-
getPositionsByTokenId	Public	-	-
getPositionsKeys	Public	-	-
getPositionLength	Public	-	-
getTokenIdOrdersLength	Public	-	-
getOrdersByAccount	Public	-	-

Reader			
getOrdersByTokenId	Public	-	-
getOrderByOrderId	Public	-	-
getOrderIds	Public	-	-
getVaultInfos	External	-	-
getVaultWL	Public	-	-
getIndexTokensInfoByVault	Public	-	-
getAssetInfo	Public	-	-
getAllAssetsInfo	Public	-	-
getMarketInfo	External	-	-
getMarketsConfigs	External	-	-
getPositionFee	Public	-	-
getClaimableFundingAmount	External	-	-
getGlobalPositionPNL	External	-	-

RoleStore			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	AccessManager Ownable
requireNotPaused	Public	-	-

OrderBook			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
createIncreaseOrder	External	Payable	-

OrderBook			
_generateTpSlOrders	Private	Can Modify State	-
_storeTPSLOrderPair	Private	Can Modify State	-
createDecreaseOrders	External	Payable	-
createDecreaseOrder	External	Payable	-
createVaultOrder	External	Payable	-
executeVaultOrder	External	Can Modify State	onlyKeeper
_transferExecFee	Private	Can Modify State	-
_createDecreaseOrder	Private	Can Modify State	-
updateOrder	External	Can Modify State	-
cancelOrders	External	Can Modify State	-
cancelOrder	Public	Can Modify State	-
afterOrderExec	External	Can Modify State	onlyKeeper

VaultFactory			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
deployVault	Public	Can Modify State	onlyVaultManager

Vault			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	ERC4626 ERC20 IAM
updateMeta	External	Can Modify State	onlyVaultManager
addWhiteList	External	Can Modify State	onlyVaultManager

Vault			
removeWhiteList	External	Can Modify State	onlyVaultManager
isAddressInWhitelist	External	-	-
getWhiteLists	External	-	-
getWhiteListsPaginated	External	-	-
reduceAssets	External	Can Modify State	onlyPositionManager
setPrice	External	Can Modify State	onlySystem
setOrderConfig	External	Can Modify State	onlyAdmin
getVaultMeta	External	-	-
_withdraw	Internal	Can Modify State	whenNotPaused
_deposit	Internal	Can Modify State	whenNotPaused
_convertToShares	Internal	-	-
_convertToAssets	Internal	-	-
getVaultOrderConfig	External	-	-

MarketVault			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-
transferOut	External	Can Modify State	-
transferOutNative	External	Can Modify State	-

Treasury			
Function Name	Visibility	Mutability	Modifiers
<Receive Ether>	External	Payable	-

Treasury			
transferOut	External	Can Modify State	-
transferOutNative	External	Can Modify State	-

MultiCall			
Function Name	Visibility	Mutability	Modifiers
call	External	Can Modify State	-

ChainPriceFeed			
Function Name	Visibility	Mutability	Modifiers
setSampleSpace	External	Can Modify State	-
setPriceFeed	External	Can Modify State	-
getLatestPrice	Public	-	-
getPrice	Public	-	-

FastPriceFeed			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	-
setPriceFeed	External	Can Modify State	-
setMaxTimeDeviation	External	Can Modify State	-
setPriceDuration	External	Can Modify State	-
setMaxPriceUpdateDelay	External	Can Modify State	-
setSpreadBasisPointsIfInactive	External	Can Modify State	-
setSpreadBasisPointsIfChainError	External	Can Modify State	-
setMinBlockInterval	External	Can Modify State	-

FastPriceFeed			
setIsSpreadEnabled	External	Can Modify State	-
setLastUpdatedAt	External	Can Modify State	-
setMaxDeviationBasisPoints	External	Can Modify State	-
setMaxCumulativeDeltaDiffs	External	Can Modify State	-
setPriceDataInterval	External	Can Modify State	-
setTokens	External	Can Modify State	-
setPrices	External	Can Modify State	-
setCompactedPrices	External	Can Modify State	-
setPricesWithBits	External	Can Modify State	-
getPrice	External	-	-
favorFastPrice	Public	-	-
getPriceData	Public	-	-
_setPricesWithBits	Private	Can Modify State	-
_setPrice	Private	Can Modify State	-
_setPriceData	Private	Can Modify State	-
_setLastUpdatedValues	Private	Can Modify State	-

Oracle			
Function Name	Visibility	Mutability	Modifiers
setAdjustment	External	Can Modify State	-
setFastPriceEnabled	External	Can Modify State	-
setFastPriceFeed	External	Can Modify State	-
setChainPriceFeed	External	Can Modify State	-

Oracle			
setSpreadBasisPoints	External	Can Modify State	-
setSpreadThresholdBasisPoints	External	Can Modify State	-
setMaxStrictPriceDeviation	External	Can Modify State	-
setStableTokens	External	Can Modify State	-
getPrices	External	-	-
getPrice	Public	-	-
_getPrice	Internal	-	-
getChainPrice	Public	-	-
getFastPrice	Public	-	-

ConfigData			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
setOrderConfig	Public	Can Modify State	onlySystem
setPositionConfig	Public	Can Modify State	onlySystem
setFeeConfig	Public	Can Modify State	onlySystem
setLiquidityConfig	Public	Can Modify State	onlySystem

FeeData			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
setAccBorrowingRate	External	Can Modify State	onlyFeeHandler
setFundingRate	External	Can Modify State	onlyFeeHandler

FeeData			
setFundingAmountPerSize	External	Can Modify State	onlyFeeHandler
setClaimableFundingAmountPerSize	External	Can Modify State	onlyFeeHandler
setClaimableFundingAmount	External	Can Modify State	onlyFeeHandler
setMisc	External	Can Modify State	onlyFeeHandler

MarketData			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
setOpenInterest	External	Can Modify State	onlyFeeHandler
increaseVaultOccupied	External	Can Modify State	onlyLiquidityHandler
decreaseVaultOccupied	External	Can Modify State	onlyLiquidityHandler
registerVault	External	Can Modify State	onlyVaultFactory
registerIndexToken	External	Can Modify State	onlyMarketRouter
updateIndexTokenMeta	External	Can Modify State	onlyMarketRouter
updateMarket	External	Can Modify State	onlyMarketRouter
allAssets	External	-	-
allVaults	External	-	-
allIndexTokens	External	-	-
getMarket	External	-	-
getMarket	External	-	-
getIndexTokensByVault	External	-	-
getIndexTokenMeta	External	-	-

OrderData			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
setTPSLPairs	External	Can Modify State	onlySystem
removeTPSLPairs	External	Can Modify State	onlySystem
genIncreaseOrderId	External	Can Modify State	onlySystem
genDecreaseOrderId	External	Can Modify State	onlySystem
storeIncreaseOrder	External	Can Modify State	onlyOrderBook
storeDecreaseOrder	External	Can Modify State	onlyOrderBook
deleteIncreaseOrder	External	Can Modify State	onlySystem
deleteDecreaseOrder	External	Can Modify State	onlySystem
updateIncreaseOrder	External	Can Modify State	onlyOrderBook
updateDecreaseOrder	External	Can Modify State	onlyOrderBook
storeVaultOrder	External	Can Modify State	onlyOrderBook
deleteVaultOrder	External	Can Modify State	onlyOrderBook
getAllOrderIds	External	-	-
getAllMarketOrderIds	External	-	-
getMarketOrderIdsPaginated	External	-	-
getOrderIdsPaginated	External	-	-
getVaultOrderIdsPaginated	Public	-	-
getTokenIdOrderIds	External	-	-
isExistsOrder	External	-	-
getTokenIdOrdersLength	External	-	-

OrderData			
getPositionPendingOrderIds	External	-	-

PositionData			
Function Name	Visibility	Mutability	Modifiers
<Constructor>	Public	Can Modify State	IAM
updatePosition	External	Can Modify State	onlyPositionManager
removePosition	External	Can Modify State	onlyPositionManager
getTokenIdPositionKeys	External	-	-
getTokenIdPositionCount	External	-	-

4.3 Vulnerability Summary

[N1] [Critical] Lack of authority control

Category: Authority Control Vulnerability Audit

Content

1.In the PositionManager contract, the increasePosition and decreasePosition functions are used to open and decrease positions respectively. However, both of these functions lack permission validation, resulting in any external user being able to pass in custom order information parameters to open or close a position.

For the createPosition function, the user can call it directly and pass in the valid order information parameter to open a position without transferring margin. For decreasePosition positions, the user can directly close any position that exists in the market by calling and passing in a valid order information parameter.

Code Location:

contracts/PositionManager.sol

```
    function increasePosition(
        Order.IncreaseOrder memory order
    ) external returns (bool, Order.CancelType, uint256, uint256) {
        ...
```

```
    }
    ...
function decreasePosition(
        Order.DecreaseOrder memory decreaseOrder
    )
        external
        returns (bool, Order.CancelType, uint256, uint256, bool, uint256)
    {
        ...
    }
```

2.In the MarketVault and Treasury contract, anyone can transfer tokens out of a contract without restriction by calling the transferOut and transferOutNative functions.

Code Location:

contracts/vault/MarketVault.sol

```
    function transferOut(
        address token,
        address receiver,
        uint256 amount
    ) external {
        if (amount == 0) return;
        IERC20(token).safeTransfer(receiver, amount);
    }

    function transferOutNative(
        address payable receiver,
        uint256 amount
    ) external {
        if (amount == 0) return;
        receiver.transfer(amount);
    }
```

contracts/vault/Treasury.sol

```
    function transferOut(
        address token,
        address receiver,
        uint256 amount
    ) external {
        IERC20(token).safeTransfer(receiver, amount);
    }
```

```
    function transferOutNative(
        address payable receiver,
        uint256 amount
    ) external {
        receiver.transfer(amount);
    }
```

3.In the Oracle contract, anyone can set oracle machine price configurations such as price source, adjustmentBasisPoints, spreadThresholdBasisPoints and token attributes without restriction by calling the setAdjustment, setFastPriceEnabled, setFastPriceFeed, setChainPriceFeed, setSpreadBasisPoints, setSpreadThresholdBasisPoints, setMaxStrictPriceDeviation and setStableTokens functions.

Code Location:

contracts/oracle/Oracle.sol#L40-107

```
    function setAdjustment(
        address _token,
        bool _isAdditive,
        uint256 _adjustmentBps
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setFastPriceEnabled(
        bool _isEnabled
    ) external /* onlyInitOr(MANAGER_ROLE) */ {
        ...
    }

    function setFastPriceFeed(
        address _feed
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setChainPriceFeed(
        address _feed
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setSpreadBasisPoints(
        address _token,
        uint256 _spreadBasisPoints
```

```
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setSpreadThresholdBasisPoints(
        uint256 _spreadThresholdBasisPoints
    ) external /*onlyInitOr(MANAGER_ROLE) */ {
        ...
    }

    function setMaxStrictPriceDeviation(
        uint256 _maxStrictPriceDeviation
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setStableTokens(
        address _token,
        bool _stable
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }
```

4.In the ChainPriceFeed contract, anyone can set the price feed configurations such as the sampleSpace, the priceFeeds and the priceDecimals without restriction by calling the setSampleSpace and setPriceFeed functions.

Code Location:

contracts/oracle/ChainPriceFeed.sol#15-27

```
    function setSampleSpace(uint256 _times) external /*onlyAdmin*/ {
        ...
    }

    function setPriceFeed(
        address _token,
        address _feed,
        uint256 _decimal
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }
```

5.In the FastPriceFeed contract, anyone can set the price feed configurations such as the chainPriceFeed, the maxTimeDeviation, the priceDuration, the maxPriceUpdateDelay, the spreadBasisPointsIfInactive, the

spreadBasisPointsIfChainError, the minBlockInterval, the isSpreadEnablede, the lastUpdatedAt, the maxDeviationBasisPoints, the maxCumulativeDeltaDiffs, the priceDataInterval, the tokens and the tokens' price without restriction by calling the setPriceFeed, the setMaxTimeDeviation, the setPriceDuration, the setMaxPriceUpdateDelay, the setSpreadBasisPointsIfInactive, the setSpreadBasisPointsIfChainError, the setMinBlockInterval, the setIsSpreadEnabled, the setLastUpdatedAt, the setMaxDeviationBasisPoints, the setMaxCumulativeDeltaDiffs, the setPriceDataInterval, the setTokens, the setPrices, the setCompactedPrices, the setPricesWithBits functions.

Code Location:

contracts/oracle/FastPriceFeed.sol#103-247

```
    function setPriceFeed(address _feed) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setMaxTimeDeviation(
        uint256 _deviation
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setPriceDuration(
        uint256 _duration
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setMaxPriceUpdateDelay(
        uint256 _delay
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setSpreadBasisPointsIfInactive(
        uint256 _point
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setSpreadBasisPointsIfChainError(
        uint256 _point
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
```

```
        ...
    }

    function setMinBlockInterval(
        uint256 _interval
    ) external /* onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setIsSpreadEnabled(
        bool _enabled
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setLastUpdatedAt(
        uint256 _lastUpdatedAt
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setMaxDeviationBasisPoints(
        uint256 _maxDeviationBasisPoints
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setMaxCumulativeDeltaDiffs(
        address[] memory _tokens,
        uint256[] memory _maxCumulativeDeltaDiffs
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setPriceDataInterval(
        uint256 _priceDataInterval
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setTokens(
        address[] memory _tokens,
        uint256[] memory _tokenPrecisions
    ) external /*onlyInitOr(MANAGER_ROLE)*/ {
        ...
    }

    function setPrices(
        address[] memory _tokens,
```

```
        uint256[] memory _prices,
        uint256 _timestamp
    ) external /*onlyUpdater*/ {
        ...
    }

    function setCompactedPrices(
        uint256[] memory _priceBitArray,
        uint256 _timestamp
    ) external /*onlyUpdater*/ {
        ...
    }

    function setPricesWithBits(
        uint256 _priceBits,
        uint256 _timestamp
    ) external /*onlyUpdater*/ {
       ...
    }
```

Solution

It is recommended to add strict permission checks to functions that set the core parameters of the contract.

Status

Fixed

[N2] [High] Liquidity incorrectly updated upon liquidation

Category: Design Logic Audit

Content

In the PositionManager contract, the Liquidator role can liquidate a specified position by calling the liquidatePosition function. The actual liquidation logic is done by calling the liquidatePosition function in the PositionHandler contract.

In the liquididatePosition function of the PositionHandler contract, the updateOccupiedLiquidity function is called to update the liquidity occupancy after passing the check. However, in the liquididatePosition function of the PositionManager contract, the liquidity occupancy is updated again after the liquidation operation is completed. Repeated update operation is not designed to be expected and may result in liquidity deductions beyond what is expected.

Code Location:

contracts/PositionManager.sol#L553-557

```
    function liquidatePosition(
        Keys.PositionKeyInfo memory keyInfo
    ) external onlyLiquidator returns (bool isSuccess, string memory res) {
        ...
        (isSuccess, res, _realisedPnL) = market
            .positionHandler
            .liquidatePosition(_position);
        if (!isSuccess) {
            return (isSuccess, "liquidate error");
        }

        market.updateOccupiedLiquidity(
            _position.collateral,
            _position.isLong,
            false
        );
        ...
    }
```

Solution

It is recommended to remove the operation of updating the liquidity occupancy in the liquidatePosition function of the PositionManager contract.

Status

Fixed

[N3] [Suggestion] Redundant code

Category: Others

Content

There are useless codes in the file and codes that are not used in actual business.

Code Location:

contracts/PositionManager.sol#L365-525

```
    function previewIncreasePosition(
        Order.IncreaseOrder memory increaseOrder,
        Market.Price memory collateralTokenPrice,
```

```
        Market.Price memory indexTokenPrice
    )
        external
        view
        returns (
            Position.Props memory position,
            Position.FundDelta memory fundDelta,
            bool canExecute,
            string memory reason
        )
    {
        ...
    }
    ...
    function previewDecreasePosition(
        Order.DecreaseOrder memory decreaseOrder,
        Market.Price memory collateralTokenPrice,
        Market.Price memory indexTokenPrice
    )
        external
        view
        returns (
            Position.Props memory position,
            Position.FundDelta memory fundDelta,
            bool canExecute,
            string memory reason
        )
    {
        ...
    }
```

contracts/data/MarketData.sol

```
    mapping(address => EnumerableSet.AddressSet) private vaultIndexTokens;

    ...

    function getIndexTokensByVault(
        address collateralTokenVault
    ) external view returns (address[] memory) {
        return vaultIndexTokens[collateralTokenVault].values();
    }
```

contracts/MarketRouter.sol

```
    function getIndexTokensByVault(
        IVault collateralTokenVault
    ) external view returns (address[] memory) {
        return this.marketData().getIndexTokensByVault(collateralTokenVault);
    }
```

Solution

It is recommended to remove redundant commented code and useless code.

Status

Acknowledged; Project team response: this part of the functions need to be left to the front-end.

[N4] [Medium] Lack of checks in the direction of price triggers

Category: Design Logic Audit

Content

In the OrderBook contract, the contract does not check isTriggerAbove and keyInfo.isLong in the incoming order information parameters when the user creates and modifies an order.

Suppose there is a situation where the user creates a limit order to open a long position and the incoming isTriggerAbove parameter is set to true by mistake, which would result in the order being executed immediately even if the price is not at the trigger price at the time the order is executed.

Code Location:

contracts/OrderBook.sol

```
    function createIncreaseOrder(
        Order.IncreaseOrder memory request
    ) external payable {
        ...
    }

function createDecreaseOrders(
        Order.DecreaseOrder[] memory orders
    ) external payable {
        ...
    }
    function createDecreaseOrder(
        Order.DecreaseOrder memory request
    ) external payable {
        ...
```

```
    }

...

function updateOrder(Order.UpdateOrder calldata request) external {
        ...
    }
```

Solution

It is recommended that the trigger price direction and long/short direction be checked against each other during order creation and order modification.

Status

Fixed

[N5] [High] Lack of cost checks for the execution of orders

Category: Design Logic Audit

Content

In the OrderBook contract, users can create decrease orders in bulk by calling the createDecreaseOrders function.

However, there is no check in the function to see if the sum of the incoming executionFee parameter is equal to msg.value. if the sum of the incoming executionFee is much greater than the value of msg.value, then this will result in unintended consumption of funds in the vault.

Code Location:

contracts/OrderBook.sol#L125-156

```
    function createDecreaseOrders(
        Order.DecreaseOrder[] memory orders
    ) external payable {

        uint256 _execFee;
        uint256 orderLen = orders.length;

        for (uint256 i; i < orderLen; i++) {
            _execFee += orders[i].executionFee;
        }

        uint256 _need;
```

```
        for (uint256 i = 0; i < orderLen; i++) {
            uint256 executionFeeRequired = orders[i]
                .keyInfo
                .collateralTokenVault
                .getVaultOrderConfig()
                .executionFee;

            _need += executionFeeRequired;
        }

        require(msg.value >= _need, "invalid execution fee");

        _transferExecFee(msg.value);

        ...
    }
```

Solution

It is recommended to check if the sum of the incoming executionFee parameters is equal to msg.value.

Status

Fixed

[N6] [Suggestion] call() should be used instead of transfer() and send()

Category: Gas Optimization Audit

Content

The transfer() and send() functions forward a fixed amount of 2300 gas. Historically, it has often been recommended to use these functions for value transfers to guard against reentrancy attacks. However, the gas cost of EVM instructions may change significantly during hard forks which may break already deployed contract systems that make fixed assumptions about gas costs. For example. EIP 1884 broke several existing smart contracts due to a cost increase of the SLOAD instruction.

Code location:

contracts/vault/MarketVault.sol#L29

```
    function transferOutNative(
        address payable receiver,
        uint256 amount
    ) external {
        if (amount == 0) return;
```

```
        receiver.transfer(amount);
    }
```

contracts/vault/Treasury.sol#L28

```
    function transferOutNative(
        address payable receiver,
        uint256 amount
    ) external {
        receiver.transfer(amount);
    }
```

Solution

It is recommended to use call() instead of transfer(), but be sure to respect the CEI pattern or add re-entrancy guards, and the return value should be checked.

Status

Fixed

[N7] [Low] Potential issues with using chainlink to get prices

Category: Others

Content

In the ChainPriceFeed contract, the getLatestPrice function and the getPrice function will obtain the price from chainlink's price feed contract through the latestAnswer interface. But the latestAnswer interface only returns the price parameter, without any other parameters to check whether the price is valid. For this reason, chainlink has deprecated this interface in v3.

In addition, the getRoundData function is used in the getPrice function to get the price, but there is no check if the return value indicates stale data(updateAt refers to the timestamp of the round. This value isn't checked to make sure it is recent).

Reference:

https://docs.chain.link/data-feeds/api-reference/#latestanswer

https://docs.chain.link/data-feeds/#check-the-timestamp-of-the-latest-answer

Code Location:

contracts/oracle/ChainPriceFeed.sol

```
    function getLatestPrice(address _token) public view returns (uint256) {
        ...

        int256 _price = _priceFeed.latestAnswer();
        require(_price > 0, "PriceFeed: invalid price");

        return uint256(_price);
    }

    function getPrice(
        address _token,
        bool _maximise
    ) public view returns (uint256) {
        ...

        uint80 _id = _priceFeed.latestRound();

        for (uint80 i = 0; i < sampleSpace; i++) {
            if (_id <= i) {
                break;
            }
            uint256 p;

            if (i == 0) {
                int256 _p = _priceFeed.latestAnswer();
                require(_p > 0, "PriceFeed: invalid price");
                p = uint256(_p);
            } else {
                (, int256 _p, , , ) = _priceFeed.getRoundData(_id - i);
                require(_p > 0, "PriceFeed: invalid price");
                p = uint256(_p);
            }

            ...
        }

        ...
    }
}
```

Solution

It is recommended to use the latestRoundData interface instead of latestAnswer, and check the updatedAt

parameter to ensure that the price obtained is fresh.

Status

Fixed; Response from the project team: ditch chainlink and use pyth to get prices instead.

[N8] [High] Rounding issues when transferring the fee

Category: Arithmetic Accuracy Deviation Vulnerability

Content

In the PositionManager contract, after the execution of opening and decreasing positions, the contract will call the transferIncreaseFee function in the market library to transfer the fee to be charged.

In the transferIncreaseFee function of the market contract, The cost of transferring to the vault is first calculated, after which a portion of the cost that needs to be transferred to the treasury address is deducted from it.

However, when calculating the number of tokens to be transferred to treasury, the Precision library's applyFactor function is used, which divides an extra FLOAT_PRECISION parameter(i.e., 10 ** 30). This will eventually result in the number of tokens transferred being incorrectly calculated as 0 due to rounding issues.

Code Location:

contracts/PositionManager.sol

```
    function increasePosition(
        Order.IncreaseOrder memory order
    ) external returns (bool, Order.CancelType, uint256, uint256) {
        ...

        market.transferIncreaseFee(
            _fees,
            marketRouter,
            market.getFeeConfig().treasuryFactor,
            collateralTokenPrice
        );

        ...
    }

    ...

    function decreasePosition(
        Order.DecreaseOrder memory decreaseOrder
    )
```

```
        external
        returns (bool, Order.CancelType, uint256, uint256, bool, uint256)
    {
        ...

        market.transferIncreaseFee(
            _fees,
            marketRouter,
            market.getFeeConfig().treasuryFactor,
            collateralTokenPrice
        );

        ...
    }
```

contracts/libs/Market.sol#L387-388

```
    function transferIncreaseFee(
        Props memory market,
        Fee.FeeDelta memory feeDelta,
        IMarketRouter marketRouter,
        uint256 treasuryFactor,
        Market.Price memory collateralTokenPrice
    ) external {
        ...

        uint256 _amount = Precision.applyFactor(_treasuryFee, 10 ** _decimals) /
            collateralTokenPrice.max;
        marketRouter.marketVault().transferOut(
            _token,
            address(marketRouter.treasury()),
            _amount
        );
    }
```

Solution

Instead of calling the applyFactor function, it is recommended to multiply the fee and token precision directly.

Status

Fixed

[N9] [Medium] Lack of deduction of fees receivable in liquidation check

Category: Design Logic Audit

Content

In the positionHandler contract, an external call to the validateLiquidation function allows you to check whether an incoming position can be liquidated. A position can be liquidated if the amount of the loss of the position plus the total cost of the position is greater than the margin.

However, the fee is calculated without deducting the funding fee that the position should receive, so there is a situation where the position does not need to pay the funding fee at the moment but receives it from the counterparty, and if this is not deducted it may result in the position being incorrectly calculated as liquidatable.

Code Location:

contracts/positionHandler.sol

```
    function validateLiquidation(
        Position.Props memory position,
        Market.Price memory collateralTokenPrice,
        uint256 markPrice,
        uint256 cumulativeBorrowingRate
    )
        public
        view
        returns (
            Position.PositionStatus status,
            Position.FundDelta memory fundDelta
        )
    {
        ...

        fundDelta.totalFee = fees.totalFee();

        // get position config in market
        Market.PositionConfig memory _config = _router
            .configData()
            .positionConfigs(_marketKey);

        // Collateral - Funding Fee - Borrow Fee - Liquidation Fee - PnL <= Collateral
* 1000bps
        if (!hasProfit) {
            uint256 _limit = position.collateral -
                Precision.mulDiv(
                    position.collateral,
                    _config.mmr,
                    Precision.BASIS_POINTS
```

```
                );
            uint256 _loss = fundDelta.totalFee + unrealisedPnL;

            if (_limit <= _loss) {
                return (Position.PositionStatus.LIQUIDATION, fundDelta);
            }
        } else {
            ...
        }

        return (Position.PositionStatus.OPEN, fundDelta);
    }
```

contracts/libs/Fee.sol

```
    function totalFee(FeeDelta memory fees) external pure returns (uint256) {
        return
            fees.marginFee +
            fees.execFee +
            fees.liquidationFee +
            fees.borrowingFee +
            fees.fundingFeeUSD;
    }
```

Solution

It is recommended that the portion of the cost of the funds to be obtained be deducted from the calculation of the total fee at the time of the liquidation inspection.

Status

Ignored; Response from the project team: The mechanism is in line with the intended design.

[N10] [Medium] Lack of deducting the fund fee to take-profit checks

Category: Design Logic Audit

Content

In the positionHandler contract, an external call to the validateLiquidation function allows you to check whether an incoming position triggers Take Profit. However, it only checks whether the profit of the position exceeds the maximum threshold, and does not deduct the fees that need to be charged from the profit, such as fund fees and position opening fees. This may result in positions being incorrectly taken out early.

Code Location:

contracts/positionHandler.sol

```
    function validateLiquidation(
        Position.Props memory position,
        Market.Price memory collateralTokenPrice,
        uint256 markPrice,
        uint256 cumulativeBorrowingRate
    )
        public
        view
        returns (
            Position.PositionStatus status,
            Position.FundDelta memory fundDelta
        )
    {
        ...

        if (!hasProfit) {
            ...

        } else {

            uint256 _maxPnL = Precision.mulDiv(
                position.collateral,
                _config.maxPnLBps,
                Precision.BASIS_POINTS
            );

            if (unrealisedPnL >= _maxPnL) {
                return (Position.PositionStatus.TP, fundDelta);
            }
        }

        return (Position.PositionStatus.OPEN, fundDelta);
    }
```

Solution

It is recommended to deduct the required fees from the current profit of the position when checking whether the position has reached the take-profit state.

Status

Ignored; Response from the project team: The mechanism is consistent with the intended design.

[N11] [Medium] Risk of excessive authority

Category: Authority Control Vulnerability Audit

Content

1.In the Keeper contract, the Keeper role can arbitrarily set the price of tokens when executing orders. If the privilege is lost or misused, this may have an impact on the user's assets.

Code Location:

contracts/Keeper.sol

```
    function setPricesAndExecOrder(
        address[] memory _tokens,
        uint256[] memory _prices,
        uint256[] memory orderIds,
        uint256 _timestamp,
        address payable receiver
    )
        external
        onlyKeeper
        returns (bool[] memory isSuccess, string[] memory res)
    {
        address _fastFeed = marketRouter.oracle().fastPriceFeed();
        IFastPriceFeed(_fastFeed).setPrices(_tokens, _prices, _timestamp);

        ...
    }

    function execVaultOrders(
        address[] memory vaults,
        uint256[] memory vaultPrices,
        uint256[] memory orderIds,
        address payable receiver
    )
        external
        onlyKeeper
        returns (bool[] memory isSuccess, string[] memory res)
    {
        require(vaults.length == vaultPrices.length, "invalid params");

        for (uint256 i = 0; i < vaults.length; i++) {
            _setVaultPrice(vaults[i], vaultPrices[i]);
        }

        ...
    }
```

```
...

function setPricesAndLiquidatePosition(
    address[] memory _tokens,
    uint256[] memory _prices,
    Keys.PositionKeyInfo[] memory keyInfo,
    uint256 _timestamp
)
    external
    onlyKeeper
    returns (bool[] memory isSuccess, string[] memory res)
{
    address _fastFeed = IOracle(marketRouter.oracle()).fastPriceFeed();
    IFastPriceFeed(_fastFeed).setPrices(_tokens, _prices, _timestamp);

    ...
}
```

2.In the MarketRouter contract, the admin role can set the addresses of the system contracts and various configurations of the market (e.g. order configurations, position configurations, etc.) by calling the setSysAddresses function and the setMarketConfigsByVault function. If the privilege is lost or misused, this may have an impact on the user's assets.

Code Location:

contracts/MarketRouter.sol

```
function setSysAddresses(SysAddresses memory addresses) external onlyAdmin {
    sysAddresses = addresses;
    emit UpdateSystemAddress(addresses);
}

function setMarketConfigsByVault(
    address vault,
    address[] memory indexTokens,
    MarketConfigs[] memory marketConfigs
) external onlyAdmin {
    ...
}
```

Solution

It is recommended that in the early stages of the project, the core role like the MINTER, Keeper and Admin

should use multi-signatures and the time-lock contract to avoid single-point risks. After the project is running stably, the authority of these roles should be handed over to community governance for management, and strict identity authentication should be performed when adding roles. If certain roles are run via automated scripts (e.g. keeper or Liquidator, then the permissions on that account should be tightly controlled, with private keys and helpers kept in isolation).

Status

Acknowledged; Response from the project team: After the project is officially launched, the permissions of core roles will be transferred to multi-signature address management.

[N12] [High] Lack of long/short position spread judgement

Category: Design Logic Audit

Content

In the FeeHandler contract, the getNextFundingRate function is used to calculate the latest funding charge, where the funding rate is increased if the positive or negative value of the previous funding rate is in the same direction as the current position pays the charge.

However, there is a missing check here: for example, when the last funding rate was positive and the current long position position is still larger than the short position; but the difference between long minus short has decreased from the previous time (i.e., the short position has increased more than the long position), this should be a decrease in the funding rate rather than an increase.

Code Location:

contracts/FeeHandler.sol

```
    function getNextFundingRate(
        Market.FeeConfig memory config,
        bytes32 market,
        uint256 openInterestLong,
        uint256 openInterestShort,
        uint256 duration
    ) public view returns (uint256, bool, int256) {
        ...

        bool _isSameDirection = (_fundingRate > 0 &&
            openInterestLong > openInterestShort) ||
            (_fundingRate < 0 && openInterestShort > openInterestLong);
```

```
        if (_isSameDirection) {

            // (open interest imbalance) ^ (funding exponent factor) / (total open
interest)

            if (_diffToOpenInterestFactor > config.thresholdForStableFunding) {
                _changeType = FundingRateChangeType.Increase;
            } else if (
                _diffToOpenInterestFactor < config.thresholdForDecreaseFunding
            ) {

                _changeType = FundingRateChangeType.Decrease;
            }
        } else {

            // if the skew has changed, then the funding should increase in the
opposite direction
            _changeType = FundingRateChangeType.Increase;
        }

        if (_changeType == FundingRateChangeType.Increase) {

            // increase funding rate
            // longShortImbalance * fundingIncreaseFactorPerSecond
            int256 _increaseValue = Precision
                .applyFactor(
                    _diffToOpenInterestFactor,
                    config.fundingIncreaseFactor
                )
                .toInt256() * duration.toInt256();

            // if there are more longs than shorts, then the
savedFundingFactorPerSecond should increase
            // otherwise the savedFundingFactorPerSecond should increase in the
opposite direction / decrease

            if (openInterestLong < openInterestShort) {
                _increaseValue = -_increaseValue;
            }


            _nextFundingRate = _fundingRate + _increaseValue;
        }

        ...
    }
```

Solution

It is recommended that the calculation of the latest funding rate take into account the current long/short position spread compared to the previous long/short position spread.

Status

Ignored; Response from the project team: The mechanism is consistent with the intended design.

[N13] [Suggestion] Missing event record

Category: Others

Content

The following functions in several contracts are for event logging of key parameter settings.

Code Location:

contracts/oracle/Oracle.sol#L40-107

```
    function setAdjustment

    function setFastPriceEnabled

    function setFastPriceFeed

    function setChainPriceFeed

    function setSpreadBasisPoints

    function setSpreadThresholdBasisPoints

    function setMaxStrictPriceDeviation

    function setStableTokens
```

contracts/oracle/ChainPriceFeed.sol#15-27

```
    function setSampleSpace

    function setPriceFeed
```

contracts/oracle/FastPriceFeed.sol#103-247

```
function setPriceFeed

function setMaxTimeDeviation

function setPriceDuration

function setMaxPriceUpdateDelay

function setSpreadBasisPointsIfInactive

function setSpreadBasisPointsIfChainError

function setMinBlockInterval

function setIsSpreadEnabled

function setLastUpdatedAt

function setMaxDeviationBasisPoints

function setMaxCumulativeDeltaDiffs

function setPriceDataInterval

function setTokens

function setPrices

function setCompactedPrices

function setPricesWithBits
```

contracts/Vault.sol

```
function setOrderConfig
```

Solution

It is recommended to record events when sensitive parameters are modified for self-inspection or community review.

Status

Fixed

[N14] [High] Lack of liquidity checking mechanism in case of position reduction

Category: Design Logic Audit

Content

In the PositionManager contract, the user decreases a position by calling the decreasePosition function, which calls the validateDecreaseOrder function in the orderHandler contract to check if the decrease can be done. However, there is no liquidity check in this function, so if the user takes a large profit and there is not enough liquidity in the vault to cover the user's profit amount, the decrease may fail due to lack of liquidity.

Code Location:

contracts/PositionManager.sol

```
function decreasePosition(
    Order.DecreaseOrder memory decreaseOrder
)
    external
    returns (bool, Order.CancelType, uint256, uint256, bool, uint256)
{
    ...

        (bool valid, Order.CancelType cancelType) = market
            .positionHandler
            .validatePosition(
                _position,
                collateralTokenPrice,
                markPrice,
                _rate
            );
        if (!valid) return (false, cancelType, 0, 0, false, 0);

        ...
}
```

Solution

It is recommended that liquidity checks should be made at the time of position reduction and the ADL mechanism should be added.

Status

Fixed

5 Audit Result

Audit Number	Audit Team	Audit Date	Audit Result
0X002403060001	SlowMist Security Team	2024.02.19 - 2024.03.06	Medium Risk

Summary conclusion: The SlowMist security team uses a manual and SlowMist team's analysis tool to audit the project, during the audit work we found 1 critical risk, 5 high risk, 4 medium risk, 1 low risk, 3 suggestion vulnerabilities. 1 high risk and 2 medium risk vulnerabilities were ignored; 1 medium risk and 1 suggestion vulnerabilities were Acknowledged; All other findings were fixed. The code was not deployed to the mainnet. Currently, the risk level is Medium Risk because the project is not online and the permissions of the core roles (e.g. admin and owner) have not been transferred to multi-signature address management yet.

6 Statement

SlowMist issues this report with reference to the facts that have occurred or existed before the issuance of this report, and only assumes corresponding responsibility based on these.

For the facts that occurred or existed after the issuance, SlowMist is not able to judge the security status of this project, and is not responsible for them. The security audit analysis and other contents of this report are based on the documents and materials provided to SlowMist by the information provider till the date of the insurance report (referred to as "provided information"). SlowMist assumes: The information provided is not missing, tampered with, deleted or concealed. If the information provided is missing, tampered with, deleted, concealed, or inconsistent with the actual situation, the SlowMist shall not be liable for any loss or adverse effect resulting therefrom. SlowMist only conducts the agreed security audit on the security situation of the project and issues this report. SlowMist is not responsible for the background and other conditions of the project.

Official Website

www.slowmist.com

E-mail

team@slowmist.com

Twitter

@SlowMist_Team

Github

https://github.com/slowmist