Competitive Security Assessment

KiloEx

Jun 6th, 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	KiloEx
Platform & Language	Solidity
Codebase	• https://github.com/benjamin122133/kilo-contracts • audit commit - e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb • final commit - 99704737b18f26233075ea1567914c9f5c907f21
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Code Vulnerability Review Summary

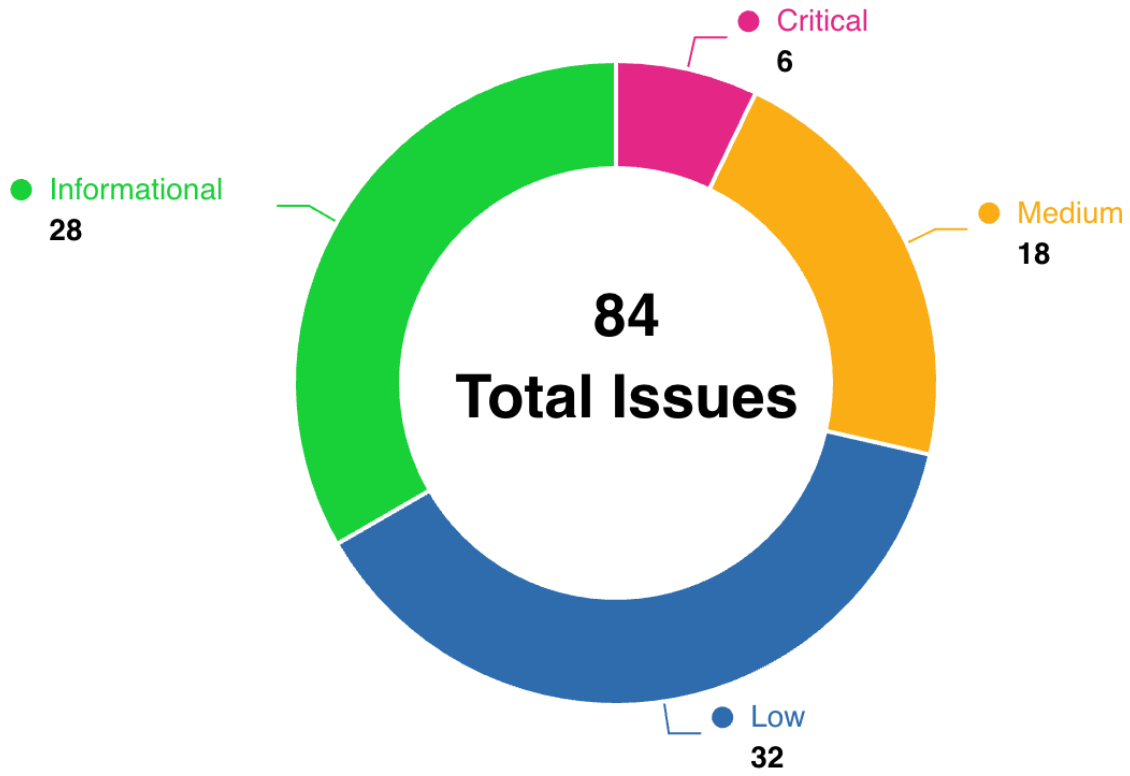
Vulnerability Level	Total	Reported	Acknowledged	Fixed	Mitigated	Declined
Critical	6	0	1	4	0	1
Medium	18	0	3	11	1	3
Low	32	0	1	25	3	3
Informational	28	0	1	23	1	3

Audit Scope

File	Commit Hash
src/core/OrderBook.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/core/PositionRouter.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/core/PerpTrade.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/core/KiloStorageManager.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/core/VaultStakeReward.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/core/KiloPriceFeed.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/core/PendingReward.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/core/ProductManager.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/peripherals/KiloPerpView.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/core/MarginFeeManager.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/libraries/PerpTradeUtil.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/tradereward/TradeRewardDistributor.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/peripherals/LiquidationPriceReader.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/referrals/ReferralStorageManager.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/interfaces/IKiloStorage.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/tradereward/ProtocolReward.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/access/OperatorOwnerGovernableUpgradeable.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/access/OperatorOwnerGovernable.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/access/OwnerGovernableUpgradeable.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/access/OwnerGovernable.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/interfaces/IPendingReward.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/interfaces/IPerpTrade.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/access/OwnerUpgradeable.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/interfaces/IProductManager.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/referrals/interfaces/IReferralStorage.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/access/GovernableUpgradeable.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb

src/access/Governable.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/referrals/ReferralReader.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/interfaces/IMarginFeeManager.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/tradereward/interfaces/ITradeRewardDistributor.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/interfaces/IOracle.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/tradereward/interfaces/IProtocolReward.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb
src/interfaces/IVaultReward.sol	e49a8ee5c8fdd0e49648f10d20c5fffca1b734cb

Code Assessment Findings



ID	Name	Category	Severity	Status	Contributor
KEX-1	A malicious early user/attacker can manipulate the vault share pricePerShare to take an unfair share of future users' deposits in `VaultStakeReward.sol`	Logical	Critical	Fixed	danielt, SerSomeone, Yaodao, ladboy233
KEX-2	Accounting issue in Orderbook, Higher leverages leads to lower margin	Logical	Critical	Fixed	ladboy233
KEX-3	Language specific in `VaultStakeReward` contract `decimals` function	Language Specific	Critical	Acknowledged	seek.guo, newway55

KEX-4	User's redeem can be DOSed by small deposit of transfer from another user to extend the stakingPeriod	Logical	Critical	Fixed	ladboy233
KEX-5	`Initializer` Function Conflict in Upgradable Smart Contracts with Inheritance	Logical	Critical	Declined	0xzoobi
KEX-6	`VaultStakeReward`: The attacker can freeze the victim's assets for stakingPeriod by transferring 1 wei share to the victim	Logical	Critical	Fixed	thereksfour
KEX-7	Execution fee is taken by keepers even when increasePosition request revert in PositionRouter.sol	Logical	Medium	Acknowledged	ladboy233
KEX-8	Incorrect state definition order in contract `PerpTrade`	Logical	Medium	Fixed	Hacker007
KEX-9	Lack of Storage Gap in Upgradeable Contracts	Logical	Medium	Fixed	Hacker007
KEX-10	Lack of check the `margin` in `OrderBook` contract `updateIncreaseOrder` function	Logical	Medium	Fixed	danielt
KEX-11	Lack of validate the timestamp from price oracle in `KiloPriceFeed` contract `getPriceAndSource` function	Oracle Manipulation	Medium	Fixed	0xzoobi, danielt, Yaodao, Hacker007, ginlee
KEX-12	OrderBook might not be able to call increasePosition/decreasePositionWithId	Privilege Related	Medium	Mitigated	SerSomeone
KEX-13	Overpaid execution fee when creating DecreaseOrder is never refunded	Logical	Medium	Fixed	ladboy233
KEX-14	Possible Denial of Service(DoS) via Frontrunning in `registerCode`	DOS	Medium	Acknowledged	0xzoobi
KEX-15	Potential DDOS attack in the functions `transfer()/transferFrom`	Logical	Medium	Fixed	Yaodao

KEX-16	Potential incorrect parameters in the function `adlDecreasePosition()`	Logical	Medium	Fixed	Yaodao
KEX-17	Potential logical flaw in `ReferralStorageManager::_setTraderReferralCode()`	Logical	Medium	Fixed	Hacker007, LiRiu
KEX-18	Rounding down to zero issue in contract "KiloPriceFeed" function "getPrice "	Precision Loss Errors	Medium	Declined	ginlee
KEX-19	User can be unfairly liquidated when addMargin is not active	Logical	Medium	Fixed	ladboy233
KEX-20	`setTraderReferralCodeByUser` function should use `tx.origin` instead of `msg.sender`.	DOS	Medium	Fixed	LiRiu
KEX-21	`tradeRewardDistributor` can potentially withdraw any amount of `xkiloToken` due to missing accounting	Privilege Related	Medium	Declined	0xzoobi
KEX-22	`var.pnl` Is Not Checked Against Total Short Size when Exiting Position Size, Potentially Resulting in Widespread Protocol Error	Logical	Medium	Acknowledged	xfu
KEX-23	access control validation did not serve its purpose in "vaultstakeaward" contract "deposit" function	Logical	Medium	Declined	ginlee
KEX-24	sanity check missing in `setMinBorrowingRate` and `setMaxBorrowingRate`	Logical	Medium	Fixed	0xzoobi
KEX-25	Division before Multiplication in `PerpTradeUtil.sol`	Logical	Low	Fixed	0xzoobi
KEX-26	Division before Multiplication in `VaultStakeReward.sol`	Code Style	Low	Fixed	Yaodao
KEX-27	Array length if not validated when calling a function can cause reverts	Code Style	Low	Fixed	0xzoobi

KEX-28	Consider using OpenZeppelin's SafeCast library to prevent unexpected overflows in `KiloStorageManager::storeIncreasePosition()`	Integer Overflow and Underflow	Low	Fixed	ginlee
KEX-29	Division before Multiplication in `MarginFeeManager.sol`、`OrderBook.sol` and `PerpTradeUtil.sol`	Logical	Low	Fixed	Hacker007
KEX-30	Follow `checks-effects-interaction` pattern	Logical	Low	Mitigated	0xzoobi
KEX-31	For upgrading contracts, add _disableInitializers() to the constructor that implements the contract	Logical	Low	Fixed	Hacker007, seek.guo
KEX-32	Incompatibility With Deflationary Tokens	Logical	Low	Declined	Yaodao, Hacker007
KEX-33	Lack of `sequencer` logic for L2 network	Logical	Low	Declined	Yaodao
KEX-34	Lack of check the `_leverage` in `OrderBook` contract `updateIncreaseOrder` function	Logical	Low	Fixed	danielt
KEX-35	Lack of logic to deal with insufficient tokens in the function `decrVaultBalance()`	Logical	Low	Mitigated	Yaodao
KEX-36	Lack of repeatability check in the function `addProduct()`	Logical	Low	Fixed	Yaodao
KEX-37	MarginFeeManager miss inheritance	Logical	Low	Fixed	seek.guo
KEX-38	Missing calls to __ReentrancyGuard_init() function in inherited contracts	Logical	Low	Fixed	Hacker007
KEX-39	Missing checks for zero address in `Governable.sol`、`GovernableUpgradeable.sol` and `OwnerUpgradeable.sol`	Logical	Low	Fixed	Yaodao
KEX-40	Missing checks for zero address in `PendingReward.sol`	Code Style	Low	Fixed	0xzoobi

KEX-41	Missing checks for zero address in `ProtocolReward.sol`	Logical	Low	Fixed	Hacker007
KEX-42	Missing input validation in `ProductManager::setLeverage()`	Logical	Low	Fixed	Hacker007
KEX-43	Ownable: Does not implement 2-Step-Process for transferring ownership	Code Style	Low	Fixed	xfu, 0xzoobi, ginlee
KEX-44	Positions cannot be executed by external user when `allowPublicKeeper` is True	Logical	Low	Fixed	xfu
KEX-45	Potential balance over cap possibility	Logical	Low	Fixed	Yaodao
KEX-46	Potential denial-of-service issues caused by accuracy conversion processes.	Logical	Low	Acknowledged	LiRiu
KEX-47	Potential divided by zero error	Logical	Low	Fixed	xfu
KEX-48	Pull-Over-Push Pattern In `setGov()` and `setOwner()` Functions	Logical	Low	Fixed	Hacker007
KEX-49	Redundant modifier `payable`	Language Specific	Low	Fixed	Hacker007
KEX-50	Unbounded loops may cause "OrderBook" contract "cancelMultiple" function to fail	DOS	Low	Fixed	ginlee
KEX-51	Uncapped Fee `minExecutionFee`	Governance Manipulation	Low	Fixed	xfu
KEX-52	Uncapped product `ProductManager.setFee()`	Governance Manipulation	Low	Fixed	xfu
KEX-53	Unchecked Return Value When Calling `IPendingReward.withdrawProtocolReward()`	Logical	Low	Declined	Hacker007
KEX-54	Wrong initial value of `minBorrowingRate`	Logical	Low	Fixed	danielt
KEX-55	initialize function can be called by anybody in KiloPriceFeed contract	Governance Manipulation	Low	Mitigated	ginlee

KEX-56	staking amount is unsafely downcasted from uint256 to uint96	Integer Overflow and Underflow	Low	Fixed	ladboy233
KEX-57	Code is never used and should be removed	Code Style	Informational	Fixed	seek.guo
KEX-58	Events not emitted or indexed for important state changes	Language Specific	Informational	Mitigated	danielt, 0xzoobi, Yaodao, ginlee
KEX-59	FEATURE SUGGESTION - Provide incentives to user's for calling `claim` on behalf of a user	Logical	Informational	Declined	0xzoobi
KEX-60	Gas Optimization: ++i costs less gas than i++, especially when it's used in for-loops (--i/i-- too)	Gas Optimization	Informational	Declined	LiRiu
KEX-61	Gas Optimization: Don’t initialize variables with default value	Gas Optimization	Informational	Fixed	ginlee
KEX-62	Gas Optimization: Double initialize `owner`	Logical	Informational	Fixed	Hacker007
KEX-63	Gas Optimization: Function `_calculatePrice` repeatedly implementation	Gas Optimization	Informational	Fixed	xfu
KEX-64	Gas Optimization: State variables set in the constructor can be declared `immutable`	Gas Optimization	Informational	Fixed	0xzoobi
KEX-65	Gas Optimization: Unused Local return Variable in `PerpTrade.sol`	Gas Optimization	Informational	Fixed	xfu
KEX-66	Gas Optimization: Unused event	Gas Optimization	Informational	Fixed	Hacker007
KEX-67	Gas Optimization: Unused statements in contract `ProductManager`	Gas Optimization	Informational	Fixed	Hacker007
KEX-68	Gas Optimization: Using calldata instead of memory for read-only arguments in external functions saves gas	Gas Optimization	Informational	Fixed	danielt, Yaodao

KEX-69	Gas Optimization: Using private rather than public for constants, saves gas	Gas Optimization	Informational	Fixed	LiRiu
KEX-70	Gas Optimization: `setReferrerDiscountShare` require conditions can be combined into a single statement	Gas Optimization	Informational	Fixed	0xzoobi
KEX-71	Gas Optimization: function `_updateOpenInterest` never used	Gas Optimization	Informational	Fixed	xfu
KEX-72	Gas Optimization: in `IKiloStorage.sol`	Gas Optimization	Informational	Fixed	0xzoobi
KEX-73	Gas Optimization: in `KiloStorageManager`::`updateIncreaseOpenInterest()`	Gas Optimization	Informational	Fixed	danielt
KEX-74	Gas Optimization: in `ReferralReader`::`getCodeOwners()`	Gas Optimization	Informational	Fixed	newway55
KEX-75	Gas Optimization: in `ReferralStorageManager.sol`	Gas Optimization	Informational	Fixed	0xzoobi
KEX-76	Gas Optimization: state mutability of `getPrices` can be restricted to `view`	Gas Optimization	Informational	Fixed	0xzoobi
KEX-77	Gas optimization: in `VaultStakeReward.sol`	Gas Optimization	Informational	Declined	xfu
KEX-78	Incomplete input validation in `KiloStorageManager`	Logical	Informational	Fixed	Hacker007
KEX-79	Incorrect commets	Code Style	Informational	Fixed	Yaodao
KEX-80	Invalid contract specified in override list: "IERC20MetadataUpgradeable".	Compiler Error	Informational	Acknowledged	xfu
KEX-81	Lack of upper and lower boundary checks in `ProductManager`::`setLeverage()`	Logical	Informational	Fixed	Yaodao
KEX-82	No need to use SafeMath in solidity version 0.8+	Gas Optimization	Informational	Fixed	Yaodao, Hacker007, ginlee
KEX-83	Unused import files	Code Style	Informational	Fixed	Hacker007

KEX-84	**`require` check without messages**	**Code Style**	**Informational**	**Fixed**	**Yaodao**

KEX-1:A malicious early user/attacker can manipulate the vault share pricePerShare to take an unfair share of future users' deposits in `VaultStakeReward.sol`

Category	Severity	Status	Contributor
Logical	Critical	Fixed	danielt, SerSomeone, Yaodao, ladboy233

Code Reference

- code/src/core/VaultStakeReward.sol#L16
- code/src/core/VaultStakeReward.sol#L64-L81
- code/src/core/VaultStakeReward.sol#L88-L104
- code/src/core/VaultStakeReward.sol#L122-L142

```
16:contract VaultStakeReward is OwnerGovernableUpgradeable, ReentrancyGuardUpgradeable, PausableUpgr
adeable, ERC20Upgradeable, ERC4626Upgradeable {

64:     function initialize(
65:         address _kiloConfigAddr,
66:         string memory _name,
67:         string memory _symbol
68:     ) public initializer {
69:         __owner_governable_init();
70:         __ReentrancyGuard_init();
71:         __Pausable_init();
72:
73:         __ERC20_init(_name, _symbol);
74:
75:         kiloConfig = IKiloStorage(_kiloConfigAddr);
76:         perpTrade = kiloConfig.perpTradeAddr();
77:         pendingReward = kiloConfig.pendingRewardAddr();
78:         token = kiloConfig.token();
79:         tokenBase = kiloConfig.tokenBase();
80:         __ERC4626_init(IERC20MetadataUpgradeable(token));
81:     }

88:     function deposit(uint256 amount, address user) public override nonReentrant returns (uint256)
{
89:         require(kiloConfig.canUserStake(), "Vault: not allowed");
90:         require(msg.sender == user, "Vault: not staker");
91:         require(IERC20Upgradeable(token).balanceOf(address(this)) * BASE / tokenBase + amount <=
cap, "over cap");
92:         _claimVaultPendingReward();
93:
94:         Stake storage _stake = stakes[user];
95:         uint256 newAmount = amount * tokenBase / BASE;
96:         _stake.owner = user;
97:         _stake.timestamp = uint128(block.timestamp);
98:         _stake.amount += uint96(newAmount);
99:         staked += uint256(newAmount);
100:         uint256 shares = previewDeposit(newAmount);
101:         _deposit(user, user, newAmount, shares);
102:         emit StakedRedeemItem(user, StakeType.Stake, int256(newAmount), int256(shares), _stake.a
mount, IERC20Upgradeable(address(this)).balanceOf(user), 0);
103:         return shares * BASE / tokenBase;
104:     }
```

```
122:     function redeem(uint256 shares, address receiver, address owner) public override nonReentran
t returns (uint256) {
123:         shares = shares * tokenBase / BASE;
124:         require(shares <= totalSupply(), "Vault: cannot redeem");
125:         address user = msg.sender;
126:         uint256 userShare = IERC20Upgradeable(address(this)).balanceOf(user);
127:         if (shares >= userShare) {
128:             shares = userShare;
129:         }
130:         _claimVaultPendingReward();
131:         Stake storage _stake = stakes[user];
132:         require(block.timestamp - _stake.timestamp > stakingPeriod, "Vault: not in period");
133:         uint256 assets = previewRedeem(shares);
134:         require(kiloConfig.totalOpenInterest() <= (IERC20Upgradeable(token).balanceOf(address(th
is)) - assets) * BASE / tokenBase * kiloConfig.getKiloConfig().utilizationMultiplier / (10 ** 4), "!
utilized");
135:         uint256 amount = shares * _stake.amount / userShare;
136:         _stake.amount -= uint96(amount);
137:         staked -= uint256(amount);
138:         _withdraw(_msgSender(), receiver, owner, assets, shares);
139:
140:         emit StakedRedeemItem(user, StakeType.Redeem, -1*int256(amount), -1*int256(shares), _sta
ke.amount, userShare-shares, assets);
141:         return assets * BASE / tokenBase;
142:     }
```

Description

danielt : It is a well-known attack vector for almost all shares-based liquidity pool contracts, where a malicious early user can manipulate the price per share and profit from late users' deposits because of the precision loss caused by the rather large value of price per share.

A malicious early user can `deposit()` with 1wei of asset token as the first depositor of the `token`, and get 1wei of shares. Then the attacker can send 10000e18-1 of asset tokens and inflate the price per share from 1.0000 to an extreme value of 1.0000e22 (from (1+10000e18-1)/1) .

```
    //VaultStakeReward.sol
    function deposit(uint256 amount, address user) public override nonReentrant returns (uint256) {
        ...
        uint256 shares = previewDeposit(newAmount);
        ...
    }

    //ERC4626Upgradeable.sol
    function previewDeposit(uint256 assets) public view virtual override returns (uint256) {
        return _convertToShares(assets, MathUpgradeable.Rounding.Down);
    }

    function _convertToShares(uint256 assets, MathUpgradeable.Rounding rounding) internal view virtu
al returns (uint256) {
        uint256 supply = totalSupply();
        return
            (assets == 0 || supply == 0)
                ? _initialConvertToShares(assets, rounding)
                : assets.mulDiv(supply, totalAssets(), rounding);
    }
```

As a result, the future user who deposits 19999e18 will only receive 1wei (from 19999 e18*1/10000e18) of the shares token. They will immediately lose 9999e18 or half of their deposits if they redeem() right after the `deposit()`.

The attacker can profit from future users' deposits. While the late users will lose part of their funds to the attacker.

SerSomeone : The `VaultStakeReward` vault adopts the ERC4646 vault structure. ERC4646 has a known vulnerability that is exploitable if the vault is not funded when created/initialized. The vulnerability causes share inflation. To exploit the vulnerability a hacker becomes the first depositor and then donates funds to the vault (which increases share size).

The issue exists because the exchange rate is calculated as the ratio between the `totalSupply` of shares and the `totalAssets()` (the vaults balance of the underlying asset).

Since there is no funding in the `initialize` function, the vulnerability is exploitable.

```
    function initialize(
        address _kiloConfigAddr,
        string memory _name,
        string memory _symbol
    ) public initializer {
        __owner_governable_init();
        __ReentrancyGuard_init();
        __Pausable_init();

        __ERC20_init(_name, _symbol);

        kiloConfig = IKiloStorage(_kiloConfigAddr);
        perpTrade = kiloConfig.perpTradeAddr();
        pendingReward = kiloConfig.pendingRewardAddr();
        token = kiloConfig.token();
        tokenBase = kiloConfig.tokenBase();
        __ERC4626_init(IERC20MetadataUpgradeable(token));
    }
```

Additional explanation of the vulnerability and mitigations can be found here:

- https://docs.openzeppelin.com/contracts/4.x/erc4626#inflation-attack
- https://www.youtube.com/watch?v=_pO2jDgL0XE&t=601s
  Yaodao : In solidity, there is a truncation problem in the division calculation. A well konwn attack vector for almost all shares based liquidity pool contracts, where an early user can manipulate the price per share and profit from late users' deposits because of the precision loss caused by the rather large value of price per share.

The function `previewDeposit()` used in the contract `VaultStakeReward` is from openzeppelin's contract `ERC4626Upgradeable`. In the following codes, the `shares` to mint calls the function `previewDeposit()` to calculate.

```
    function deposit(uint256 amount, address user) public override nonReentrant returns (uint256) {
        require(kiloConfig.canUserStake(), "Vault: not allowed");
        require(msg.sender == user, "Vault: not staker");
        require(IERC20Upgradeable(token).balanceOf(address(this)) * BASE / tokenBase + amount <= ca
p, "over cap");
        _claimVaultPendingReward();

        Stake storage _stake = stakes[user];
        uint256 newAmount = amount * tokenBase / BASE;
        _stake.owner = user;
        _stake.timestamp = uint128(block.timestamp);
        _stake.amount += uint96(newAmount);
        staked += uint256(newAmount);
        uint256 shares = previewDeposit(newAmount);
        _deposit(user, user, newAmount, shares);
        emit StakedRedeemItem(user, StakeType.Stake, int256(newAmount), int256(shares), _stake.amoun
t, IERC20Upgradeable(address(this)).balanceOf(user), 0);
        return shares * BASE / tokenBase;
    }
```

Consider below functions in the `ERC4626Upgradeable`, the malicious early user can `deposit()` with 1 wei of asset token as the first depositor, and get 1wei of shares.

```
    /** @dev See {IERC4626-previewDeposit}. */
    function previewDeposit(uint256 assets) public view virtual override returns (uint256) {
        return _convertToShares(assets, MathUpgradeable.Rounding.Down);
    }

    /**
     * @dev Internal conversion function (from assets to shares) with support for rounding directio
n.
     */
    function _convertToShares(uint256 assets, MathUpgradeable.Rounding rounding) internal view virtu
al returns (uint256) {
        return assets.mulDiv(totalSupply() + 10 ** _decimalsOffset(), totalAssets() + 1, rounding);
    }
```

For the malicious early user, the shares will be `1 * ( 0 + 1) / ( 0 + 1 ) = 1`. And the attacker can transfer asset tokens into the contract to influence the other users' calls of `deposit()`.

For example, a user wants to deposit 1e18 wei asset tokens. The attacker can `front-run` transfer 2e18 wei asset tokens , and the shares of the user will be `1e18 * ( 1 + 1 ) / ( 2e18 + 1 + 1 ) = 0`. As a result, the attacker can make the shares of all users to 0 by `front-run` transferring asset tokens and all the users' asset tokens will loss.

To further more, consider below functions in the `ERC4626Upgradeable`. When the attacker wants to get asset tokens back, the attacker can deposit a large amount of asset tokens and the redeem all shares.(For more effective and safe to attack with sufficient funds, the attacker can perform this operation through flash loans.)

```
    /** @dev See {IERC4626-previewRedeem}. */
    function previewRedeem(uint256 shares) public view virtual override returns (uint256) {
        return _convertToAssets(shares, MathUpgradeable.Rounding.Down);
    }

    /**
     * @dev Internal conversion function (from shares to assets) with support for rounding directio
n.
     */
    function _convertToAssets(uint256 shares, MathUpgradeable.Rounding rounding) internal view virtu
al returns (uint256) {
        return shares.mulDiv(totalAssets() + 1, totalSupply() + 10 ** _decimalsOffset(), rounding);
    }
```

For example, refer to the above example, the attacker can deposit 10e18 wei asset tokens and will get `10e18 * ( 1 + 1 ) / ( 3e18 + 1 + 1 ) = 6` shares. As a result, the total shares of the attacker is 7 wei, and the amount to get back is `7 * ( 13e18 + 1 + 1 ) / ( 7 + 1 )`.

As a result, the attacker can `deposit()` with 1 wei of asset token as the first depositor and then make the shares of all users to 0 by `front-run` transferring asset tokens. At last, the attacker can depost a large amount of asset tokens to get `n` shares, and the redeem the `n+1` shares. All the asset tokens the attacker can get is `(n+1)/(n+2)` of the attacker transferred and the other users deposit.

ladboy233 : A malicious early user/attacker can manipulate the vault share pricePerShare to take an unfair share of future users' deposits in VaultStakeReward.sol because VaultStakeReward.sol inherit from ERC4626Upgradeable

A well known attack vector for almost all shares based liquidity pool contracts, where an early user can manipulate the price per share and profit from late users' deposits because of the precision loss caused by the rather large value of price per share.

A malicious early user can `deposit()` with `1 wei` of `asset` token as the first depositor of the vault stake reward share, and get `1 wei` of shares.

Then the attacker can send `10000e18 - 1` of `asset` tokens and inflate the price per share from 1.0000 to an extreme value of 1.0000e22 (from `(1 + 10000e18 - 1) / 1`).

As a result, the future user who deposits `19999e18` will only receive `1 wei` (from `19999e18 * 1 / 10000e18`) of shares token.

They will immediately lose `9999e18` or half of their deposits if they `redeem()` right after the `deposit()`.

Recommendation

danielt : Consider requiring a minimal amount of share tokens to be minted for the first minter, and send a port of the initial mints as a reserve to the DAO so that the price per share can be more resistant to manipulation.
SerSomeone : either as part of the `initialize` function fund the vault to create a large enough supply or create virtual offset as recommended in https://docs.openzeppelin.com/contracts/4.x/erc4626#inflation-attack
Yaodao : Recommend requiring a minimal amount of share tokens to be minted for the first minter, and send a port of the initial mints as a reserve to the DAO so that the share can be more resistant to manipulation.
ladboy233 : Consider requiring a minimal amount of share tokens to be minted for the first minter, and send a port of the initial mints as a reserve to the DAO so that the pricePerShare can be more resistant to manipulation.

Client Response

Fixed

KEX-2:Accounting issue in Orderbook, Higher leverages leads to lower margin

Category	Severity	Status	Contributor
Logical	Critical	Fixed	ladboy233

Code Reference

- code/src/core/OrderBook.sol#L379-L403

```
379:     function updateIncreaseOrder(
380:         uint256 _orderIndex,
381:         uint256 _leverage,
382:         uint256 _triggerPrice,
383:         bool _triggerAboveThreshold
384:     ) external nonReentrant {
385:         IncreaseOrder storage order = increaseOrders[msg.sender][_orderIndex];
386:         require(order.account != address(0), "OrderBook: non-existent order");
387:         if (order.leverage != _leverage) {
388:             uint256 margin = (order.margin + order.tradeFee) * BASE / (BASE + getTradeFeeRate(or
der.productId, order.account) * _leverage / 10**4);
389:             uint256 tradeFee = order.tradeFee + order.margin - margin;
390:             order.margin = margin;
391:             order.tradeFee = tradeFee;
392:             order.leverage = _leverage;
393:         }
394:         order.triggerPrice = _triggerPrice;
395:         order.triggerAboveThreshold = _triggerAboveThreshold;
396:
397:         emit UpdateIncreaseOrder(
398:             msg.sender,
399:             _orderIndex,
400:             order.productId,
401:             order.margin,
402:             order.leverage,
403:             order.tradeFee,
```

Description

ladboy233 : Loss of margin and trading fee if user adjust leverage by calling updateIncreaseOrder in Orderbook.sol

In the current implementation, when create a increase order, the leverage applies to trader fee and user needs to pay the margin + trader fee to create an increase order

```
    function createIncreaseOrder(
        uint256 _productId,
        uint256 _margin,
        uint256 _leverage,
        bool _isLong,
        uint256 _triggerPrice,
        bool _triggerAboveThreshold,
        uint256 _executionFee,
        bytes32 _referralCode
    ) external payable nonReentrant {
        require(_executionFee >= minExecutionFee, "OrderBook: insufficient execution fee");
        require(_margin >= kiloStorage.getKiloConfig().minMargin, "OrderBook: insufficient collatera
l");

        uint256 tradeFee = getTradeFeeRate(_productId, msg.sender) * _margin * _leverage / (FEE_BASE
* BASE);
        require(msg.value == _executionFee, "OrderBook: incorrect execution fee transferred");

        if (_referralCode != bytes32(0) && referralStorage != address(0)) {
            IReferralStorage(referralStorage).setTraderReferralCodeByUser(_referralCode);
        }

        IERC20Upgradeable(token).safeTransferFrom(msg.sender, address(this), (_margin + tradeFee) *
tokenBase / BASE);
```

however, if user adjust levearge, the order.margin and order.trader fee is also updated

```
    function updateIncreaseOrder(
        uint256 _orderIndex,
        uint256 _leverage,
        uint256 _triggerPrice,
        bool _triggerAboveThreshold
    ) external nonReentrant {
        IncreaseOrder storage order = increaseOrders[msg.sender][_orderIndex];
        require(order.account != address(0), "OrderBook: non-existent order");
        if (order.leverage != _leverage) {
            uint256 margin = (order.margin + order.tradeFee) * BASE / (BASE + getTradeFeeRate(order.
productId, order.account) * _leverage / 10**4);
            uint256 tradeFee = order.tradeFee + order.margin - margin;
            order.margin = margin;
            order.tradeFee = tradeFee;
            order.leverage = _leverage;
        }
        order.triggerPrice = _triggerPrice;
        order.triggerAboveThre
```

If user want to adjust their leverage, their initally paid trader fee and margin can be lost,

let us just say the wants to update the leverage to 0

```
uint256 margin = (order.margin + order.tradeFee) * BASE / (BASE + getTradeFeeRate(order.productId, o
rder.account) * _leverage / 10**4);
        uint256 tradeFee = order.tradeFee + order.margin - margin;
        order.margin = margin;
        order.tradeFee = tradeFee;
        order.leverage = _leverage;
```

the margin would be equal

```
(order.margin + order.traderFee) * BASE / BASE

and traderFee = order.tradeFee + order.margin - margin
```

is

```
traderFee = order.tradeFee + order.margin - (order.tradeFee + order.margin) = 0
```

In this case, the traderFee is 0 when updateIncreaseOrder, but the paid trader fee is never refunded

Same issue happens for the margin variable

```
uint256 margin = (order.margin + order.tradeFee) * BASE / (BASE + getTradeFeeRate(order.productId, o
rder.account) * _leverage / 10**4);
```

if user set the leverage very very very high, the margin can be truncated down to 0

if both case, the code failed to transfer the overpaid traderFee and margin after the leverage is updated

and when the increaseOrder is canceled

```
    function cancelIncreaseOrder(uint256 _orderIndex) public nonReentrant {
        IncreaseOrder memory order = increaseOrders[msg.sender][_orderIndex];
        require(order.account != address(0), "OrderBook: non-existent order");

        delete increaseOrders[msg.sender][_orderIndex];
        accountOrderSize[msg.sender] = accountOrderSize[msg.sender] - 1;

        // @audit order.margin can be updated with updateIncreaseOrder!!!!
        IERC20Upgradeable(token).safeTransfer(msg.sender, (order.margin + order.tradeFee) * tokenBas
e / BASE);
        _transferOutETH(order.executionFee, payable(msg.sender));
        emit CancelIncreaseOrder(
            order.account,
            _orderIndex,
            order.productId,
            order.margin,
            order.leverage,
            order.tradeFee,
            order.isLong,
            order.triggerPrice,
            order.triggerAboveThreshold,
            order.executionFee
        );
    }
```

the transferred amount when cancel the increase order is based on order.margin and order.traderFee, which already decrease after the leverage adjustment when calling updateIncreaseOrder

ladboy233 : Loss of margin and trading fee if user adjust leverage by calling updateIncreaseOrder in Orderbook.sol

In the current implementation, when create a increase order, the leverage applies to trader fee and user needs to pay the margin + trader fee to create an increase order

```
    function createIncreaseOrder(
        uint256 _productId,
        uint256 _margin,
        uint256 _leverage,
        bool _isLong,
        uint256 _triggerPrice,
        bool _triggerAboveThreshold,
        uint256 _executionFee,
        bytes32 _referralCode
    ) external payable nonReentrant {
        require(_executionFee >= minExecutionFee, "OrderBook: insufficient execution fee");
        require(_margin >= kiloStorage.getKiloConfig().minMargin, "OrderBook: insufficient collatera
l");

        uint256 tradeFee = getTradeFeeRate(_productId, msg.sender) * _margin * _leverage / (FEE_BASE
* BASE);
        require(msg.value == _executionFee, "OrderBook: incorrect execution fee transferred");

        if (_referralCode != bytes32(0) && referralStorage != address(0)) {
            IReferralStorage(referralStorage).setTraderReferralCodeByUser(_referralCode);
        }

        IERC20Upgradeable(token).safeTransferFrom(msg.sender, address(this), (_margin + tradeFee) *
tokenBase / BASE);
```

when updateIncreaseOrder, user can update the leverage

```
    function updateIncreaseOrder(
        uint256 _orderIndex,
        uint256 _leverage,
        uint256 _triggerPrice,
        bool _triggerAboveThreshold
    ) external nonReentrant {
        IncreaseOrder storage order = increaseOrders[msg.sender][_orderIndex];
        require(order.account != address(0), "OrderBook: non-existent order");
        if (order.leverage != _leverage) {
            uint256 margin = (order.margin + order.tradeFee) * BASE / (BASE + getTradeFeeRate(order.
productId, order.account) * _leverage / 10**4);
            uint256 tradeFee = order.tradeFee + order.margin - margin;
            order.margin = margin;
            order.tradeFee = tradeFee;
            order.leverage = _leverage;
        }
        order.triggerPrice = _triggerPrice;
        order.triggerAboveThreshold = _triggerAboveThreshold;
```

note the line of code:

```
uint256 margin = (order.margin + order.tradeFee) * BASE / (BASE + getTradeFeeRate(order.productId, o
rder.account) * _leverage / 10**4);
```

the code is implemented in a way that the higher the leverage is, the lower the margin is,

I find this one strange because the higher the leverage, the higher the margin and the code never charge additional margin when the levearge increase.

Recommendation

ladboy233 : The recommendation is that protocol should refund the updated margin and trading fee when user update increase order leveage

ladboy233 : When createIncreaseOrder, the levearge applies to traderFee and force user to pay a higher collateral when the leverage is high

The recommendation is that the protocol should make user pay higher margin if the leverage increase.

Client Response

Fixed

KEX-3:Language specific in `VaultStakeReward` contract `decimals` function

Category	Severity	Status	Contributor
Language Specific	Critical	Acknowledged	seek.guo, newway55

Code Reference

- code/src/core/VaultStakeReward.sol#L234-L236

```
234:    function decimals() public view override(ERC20Upgradeable, IERC20MetadataUpgradeable) returns (uint8) {
235:        return IERC20MetadataUpgradeable(token).decimals();
236:    }
```

Description

seek.guo : Compiler run failed error[4327]: TypeError: Function needs to specify overridden contract "ERC4626Upgradeable". --> src/core/VaultStakeReward.sol:234:37: | 234 | function decimals() public view override(ERC20Upgradeable, IERC20MetadataUpgradeable) returns (uint8) { | ^^^ Note: This contract: --> lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/extensions/ERC4626Upgradeable.sol:31:1: | 31 | abstract contract ERC4626Upgradeable is Initializable, ERC20Upgradeable, IERC4626Upgradeable { | ^ (Relevant source part starts here and spans across multiple lines).

error[2353]: TypeError: Invalid contract specified in override list: "IERC20MetadataUpgradeable". --> src/core/VaultStakeReward.sol:234:37: | 234 | function decimals() public view override(ERC20Upgradeable, IERC20MetadataUpgradeable) returns (uint8) { | ^^^ Note: This contract: --> lib/openzeppelin-contracts-upgradeable/contracts/token/ERC20/extensions/IERC20MetadataUpgradeable.sol:13:1: | 13 | interface IERC20MetadataUpgradeable is IERC20Upgradeable { | ^ (Relevant source part starts here and spans across multiple lines).

newway55 : Wrong overriding in the decimals function in the VaultStakeReward contract. The contract may not function as expected if other contracts rely on the overridden behavior.

Consider below POC contract

```
 TypeError: Function needs to specify overridden contract "ERC4626Upgradeable".
   --> src/core/VaultStakeReward.sol:234:37:
    |
234 |     function decimals() public view override(ERC20Upgradeable, IERC20MetadataUpgradeable) retu
rns (uint8) {
```

Recommendation

seek.guo : Here's how to fix it

```
function decimals() public view override(ERC4626Upgradeable, ERC20Upgradeable) returns (uint8) {
        return IERC20MetadataUpgradeable(token).decimals();
}
```

newway55 : Change the overriding parameters.

Consider below fix in the `sample.test()` function

```
   function decimals() public view override(ERC4626Upgradeable, ERC20Upgradeable) returns (uint8) {
        return IERC20MetadataUpgradeable(token).decimals();
    }
```

Client Response

Acknowledged, Can not be fix, same as kex-80,Modified as reported, compilation is wrong

KEX-4:User's redeem can be DOSed by small deposit of transfer from another user to extend the stakingPeriod

Category	Severity	Status	Contributor
Logical	Critical	Fixed	ladboy233

Code Reference

- code/src/core/VaultStakeReward.sol#L132
- code/src/core/VaultStakeReward.sol#L155-L177

```
132:         require(block.timestamp - _stake.timestamp > stakingPeriod, "Vault: not in period");

155:
156:     function transfer(address to, uint256 amount) public override(IERC20Upgradeable, ERC20Upgrad
eable) returns (bool) {
157:         address owner = _msgSender();
158:         Stake storage _fromStakeItem = stakes[owner];
159:         Stake storage _toStakeItem = stakes[to];
160:
161:         _claimVaultPendingReward();
162:         uint256 userShare = IERC20Upgradeable(address(this)).balanceOf(owner);
163:         require(userShare > 0, "Vault: no assets");
164:         uint256 decreasedShares = amount;
165:         uint256 changedAmounts = decreasedShares * _fromStakeItem.amount / userShare;
166:
167:         require(_fromStakeItem.amount >= changedAmounts, "Vault: transfer amount exceeds balanc
e");
168:         _fromStakeItem.amount -= uint96(changedAmounts);
169:         _toStakeItem.amount += uint96(changedAmounts);
170:         _toStakeItem.owner = to;
171:         _toStakeItem.timestamp = uint128(Math.max(_fromStakeItem.timestamp, _toStakeItem.timesta
mp));
172:         _transfer(owner, to, amount);
173:
174:         emit StakedRedeemItem(owner, StakeType.TransferOut, -1*int256(changedAmounts), -1*int256
(decreasedShares), _fromStakeItem.amount, userShare-decreasedShares, 0);
175:         emit StakedRedeemItem(to, StakeType.TransferIn, int256(changedAmounts), int256(decreased
Shares), _toStakeItem.amount, IERC20Upgradeable(address(this)).balanceOf(to), 0);
176:
177:         return true;
```

Description

ladboy233 : User's redeem can be DOSed by small deposit of another user to extend the stakingPeriod

In Deposit function in VaultStakeReward.sol

```
    function deposit(uint256 amount, address user) public override nonReentrant returns (uint256) {
        require(kiloConfig.canUserStake(), "Vault: not allowed");
        require(msg.sender == user, "Vault: not staker");
        require(IERC20Upgradeable(token).balanceOf(address(this)) * BASE / tokenBase + amount <= ca
p, "over cap");
        _claimVaultPendingReward();

        Stake storage _stake = stakes[user];
        uint256 newAmount = amount * tokenBase / BASE;
        _stake.owner = user;
        _stake.timestamp = uint128(block.timestamp);
        // @audit safe downcasting
        _stake.amount += uint96(newAmount);
        staked += uint256(newAmount);
        // @audit inflation attack
        // @audit 0 shares
        uint256 shares = previewDeposit(newAmount);
        _deposit(user, user, newAmount, shares);
        emit StakedRedeemItem(user, StakeType.Stake, int256(newAmount), int256(shares), _stake.amoun
t, IERC20Upgradeable(address(this)).balanceOf(user), 0);
        // @audit what are you returning?
        return shares * BASE / tokenBase;
    }
```

and when redeem, we are calling

```
    function redeem(uint256 shares, address receiver, address owner) public override nonReentrant re
turns (uint256) {
        // @audit how did you validate owner?
        shares = shares * tokenBase / BASE;
        require(shares <= totalSupply(), "Vault: cannot redeem");
        address user = msg.sender;
        uint256 userShare = IERC20Upgradeable(address(this)).balanceOf(user);
        if (shares >= userShare) {
            shares = userShare;
        }
        _claimVaultPendingReward();
        Stake storage _stake = stakes[user];

        // @audit DOS by transfer hahaahha
        require(block.timestamp - _stake.timestamp > stakingPeriod, "Vault: not in period");
```

note that the redeem is subject to stakingPeriod so user cannot redeem / withdraw too frequently

the issue is that, user A can modify user B's staking timestamp by transfer a small amount of token

```
    // @audit strange transfer logic
    function transfer(address to, uint256 amount) public override(IERC20Upgradeable, ERC20Upgradeabl
e) returns (bool) {
        address owner = _msgSender();
        Stake storage _fromStakeItem = stakes[owner];
        Stake storage _toStakeItem = stakes[to];

        _claimVaultPendingReward();
        uint256 userShare = IERC20Upgradeable(address(this)).balanceOf(owner);
        require(userShare > 0, "Vault: no assets");
        uint256 decreasedShares = amount;
        uint256 changedAmounts = decreasedShares * _fromStakeItem.amount / userShare;

        require(_fromStakeItem.amount >= changedAmounts, "Vault: transfer amount exceeds balance");
        _fromStakeItem.amount -= uint96(changedAmounts);
        _toStakeItem.amount += uint96(changedAmounts);
        _toStakeItem.owner = to;
        _toStakeItem.timestamp = uint128(Math.max(_fromStakeItem.timestamp, _toStakeItem.timestam
p));
        _transfer(owner, to, amount);

        emit StakedRedeemItem(owner, StakeType.TransferOut, -1*int256(changedAmounts), -1*int256(dec
reasedShares), _fromStakeItem.amount, userShare-decreasedShares, 0);
        emit StakedRedeemItem(to, StakeType.TransferIn, int256(changedAmounts), int256(decreasedShar
es), _toStakeItem.amount, IERC20Upgradeable(address(this)).balanceOf(to), 0);

        return true;
    }
```

note the logic

```
_fromStakeItem.amount -= uint96(changedAmounts);
_toStakeItem.amount += uint96(changedAmounts);
_toStakeItem.owner = to;
_toStakeItem.timestamp = uint128(Math.max(_fromStakeItem.timestamp, _toStakeItem.timestamp));
```

when user transfer token to address(to), address(to)'s timestamp is updated and extended, then a malicious user can keep transfer small amount of token to lock user B's asset, wen user B try to redeem, his redeem transaction will always revert beacuse user B is subject to maliciously extended staking cooldown period

Recommendation

ladboy233 : Either remove the cooldown staking period, or not let other user modify the cooldown period for other user.

Client Response

Fixed, prohibit users from transferring shares during the frozen period.

KEX-5: `Initializer` Function Conflict in Upgradable Smart Contracts with Inheritance

Category	Severity	Status	Contributor
Logical	Critical	Declined	0xzoobi

Code Reference

- code/src/tradereward/ProtocolReward.sol#L24-L27
- code/src/referrals/ReferralStorageManager.sol#L38-L39
- code/src/core/MarginFeeManager.sol#L42-L46
- code/src/core/PendingReward.sol#L44-L47
- code/src/core/ProductManager.sol#L45-L50
- code/src/core/KiloPriceFeed.sol#L48-L49
- code/src/tradereward/TradeRewardDistributor.sol#L49-L52
- code/src/core/VaultStakeReward.sol#L64-L69
- code/src/core/KiloStorageManager.sol#L85-L88
- code/src/core/PerpTrade.sol#L99-L103
- code/src/core/OrderBook.sol#L162-L167
- code/src/core/PositionRouter.sol#L178-L183

```
24:     function initialize(
25:         address _rewardToken, address _pendingReward, address _protocolRewardReceiver
26:     ) public initializer {
27:         __owner_governable_init();

38:     function initialize() public initializer {
39:         __owner_governable_init();

42:     function initialize(
43:         address _productManager,
44:         address _kiloStorageAddr
45:     ) public initializer {
46:         __owner_governable_init();

44:     function initialize (
45:         address _kiloStorageAddr
46:     ) public initializer {
47:         __owner_governable_init();

45:     function initialize(
46:
47:     ) public initializer {
48:         discountEnabled = false;
49:         __owner_governable_init();
50:     }

48:     function initialize() public initializer {
49:         __owner_governable_init();

49:     function initialize(
50:         address _rewardToken
51:     ) public initializer {
52:         __owner_governable_init();

64:     function initialize(
65:         address _kiloConfigAddr,
66:         string memory _name,
67:         string memory _symbol
68:     ) public initializer {
69:         __owner_governable_init();

85:     function initialize(
```

```
86:
87:     ) public initializer {
88:         __owner_governable_init();

99:     function initialize(
100:         address _kiloStorageAddr
101:     ) public initializer {
102:         __ReentrancyGuard_init();
103:         __owner_governable_init();

162:     function initialize(
163:         address _collateralToken,
164:         uint256 _minExecutionFee,
165:         address _kiloStorageAddr
166:     ) public initializer {
167:         __owner_governable_init();

178:     function initialize(
179:         address _collateralToken,
180:         uint256 _minExecutionFee,
181:         address _kiloStorageAddr
182:     ) public initializer {
183:         __owner_governable_init();
```

Description

0xzoobi : The `initializer` modifier which is part of the `contracts-upgradeable/proxy/utils/Initializable.sol`, is used in case of upgradable contracts, instead of a constructor to be called only once. Suppose there is a `initialize` function with `initializer` modifier, then it can be called only once, Calling it the second time will cause a revert with error like `"Initializable: contract is already initialized"`.

Lets understand the issue in our case step by step.

1. `ProductManager.sol` has a `initialize` function that looks like the below code

```
    function initialize(

    ) public initializer {
        discountEnabled = false;
        __owner_governable_init();
    }
```

2. The `initialize` function is protected by `initializer`, so that it can be called only once.
3. The `initialize` function in turn calls the `__owner_governable_init`. The code for which is shown below.

```
    function __owner_governable_init() internal initializer {
        gov = msg.sender;
        owner = msg.sender;
    }
```

4. As you can the `__owner_governable_init` is also protected by `initializer` function.
5. Since the contract can be initialized only once, the first initialize call made via `ProductManager.sol` already locks the contract. As a result, `__owner_governable_init` will revert with error `"Initializable: contract is already initialized".`

The impact of the issue is that, none of the Upgradable Smart Contracts can be initialized.

Sample PoC: Run via Remix IDE

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;

import "https://github.com/OpenZeppelin/openzeppelin-contracts-upgradeable/blob/master/contracts/access/OwnableUpgradeable.sol";

contract OwnerGovernableUpgradeable is Initializable {

    address public gov;
    address public owner;

    uint256[50] private __gap;

    event SetGov(address gov);
    event SetOwner(address owner);

    function __owner_governable_init() internal initializer {
        gov = msg.sender;
        owner = msg.sender;
    }

    modifier onlyGov() {
        require(msg.sender == gov, "OwnerGovernable: forbidden");
        _;
    }

    modifier onlyOwner() {
        require(msg.sender == owner, "OwnerGovernable: forbidden");
        _;
    }
}
-------------------------------------------------------------------------------------------------------------
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;

import "./OwnerGovernableUpgradeable.sol";

contract ProductManager is OwnerGovernableUpgradeable {

    bool public discountEnabled = false;

    function initialize(
```

```
) public initializer {
        discountEnabled = true;
        __owner_governable_init();
    }
}
```

Recommendation

0xzoobi : yet to find a fix

Client Response

Declined, In OpenZeppelin 4.7 version, it is possible to call the initializer function multiple times during contract deployment. This is because the initializer function is no longer marked as initializer after it has been called once, so subsequent calls to the function will not trigger the initializer logic.

KEX-6: `VaultStakeReward` : The attacker can freeze the victim's assets for stakingPeriod by transferring 1 wei share to the victim

Category	Severity	Status	Contributor
Logical	Critical	Fixed	thereksfour

Code Reference

- code/src/core/VaultStakeReward.sol#L155-L207

```
155:
156:     function transfer(address to, uint256 amount) public override(IERC20Upgradeable, ERC20Upgrad
eable) returns (bool) {
157:         address owner = _msgSender();
158:         Stake storage _fromStakeItem = stakes[owner];
159:         Stake storage _toStakeItem = stakes[to];
160:
161:         _claimVaultPendingReward();
162:         uint256 userShare = IERC20Upgradeable(address(this)).balanceOf(owner);
163:         require(userShare > 0, "Vault: no assets");
164:         uint256 decreasedShares = amount;
165:         uint256 changedAmounts = decreasedShares * _fromStakeItem.amount / userShare;
166:
167:         require(_fromStakeItem.amount >= changedAmounts, "Vault: transfer amount exceeds balanc
e");
168:         _fromStakeItem.amount -= uint96(changedAmounts);
169:         _toStakeItem.amount += uint96(changedAmounts);
170:         _toStakeItem.owner = to;
171:         _toStakeItem.timestamp = uint128(Math.max(_fromStakeItem.timestamp, _toStakeItem.timesta
mp));
172:         _transfer(owner, to, amount);
173:
174:         emit StakedRedeemItem(owner, StakeType.TransferOut, -1*int256(changedAmounts), -1*int256
(decreasedShares), _fromStakeItem.amount, userShare-decreasedShares, 0);
175:         emit StakedRedeemItem(to, StakeType.TransferIn, int256(changedAmounts), int256(decreased
Shares), _toStakeItem.amount, IERC20Upgradeable(address(this)).balanceOf(to), 0);
176:
177:         return true;
178:     }
179:
180:     function transferFrom(
181:         address from,
182:         address to,
183:         uint256 amount
184:     ) public override(IERC20Upgradeable, ERC20Upgradeable) returns (bool) {
185:
186:         Stake storage _fromStakeItem = stakes[from];
187:         Stake storage _toStakeItem = stakes[to];
188:
189:         _claimVaultPendingReward();
190:         uint256 userShare = IERC20Upgradeable(address(this)).balanceOf(from);
```

```
191:         require(userShare > 0, "no assets");
192:         uint256 decreasedShares = amount;
193:         uint256 changedAmounts = decreasedShares * _fromStakeItem.amount / userShare;
194:         require(_fromStakeItem.amount >= changedAmounts, "Vault: transfer amount exceeds balanc
e");
195:         _fromStakeItem.amount -= uint96(changedAmounts);
196:         _toStakeItem.amount += uint96(changedAmounts);
197:         _toStakeItem.owner = to;
198:         _toStakeItem.timestamp = uint128(Math.max(_fromStakeItem.timestamp, _toStakeItem.timesta
mp));
199:
200:         _spendAllowance(from, msg.sender, amount);
201:         _transfer(from, to, amount);
202:
203:         emit StakedRedeemItem(from, StakeType.TransferOut, -1*int256(changedAmounts), -1*int256
(decreasedShares), _fromStakeItem.amount, userShare-decreasedShares, 0);
204:         emit StakedRedeemItem(to, StakeType.TransferIn, int256(changedAmounts), int256(decreased
Shares), _toStakeItem.amount, IERC20Upgradeable(address(this)).balanceOf(to), 0);
205:
206:         return true;
207:     }
```

Description

thereksfour : In VaultStakeReward, the user can redeem the deposited assets only after at least stakingPeriod, which will be 0 to 30 days.

```
    function deposit(uint256 amount, address user) public override nonReentrant returns (uint256) {
        ...
        Stake storage _stake = stakes[user];
        uint256 newAmount = amount * tokenBase / BASE;
        _stake.owner = user;
        _stake.timestamp = uint128(block.timestamp);
...
    function redeem(uint256 shares, address receiver, address owner) public override nonReentrant re
turns (uint256) {
        ...
        Stake storage _stake = stakes[user];
        require(block.timestamp - _stake.timestamp > stakingPeriod, "Vault: not in period");
...
    function updateVaultCapAndPeriod(uint256 _cap, uint256 _stakingPeriod) external onlyOwner {
        require(_cap > 0 && _stakingPeriod > 0 && _stakingPeriod < 30 days, "Vault: not allowed");
        cap = _cap;
        stakingPeriod = _stakingPeriod;
```

However, in transfer/transferFrom, frozen assets can be transferred and a new timestamp will be set for the recipient

```
    function transfer(address to, uint256 amount) public override(IERC20Upgradeable, ERC20Upgradeabl
e) returns (bool) {
        address owner = _msgSender();
        Stake storage _fromStakeItem = stakes[owner];
        Stake storage _toStakeItem = stakes[to];
        ...
        _toStakeItem.timestamp = uint128(Math.max(_fromStakeItem.timestamp, _toStakeItem.timestam
p));
```

This allows the attacker to transfer 1 wei share to the victim to freeze the victim's assets for stakingPeriod.

Consider stakingPeriod = 10 days

On day 0, Alice deposits 1000 ETH, and Alice can redeem the assets on day 10. On day 9, Bob deposits 0.0001 ETH and transfers 1 wei share to alice, at which point Alice's redemption time is extended to day 19.

Bob can freeze Alice's assets at little cost and can blackmail Alice.

Recommendation

thereksfour : Consider adding the following check to transfer/transferFrom to prohibit users from transferring shares during the frozen period

```
require(block.timestamp - _fromStakeItem.timestamp > stakingPeriod, "Vault: not in period");
```

Client Response

Fixed, same as kex-4

KEX-7:Execution fee is taken by keepers even when increasePosition request revert in PositionRouter.sol

Category	Severity	Status	Contributor
Logical	Medium	Acknowledged	ladboy233

Code Reference

- code/src/core/PositionRouter.sol#L290-L324

```
290:     function executeIncreasePositions(uint256 _endIndex, address payable _executionFeeReceiver)
internal {
291:         uint256 index = increasePositionRequestKeysStart;
292:         uint256 length = increasePositionRequestKeys.length;
293:
294:         if (index >= length) { return; }
295:
296:         if (_endIndex > length) {
297:             _endIndex = length;
298:         }
299:
300:         while (index < _endIndex) {
301:             bytes32 key = increasePositionRequestKeys[index];
302:
303:             // if the request was executed then delete the key from the array
304:             // if the request was not executed then break from the loop, this can happen if the
305:             // minimum number of blocks has not yet passed
306:             // an error could be thrown if the request is too old or if the slippage is
307:             // higher than what the user specified, or if there is insufficient liquidity for th
e position
308:             // in case an error was thrown, cancel the request
309:             try this.executeIncreasePosition(key, _executionFeeReceiver) returns (bool _wasExecu
ted) {
310:                 if (!_wasExecuted) { break; }
311:             } catch Error(string memory executionError) {
312:                 emit ExecuteIncreasePositionError(increasePositionRequests[key].account, index,
executionError);
313:                 // wrap this call in a try catch to prevent invalid cancels from blocking the lo
op
314:                 try this.cancelIncreasePosition(key, _executionFeeReceiver) returns (bool _wasCa
ncelled) {
315:                     if (!_wasCancelled) { break; }
316:                 } catch {}
317:             } catch (bytes memory /*lowLevelData*/) {
318:                 // wrap this call in a try catch to prevent invalid cancels from blocking the lo
op
319:                 try this.cancelIncreasePosition(key, _executionFeeReceiver) returns (bool _wasCa
ncelled) {
320:                     if (!_wasCancelled) { break; }
321:                 } catch {}
```

```
322:             }
323:
324:             delete increasePositionRequestKeys[index];
```

Description

ladboy233 : Execution fee is taken by keepers even when increasePosition request revert in PositionRouter.sol

In PositionRouter.sol, keeper can execution the position

```
    function executePositionsWithPrices(
        address[] memory tokens,
        uint256[] memory prices,
        uint256 _openEndIndex,
        uint256 _closeEndIndex,
        address payable _executionFeeReceiver
    ) public onlyPositionKeeper {
        IOracle(oracle).setPrices(tokens, prices);
        executeIncreasePositions(_openEndIndex, _executionFeeReceiver);
        executeDecreasePositions(_closeEndIndex, _executionFeeReceiver);
    }
```

this is calling

```
    function executeIncreasePositions(uint256 _endIndex, address payable _executionFeeReceiver) inte
rnal {
        uint256 index = increasePositionRequestKeysStart;
        uint256 length = increasePositionRequestKeys.length;

        if (index >= length) { return; }

        if (_endIndex > length) {
            _endIndex = length;
        }

        while (index < _endIndex) {
            bytes32 key = increasePositionRequestKeys[index];

            // if the request was executed then delete the key from the array
            // if the request was not executed then break from the loop, this can happen if the
            // minimum number of blocks has not yet passed
            // an error could be thrown if the request is too old or if the slippage is
            // higher than what the user specified, or if there is insufficient liquidity for the po
sition
            // in case an error was thrown, cancel the request
            try this.executeIncreasePosition(key, _executionFeeReceiver) returns (bool _wasExecuted)
{
                if (!_wasExecuted) { break; }
            } catch Error(string memory executionError) {
                emit ExecuteIncreasePositionError(increasePositionRequests[key].account, index, exec
utionError);
                // wrap this call in a try catch to prevent invalid cancels from blocking the loop
                try this.cancelIncreasePosition(key, _executionFeeReceiver) returns (bool _wasCancel
led) {
                    if (!_wasCancelled) { break; }
                } catch {}
            }
```

note the code section:

```
try this.executeIncreasePosition(key, _executionFeeReceiver) returns (bool _wasExecuted) {
        if (!_wasExecuted) { break; }
} catch Error(string memory executionError) {
        emit ExecuteIncreasePositionError(increasePositionRequests[key].account, index, executionErr
or);
        // wrap this call in a try catch to prevent invalid cancels from blocking the loop
        try this.cancelIncreasePosition(key, _executionFeeReceiver) returns (bool _wasCancelled) {
                if (!_wasCancelled) { break; }
        } catch {}
}
```

the code try to execute the increasePosition logic on PerpTrader.sol smart contract and if there is a error message, the code try to cancel the increasePosition request, yet still taking the exectuion fee

However, if we take a look at the increasePosition code in PerpTrader.sol

```
    function increasePosition(
        address user,
        uint256 productId,
        uint256 margin,
        bool isLong,
        uint256 leverage,
        uint256 orderType
    ) public payable nonReentrant {
        _validateRouter();
        IKiloStorage.KiloConfig memory kiloConfig = kiloStorage.getKiloConfig();
        // @audit when disabled, execute order lose execution fee
        require(kiloConfig.isTradeEnabled, "not enable");
        require(margin >= kiloConfig.minMargin && margin < type(uint64).max, "margin error");
        IProductManager.Product memory product  = IProductManager(productManager).getProduct(product
Id);
        require(product.isActive, "not active");
        require(leverage <= uint256(product.maxLeverage), "leverage too big");
        require(leverage >= product.minLeverage, "leverage too low");
```

it is possible when keeper execute the position change request, kiloConfig.isTradeEnabled is set to false

it is possible that the product.isActive is set to false, it is possible the leverage set by user is too large or too low, larger than configuration setting of product.maxLeverage or lower than product.minLeverage request,

then the increasePosition revert and the increasePosition request is canceled by keeper, yet the execution fee is never refunded to user it is taken by keeper

such mechanism because very unfair when the product is disabled or the trade is disabled

Recommendation

ladboy233 : Refund the execution fee in if the increasePosition or decreasePosition revert due to project confiugration

Client Response

Acknowledged, Will be optimized in the future

KEX-8:Incorrect state definition order in contract `PerpTrade`

Category	Severity	Status	Contributor
Logical	Medium	Fixed	Hacker007

Code Reference

- code/src/core/PerpTrade.sol#L48-L50

```
48:     uint256[50] private __gap;
49:
50:     uint256 private constant ADL_ORDER_TYPE = 2;
```

Description

Hacker007 : The storage gap is used to allow developers to freely add new state variables in the future without compromising the storage compatibility with existing deployments.

```
contract PerpTrade is IPerpTrade, OwnerGovernableUpgradeable, ReentrancyGuardUpgradeable {
    //..
    uint256[50] private __gap;

    uint256 private constant ADL_ORDER_TYPE = 2;
    //...
}
```

However, if a new state `ADL_ORDER_TYPE` is defined after the storage gap `__gap`, `ADL_ORDER_TYPE` still can be overridden in the derived contract, which may lead to a corruptible Upgradability Pattern.

Recommendation

Hacker007 : Add storage gap at the end of contract `PerpTrade`.

```
contract PerpTrade is IPerpTrade, OwnerGovernableUpgradeable, ReentrancyGuardUpgradeable {
    //..
    uint256 private constant ADL_ORDER_TYPE = 2;
    //events
    //functions
    uint256[50] private __gap;
}
```

Client Response

Fixed

KEX-9:Lack of Storage Gap in Upgradeable Contracts

Category	Severity	Status	Contributor
Logical	Medium	Fixed	Hacker007

Code Reference

- code/src/core/ProductManager.sol#L7
- code/src/referrals/ReferralStorageManager.sol#L7
- code/src/tradereward/ProtocolReward.sol#L11
- code/src/core/KiloStorageManager.sol#L13-L14
- code/src/tradereward/TradeRewardDistributor.sol#L13
- code/src/core/PendingReward.sol#L14

```
7:contract ProductManager is IProductManager, OwnerGovernableUpgradeable {

7:contract ReferralStorageManager is IReferralStorage, OperatorOwnerGovernableUpgradeable {

11:contract ProtocolReward is OwnerGovernableUpgradeable, PausableUpgradeable, ReentrancyGuardUpgrad
eable {

13:contract KiloStorageManager is IKiloStorage, OwnerGovernableUpgradeable, ReentrancyGuardUpgradeab
le {
14:     using SafeERC20Upgradeable for IERC20Upgradeable;

13:contract TradeRewardDistributor is ITradeRewardDistributor, OperatorOwnerGovernableUpgradeable, P
ausableUpgradeable, ReentrancyGuardUpgradeable {

14:contract PendingReward  is IPendingReward, OwnerGovernableUpgradeable,ReentrancyGuardUpgradeable
{
```

Description

Hacker007 : For upgradeable contracts, there must be a storage gap to "allow developers to freely add new state variables in the future without compromising the storage compatibility with existing deployments". Otherwise, it may be very difficult to write new implementation code. See https://docs.openzeppelin.com/contracts/4.x/upgradeable#storage_gaps

Recommendation

Hacker007 : Add a storage gap at the end of upgradeable contracts. An example:

```
contract KiloPriceFeed is OwnerGovernableUpgradeable, IOracle {
//...
    modifier onlyKeeper() {
        require(keepers[msg.sender], "!keepers");
        _;
    }
    //solhint-disable-next-line ordering
    uint256[50] private __gap;
}
```

Client Response

Fixed

KEX-10:Lack of check the `margin` in `OrderBook` contract `updateIncreaseOrder` function

Category	Severity	Status	Contributor
Logical	Medium	Fixed	danielt

Code Reference

- code/src/core/OrderBook.sol#L379-L388

```
379:     function updateIncreaseOrder(
380:         uint256 _orderIndex,
381:         uint256 _leverage,
382:         uint256 _triggerPrice,
383:         bool _triggerAboveThreshold
384:     ) external nonReentrant {
385:         IncreaseOrder storage order = increaseOrders[msg.sender][_orderIndex];
386:         require(order.account != address(0), "OrderBook: non-existent order");
387:         if (order.leverage != _leverage) {
388:             uint256 margin = (order.margin + order.tradeFee) * BASE / (BASE + getTradeFeeRate(or
der.productId, order.account) * _leverage / 10**4);
```

Description

danielt : There is a check on the `_margin` to keep it from being less than the lower boundary in the `OrderBook` contract `createIncreaseOrder` function:

```
    function createIncreaseOrder(
        uint256 _productId,
        uint256 _margin,
        uint256 _leverage,
        bool _isLong,
        uint256 _triggerPrice,
        bool _triggerAboveThreshold,
        uint256 _executionFee,
        bytes32 _referralCode
    ) external payable nonReentrant {
        require(_executionFee >= minExecutionFee, "OrderBook: insufficient execution fee");
        require(_margin >= kiloStorage.getKiloConfig().minMargin, "OrderBook: insufficient collatera
l");
        ...
```

However, there is no corresponding check on `_margin` in the `updateIncreaseOrder` function:

```
    function updateIncreaseOrder(
        uint256 _orderIndex,
        uint256 _leverage,
        uint256 _triggerPrice,
        bool _triggerAboveThreshold
    ) external nonReentrant {
        IncreaseOrder storage order = increaseOrders[msg.sender][_orderIndex];
        require(order.account != address(0), "OrderBook: non-existent order");
        if (order.leverage != _leverage) {//audit: check the below formula
            uint256 margin = (order.margin + order.tradeFee) * BASE / (BASE + getTradeFeeRate(order.
productId, order.account) * _leverage / 10**4);
            uint256 tradeFee = order.tradeFee + order.margin - margin;
            order.margin = margin;
            ...
    }
```

The `margin` is calculated from the formula `(order.margin + order.tradeFee) * BASE / (BASE + getTrad eFeeRate(order.productId, order.account) * _leverage / 10**4);` which is may less the `minMargin`.

Recommendation

danielt : Adding the check on `margin` in the `updateIncreaseOrder` to ensure it is not less than the lower boundary.

Client Response

Fixed

KEX-11:Lack of validate the timestamp from price oracle in `KiloPriceFeed` contract `getPriceAndSource` function

Category	Severity	Status	Contributor
Oracle Manipulation	Medium	Fixed	0xzoobi, danielt, Yaodao, Hacker007, ginlee

Code Reference

- code/src/core/KiloPriceFeed.sol#L101-L120
- code/src/core/KiloPriceFeed.sol#L102-L110
- code/src/core/KiloPriceFeed.sol#L122-L136
- code/src/core/KiloPriceFeed.sol#L125-L128

```
101:    function getPriceAndSource(address token) public view returns (uint256, bool) {
102:        (uint256 chainlinkPrice, uint256 chainlinkTimestamp) = getChainlinkPrice(token);
103:        if (isChainlinkOnly ||
104:                (!isKiloOracleOnly
105:                    && (block.timestamp > lastUpdatedTimes[token].add(priceDuration)
106:                    && chainlinkTimestamp > lastUpdatedTimes[token])
107:                )
108:            ) {
109:            return (chainlinkPrice, true);
110:        }
111:
112:        uint256 kiloPrice = priceMap[token];
113:        uint256 priceDiff = kiloPrice > chainlinkPrice ? (kiloPrice.sub(chainlinkPrice)).mul(1e1
8).div(chainlinkPrice) :
114:            (chainlinkPrice.sub(kiloPrice)).mul(1e18).div(chainlinkPrice);
115:        uint256 maxPriceDiff = maxPriceDiffs[token] == 0 ? defaultMaxPriceDiff : maxPriceDiffs[t
oken];
116:        if (priceDiff > maxPriceDiff) {
117:            return (chainlinkPrice, true);
118:        }
119:        return (kiloPrice, false);
120:    }
```

```
102:        (uint256 chainlinkPrice, uint256 chainlinkTimestamp) = getChainlinkPrice(token);
103:        if (isChainlinkOnly ||
104:                (!isKiloOracleOnly
105:                    && (block.timestamp > lastUpdatedTimes[token].add(priceDuration)
106:                    && chainlinkTimestamp > lastUpdatedTimes[token])
107:                )
108:            ) {
109:            return (chainlinkPrice, true);
110:        }
```

```
122:    function getChainlinkPrice(address token) public view returns (uint256 priceToReturn, uint25
6 chainlinkTimestamp) {
123:        require(token != address(0), '!feed-error');
124:
125:        (,int256 price,,uint256 timeStamp,) = AggregatorV3Interface(token).latestRoundData();
126:
```

```
127:         require(price > 0, '!price');
128:         require(timeStamp > 0, '!timeStamp');
129:         uint8 decimals = AggregatorV3Interface(token).decimals();
130:         chainlinkTimestamp = timeStamp;
131:         if (decimals != 8) {
132:             priceToReturn = uint256(price) * (10**8) / (10**uint256(decimals));
133:         } else {
134:             priceToReturn = uint256(price);
135:         }
136:     }

125:         (,int256 price,,uint256 timeStamp,) = AggregatorV3Interface(token).latestRoundData();
126:
127:         require(price > 0, '!price');
128:         require(timeStamp > 0, '!timeStamp');
```

Description

0xzoobi : The `KiloPriceFeed.sol` function `getChainlinkPrice` uses Chainlink's `latestRoundData()` to get the latest price. However, there is no check if the return value indicates stale data.

However, there is a check for stateless present in the `getPriceAndSource`, which basically calls the `getChainlinkPrice` and performs these checks later on.

`getChainlinkPrice` does not natively has these checks. As a result, any function calling these function may be using stale prices

For Example: The `getPrices` function from the `LiquidationPriceReader.sol` The impact of this would be a user could be liquidated because of state price feed.

danielt : When the `isChainlinkOnly` is true, the `KiloPriceFeed` contract's `getPriceAndSource` function does not validate for update time, `chainlinkTimestamp`, returned from the `getChainlinkPrice` function.

```
    function getPriceAndSource(address token) public view returns (uint256, bool) {
        (uint256 chainlinkPrice, uint256 chainlinkTimestamp) = getChainlinkPrice(token);
        if (isChainlinkOnly ||//audit: edge case if isChainlinkOnly, the price may be stale
                (!isKiloOracleOnly
                    && (block.timestamp > lastUpdatedTimes[token].add(priceDuration) //600秒更新一次
                    && chainlinkTimestamp > lastUpdatedTimes[token])
                )
            ) {
            return (chainlinkPrice, true);
        }
        ...
    }
```

So, we would not know if the price returned by latestRoundData() exceeded the timeout if `isChainlinkOnly` is true. It probably results in an unexpected result.

The price is used in the `OrderBook` contract's `validatePositionOrderPrice` function:

```
    function validatePositionOrderPrice(
        bool _triggerAboveThreshold,
        uint256 _triggerPrice,
        uint256 _productId,
        bool _isLong
    ) public view returns (uint256 currentPrice) {
        IProductManager.Product memory product = IProductManager(productManager).getProduct(_product
Id);
        currentPrice = _isLong ? IOracle(oracle).getPrice(product.productToken, true) : IOracle(orac
le).getPrice(product.productToken, false);
        bool isPriceValid = _triggerAboveThreshold ? currentPrice >= _triggerPrice : currentPrice <=
_triggerPrice;
        require(isPriceValid, "OrderBook: invalid price for execution");
    }
```

A stale price a day ago from Chainlink will of course incur side effects on the business of the `OrderBook` contract.

For example:

- productToken is ETH;
- ETH price today is actually 1820, however, a stale price a day ago from Chainlink is 1750 since lack of validating the chainlinkTimestamp;
- _triggerPrice is 1800, however, when `_triggerAboveThreshold` is true, the `validatePositionOrderPrice` function still returns `false` since `1750 >= 1800` is false.

  Yaodao : The function `getPriceAndSource()` is used to get the price from `chainlink` or `kilo`. It will use `cha-`

`inlink` prices first if `isChainlinkOnly` is true, or `isKiloOracleOnly` is false and `kilo` prices expires and `chainlink` prices is newer than `kilo` prices.

In other conditions, it will compare whether `kilo` prices is within the allowed difference of `chainlink` prices and then choose which price to return.

Consider below codes

```
    function getPriceAndSource(address token) public view returns (uint256, bool) {
        (uint256 chainlinkPrice, uint256 chainlinkTimestamp) = getChainlinkPrice(token);
        if (isChainlinkOnly ||
                (!isKiloOracleOnly
                    && (block.timestamp > lastUpdatedTimes[token].add(priceDuration)
                    && chainlinkTimestamp > lastUpdatedTimes[token])
                )
            ) {
            return (chainlinkPrice, true);
        }

        uint256 kiloPrice = priceMap[token];
        uint256 priceDiff = kiloPrice > chainlinkPrice ? (kiloPrice.sub(chainlinkPrice)).mul(1e18).d
iv(chainlinkPrice) :
            (chainlinkPrice.sub(kiloPrice)).mul(1e18).div(chainlinkPrice);
        uint256 maxPriceDiff = maxPriceDiffs[token] == 0 ? defaultMaxPriceDiff : maxPriceDiffs[toke
n];
        if (priceDiff > maxPriceDiff) {
            return (chainlinkPrice, true);
        }
        return (kiloPrice, false);
    }
```

However, there is no logic to handle whether the `chainlink` price has expired or not been updated for a long time. In some specific cases, `chainlink` will stop updating the price of a certain token, for example, the update of `Luna` price is stopped in the Luna event.

In this condition, the incorrect price will return to the protocol logic. For example, the chainlink price of `tokenA` is 0.1 ETH and has not been updated for a long time, and the real price given by `kilo` is 0.001 ETH. However, the `priceDiff` will be 99% and should over the `maxPriceDiff`. As a result, the chainlink price will be return which is the incorrect price.

Hacker007 : The function `getPriceAndSource()` has a few validations to avoid incorrect prices. If such validations succeed, the function returns the non-zero oracle price. For the Chainlink oracle, `getPriceAndSource()` ultimately calls `getChainlinkPrice()` with the following validation.

```
    function getChainlinkPrice(address token) public view returns (uint256 priceToReturn, uint256 ch
ainlinkTimestamp) {
        require(token != address(0), '!feed-error');

        (,int256 price,,uint256 timeStamp,) = AggregatorV3Interface(token).latestRoundData();

        require(price > 0, '!price');
        require(timeStamp > 0, '!timeStamp');
```

`timeStamp` refers to the timestamp of the round. This value isn't checked to make sure it is recent. Additionally, regarding price, it is crucial to be aware of the minAnswer and maxAnswer of the Chainlink oracle, these values are not allowed to be reached or surpassed. See Chainlink API reference for documentation on minAnswer and maxAnswer as well as this piece of code : OffchainAggregator.sol

ginlee : (,int256 price,,uint256 timeStamp,) = AggregatorV3Interface(token).latestRoundData(); timeStamp represents updatedAt value, but it only checks timeStamp is greater than 0, it does not check for price feed staleness. It has happened before - a feed stops updating the price and returns a stale one

Recommendation

0xzoobi : Add a check to see when the price was last updated and revert if the data is older than a certain time period For documentation and risks related to the issue. Please refer

1. https://docs.chain.link/data-feeds/historical-data
2. https://0xmacro.com/blog/how-to-consume-chainlink-price-feeds-safely/

   danielt : Validating the update time return from the `getChainlinkPrice` function if `isChainlinkOnly` is true to ensure that the price is on time.

   Yaodao : Recommend adding logic for `isKiloOracleOnly` to use `kilo` prices when chainlink prices has expired or not been updated for a long time.

   Hacker007 : • Determine the tolerance threshold for updateAt. If block.timestamp - updateAt exceeds that threshold, revert the function call, which is consistent with how the current validations are handled. • Consider having off-chain monitoring to identify when the market price moves out of [minAnswer, maxAn- swer] range.

   ginlee : validate that no more than 1 hour(or any value you want to set) has passed from the updatedAt timestamp value returned from latestRoundData, otherwise the transaction will revert. require (timeStamp >= block.timestamp - 3600, "stale price")

Client Response

Fixed, add stale price check, and If the Chainlink price feed consistently fails, the Kilo secondary oracle will be activated.

KEX-12:OrderBook might not be able to call increasePosition/decreasePositionWithId

Category	Severity	Status	Contributor
Privilege Related	Medium	Mitigated	SerSomeone

Code Reference

code/src/core/PerpTrade.sol#L388

```
388:    function _validateRouter() private view {
```

Description

SerSomeone : Both `decreasePositionWithId` and `increasePosition` call the `validateRouter` function to make sure the `PositionRouter` is calling these function

```
    function _validateRouter() private view {
        require(approvedRouters[msg.sender], "not allowed");
    }
```

Since `OrderBook` also calls these function, it is important that it also gets added to the whitelist.

Recommendation

SerSomeone : Add another whitelist for approved `OrderBook`s

Client Response

Mitigated, The deployment script has been synchronized to complete the initialization of the whitelist

KEX-13:Overpaid execution fee when creating DecreaseOrder is never refunded

Category	Severity	Status	Contributor
Logical	Medium	Fixed	ladboy233

Code Reference

- code/src/core/OrderBook.sol#L480

```
480:         _createDecreaseOrder(
```

Description

ladboy233 : In OrderBook.sol, when creating order, the execution fee needs to be paid

When creating IncreaseOrder, the code validates that the exact execution fee is paid

```
    function createIncreaseOrder(
        uint256 _productId,
        uint256 _margin,
        uint256 _leverage,
        bool _isLong,
        uint256 _triggerPrice,
        bool _triggerAboveThreshold,
        uint256 _executionFee,
        bytes32 _referralCode
    ) external payable nonReentrant {
        require(_executionFee >= minExecutionFee, "OrderBook: insufficient execution fee");
        require(_margin >= kiloStorage.getKiloConfig().minMargin, "OrderBook: insufficient collatera
l");

        uint256 tradeFee = getTradeFeeRate(_productId, msg.sender) * _margin * _leverage / (FEE_BASE
* BASE);
        require(msg.value == _executionFee, "OrderBook: incorrect execution fee transferred");
```

note the check

```
require(msg.value == _executionFee, "OrderBook: incorrect execution fee transferred");
```

however, when creating decreaseOrder, the code never refund the overpaid execution fee and never validate the paid execution fee is the exact amount and allows fee to be overpaid

```
    function createDecreaseOrder(
        uint256 _productId,
        uint256 _size,
        bool _isLong,
        uint256 _triggerPrice,
        bool _triggerAboveThreshold
    ) external payable nonReentrant {

        // @audit overpaid execution fee not refunded
        // @audit missing check on order size
        require(msg.value >= minExecutionFee, "OrderBook: insufficient execution fee");

        _createDecreaseOrder(
            msg.sender,
            _productId,
            _size,
            _isLong,
            _triggerPrice,
            _triggerAboveThreshold
        );
    }
```

Recommendation

ladboy233 : The recommendation is validating the exact amount of execution fee is paid or refund the overpaid fee at last after creating the decrease order

Client Response

Fixed

KEX-14:Possible Denial of Service(DoS) via Frontrunning in `registerCode`

Category	Severity	Status	Contributor
DOS	Medium	Acknowledged	0xzoobi

Code Reference

- code/src/referrals/ReferralStorageManager.sol#L93-L98

```
93:     function registerCode(bytes32 _code) external override {
94:         require(_code != bytes32(0), "ReferralStorage: invalid _code");
95:         require(codeOwners[_code] == address(0), "ReferralStorage: _code already exists");
96:         codeOwners[_code] = msg.sender;
97:         emit RegisterCode(msg.sender, _code);
98:     }
```

Description

0xzoobi : The expected behaviour of `registerCode` is that a user sets a referral code via this function and share this code to other users so that they can use the referral code and register via `setTraderReferralCodeByUser`.

Imagine the following attacker scenario. A malicious attacker Bob is monitoring the transactions in the mempool and Bob has a large list of disposable accounts.

The Attack

1. Alice - a regular user tries calls the `registerCode` function with `9ce7c58ade18f87506c9a01de3955799dd3ea212550bab68075d2e0fc8891fff`. Just for context Alice is well known trader and has 1000s of followers waiting to join. So, there is a very good possibility that Alice can bring in a large number of traders.
2. Bob who is monitoring the mempool, looks at this transaction and calls the `registerCode` function with the same code as Alice - `9ce7c58ade18f87506c9a01de3955799dd3ea212550bab68075d2e0fc8891fff` and front runs the transaction by paying higher gas fees.
3. Alice's transaction reverts with `"ReferralStorage: _code already exists"`
4. Alice calls the `registerCode` function once again with a new code `d0560f2f5d5102dbb6f8e61ed4579f4e8947fd7d7f5076b5e8e5726c9fe2b36e`
5. Bob once again front runs the transaction with a new account and Alice's transaction reverts again.

Since Bob has several disposable accounts, Bob can keep doing this forever.

The impact of this is Denial of Service to honest users calling the `registerCode` function.

Recommendation

0xzoobi : A better approach would be to generate the code using the address of the user and some random salt, when user calls `registerCode` function instead of allowing the user to set the code. This will eliminate the possibility of front running attacks.

Consider below fix

```
    uint256 constant SALT = 17413123123132; //random salt

     function registerCode() external {
         bytes32 code = keccak256(abi.encodePacked(msg.sender, SALT));
         //it is almost unlikely for the generated code to be reverted with `"ReferralStorage: _code
already exists"` but just in case if it happens, the owner can call `govSetCodeOwner` in such an ins
tance manually
         require(codeOwners[code] == address(0), "ReferralStorage: _code already exists");
         codeOwners[code] = msg.sender;
         emit RegisterCode(msg.sender, code);
     }
```

Client Response

Acknowledged, Will be optimized in the future

KEX-15:Potential DDOS attack in the functions `transfer()/transferFrom`

Category	Severity	Status	Contributor
Logical	Medium	Fixed	Yaodao

Code Reference

- code/src/core/VaultStakeReward.sol#L122-L142
- code/src/core/VaultStakeReward.sol#L171
- code/src/core/VaultStakeReward.sol#L198

```
122:    function redeem(uint256 shares, address receiver, address owner) public override nonReentran
t returns (uint256) {
123:        shares = shares * tokenBase / BASE;
124:        require(shares <= totalSupply(), "Vault: cannot redeem");
125:        address user = msg.sender;
126:        uint256 userShare = IERC20Upgradeable(address(this)).balanceOf(user);
127:        if (shares >= userShare) {
128:            shares = userShare;
129:        }
130:        _claimVaultPendingReward();
131:        Stake storage _stake = stakes[user];
132:        require(block.timestamp - _stake.timestamp > stakingPeriod, "Vault: not in period");
133:        uint256 assets = previewRedeem(shares);
134:        require(kiloConfig.totalOpenInterest() <= (IERC20Upgradeable(token).balanceOf(address(th
is)) - assets) * BASE / tokenBase * kiloConfig.getKiloConfig().utilizationMultiplier / (10 ** 4), "!
utilized");
135:        uint256 amount = shares * _stake.amount / userShare;
136:        _stake.amount -= uint96(amount);
137:        staked -= uint256(amount);
138:        _withdraw(_msgSender(), receiver, owner, assets, shares);
139:
140:        emit StakedRedeemItem(user, StakeType.Redeem, -1*int256(amount), -1*int256(shares), _sta
ke.amount, userShare-shares, assets);
141:        return assets * BASE / tokenBase;
142:    }

171:        _toStakeItem.timestamp = uint128(Math.max(_fromStakeItem.timestamp, _toStakeItem.timesta
mp));

198:        _toStakeItem.timestamp = uint128(Math.max(_fromStakeItem.timestamp, _toStakeItem.timesta
mp));
```

Description

Yaodao : According to the following codes, the `timestamp` of the `_toStakeItem` will be updated in the functions `transfer()` and `transferFrom()`. The `_fromStakeItem.timestamp` will be updated for the new deposit.

```
_toStakeItem.timestamp = uint128(Math.max(_fromStakeItem.timestamp, _toStakeItem.timestamp));
```

```
_toStakeItem.timestamp = uint128(Math.max(_fromStakeItem.timestamp, _toStakeItem.timestamp));
```

In the `redeem()` function, the call of `redeem()` will fail if the `block.timestamp - _stake.timestamp` is not over the `stakingPeriod`.

```
    function redeem(uint256 shares, address receiver, address owner) public override nonReentrant re
turns (uint256) {
        shares = shares * tokenBase / BASE;
        require(shares <= totalSupply(), "Vault: cannot redeem");
        address user = msg.sender;
        uint256 userShare = IERC20Upgradeable(address(this)).balanceOf(user);
        if (shares >= userShare) {
            shares = userShare;
        }
        _claimVaultPendingReward();
        Stake storage _stake = stakes[user];
        require(block.timestamp - _stake.timestamp > stakingPeriod, "Vault: not in period");
        uint256 assets = previewRedeem(shares);
        require(kiloConfig.totalOpenInterest() <= (IERC20Upgradeable(token).balanceOf(address(this))
- assets) * BASE / tokenBase * kiloConfig.getKiloConfig().utilizationMultiplier / (10 ** 4), "!utili
zed");
        uint256 amount = shares * _stake.amount / userShare;
        _stake.amount -= uint96(amount);
        staked -= uint256(amount);
        _withdraw(_msgSender(), receiver, owner, assets, shares);

        emit StakedRedeemItem(user, StakeType.Redeem, -1*int256(amount), -1*int256(shares), _stake.a
mount, userShare-shares, assets);
        return assets * BASE / tokenBase;
    }
```

As a result, the attacker can periodically deposit small amounts of shares(like 1 wei) and transfer them to other users so that these users can never redeem their deposit.

Recommendation

Yaodao : Recommend setting a reasonable minimum transfer amount to increase the cost of DDOS attacks.

Client Response

Fixed, same as kex-4

KEX-16:Potential incorrect parameters in the function `adlDecreasePosition()`

Category	Severity	Status	Contributor
Logical	Medium	Fixed	Yaodao

Code Reference

- code/src/core/PerpTrade.sol#L196-L215

```
196:    function adlDecreasePosition(
197:        uint256 positionId,
198:        address productToken,
199:        address[] memory tokens,
200:        uint256[] memory prices
201:    ) external nonReentrant {
202:        require(liquidators[msg.sender], "not adl operator");
203:        IOracle(oracle).setPrices(tokens, prices);
204:        IKiloStorage.Position memory position = kiloStorage.getPositionById(positionId);
205:        IKiloStorage.KiloConfig memory kiloConfig = kiloStorage.getKiloConfig();
206:        uint256 price = IOracle(oracle).getPrice(productToken);
207:
208:        IMarginFeeManager(marginFeeManager).updateMarginFee(position.productId);
209:        (int256 fundingPayment, uint256 borrowingFee) = PerpTradeUtil._getMarginFee(marginFeeMan
ager, position.productId, position.isLong, position.leverage, position.margin, position.funding, pos
ition.borrowing);
210:
211:        int256 pnl = PerpTradeUtil._getPnl(position.isLong, position.price, position.leverage, p
osition.margin, price) - fundingPayment - int256(borrowingFee);
212:        if (pnl >= 0 && uint256(pnl) > position.margin * kiloConfig.adlMultiplier) {
213:            return decreasePositionWithId(positionId, position.margin, ADL_ORDER_TYPE, price);
214:        }
215:    }
```

Description

Yaodao : According to the following codes, both positionId and productToken are passed via parameters in the function `adlDecreasePosition()`. There is no check whether `position.productId` and `productToken` are consistent.

The price of `token` uses the `productToken` to get from oracle.

As a result, the calculation will be incorrect if the parameter `productToken` is an incorrect value that does not match the `positionId`.

```
    function adlDecreasePosition(
        uint256 positionId,
        address productToken,
        address[] memory tokens,
        uint256[] memory prices
    ) external nonReentrant {
        require(liquidators[msg.sender], "not adl operator");
        IOracle(oracle).setPrices(tokens, prices);
        IKiloStorage.Position memory position = kiloStorage.getPositionById(positionId);
        IKiloStorage.KiloConfig memory kiloConfig = kiloStorage.getKiloConfig();
        uint256 price = IOracle(oracle).getPrice(productToken);

        IMarginFeeManager(marginFeeManager).updateMarginFee(position.productId);
        (int256 fundingPayment, uint256 borrowingFee) = PerpTradeUtil._getMarginFee(marginFeeManage
r, position.productId, position.isLong, position.leverage, position.margin, position.funding, positi
on.borrowing);

        int256 pnl = PerpTradeUtil._getPnl(position.isLong, position.price, position.leverage, posit
ion.margin, price) - fundingPayment - int256(borrowingFee);
        if (pnl >= 0 && uint256(pnl) > position.margin * kiloConfig.adlMultiplier) {
            return decreasePositionWithId(positionId, position.margin, ADL_ORDER_TYPE, price);
        }
    }
```

Recommendation

Yaodao : Recommend getting the product information via `position.productId`.

Client Response

Fixed

KEX-17:Potential logical flaw in `ReferralStorageManager::_setTraderReferralCode()`

Category	Severity	Status	Contributor
Logical	Medium	Fixed	Hacker007, LiRiu

Code Reference

- code/src/referrals/ReferralStorageManager.sol#L109-L113

```
109:     function _setTraderReferralCode(address _account, bytes32 _code) private {
110:         if (_code == bytes32(0) || traderReferralCodes[_account] != bytes32(0)) return;
111:         traderReferralCodes[_account] = _code;
112:         emit SetTraderReferralCode(_account, _code);
113:     }
```

Description

Hacker007 : The function `setTraderReferralCodeByUser()` is intended to set referral code for a user and ultimately call the function `_setTraderReferralCode()`.

```
    function setTraderReferralCodeByUser(bytes32 _code) external override {
        _setTraderReferralCode(msg.sender, _code);
    }
    function _setTraderReferralCode(address _account, bytes32 _code) private {
        if (_code == bytes32(0) || traderReferralCodes[_account] != bytes32(0)) return;
        traderReferralCodes[_account] = _code;
        emit SetTraderReferralCode(_account, _code);
    }
```

However, the function `_setTraderReferralCode` does not check whether the referral code generator is the trader himself. A malicious user can set a referral code for himself and gain extra profit.

LiRiu : _setTraderReferralCode does not check if `codeOwners[_code] != address(0)`. This creates a lack of strict invocation order between "registerCode" and "setTraderReferralCodeByUser". By guiding the Keeper to call "setTraderReferralCodeByUser" through the "OrderBook::createIncreaseOrder" function, the attacker binds the OrderBook with the "attacker_code". Subsequently, the attacker can register themselves as the owner of "attacker_code" through "registerCode". Ultimately, they set themselves as the referrer of the OrderBook.

Recommendation

Hacker007 : Check if the referral code is generated by the user himself.

```
    function _setTraderReferralCode(address _account, bytes32 _code) private {
        if (_code == bytes32(0) || traderReferralCodes[_account] != bytes32(0) || codeOwners[_code]
== _account) return;
        traderReferralCodes[_account] = _code;
        emit SetTraderReferralCode(_account, _code);
    }
```

LiRiu : To check if the code is registered within the `_setTraderReferralCode` function:

```
    function _setTraderReferralCode(address _account, bytes32 _code) private {
        if (_code == bytes32(0) || traderReferralCodes[_account] != bytes32(0)) return;
        require(codeOwners[_code] != address(0), "ReferralStorage: _code isn't exists");
        traderReferralCodes[_account] = _code;
        emit SetTraderReferralCode(_account, _code);
    }
```

Client Response

Fixed, We have made changes to address this recommendation and added a new method called setTraderReferralCode. This method can only be called by PositionRouter and OrderBook to ensure that the referral code generator is a trusted source.

KEX-18:Rounding down to zero issue in contract "KiloPriceFeed" function "getPrice "

Category	Severity	Status	Contributor
Precision Loss Errors	Medium	Declined	ginlee

Code Reference

- code/src/core/KiloPriceFeed.sol#L68

```
68:             return  price * (SPREAD_PRICE_BASE - spread) / SPREAD_PRICE_BASE;
```

Description

ginlee : division in solidity can result in rounding down, and to minimize precision loss we always need to do multiplication before division. Yet even when we do this some precision loss can occur especially when dealing with small numbers, and rounding down to zero can be a source of major error if not handled correctly. These is a chance that price * (SPREAD_PRICE_BASE - spread) is less than SPREAD_PRICE_BASE which will lead to rounding down to zero issue and make price 0. This problem also exists in other contracts

Recommendation

ginlee : To prevent this error always consider if your computation may round down to zero, especially when using small numbers, and if so whether your code should revert. For incoming assets, it’s recommended to round up the required amount. We could use solmate’s FixedPointMathLib library to calculate the quote and round up(mulDivUp function)

Client Response

Declined, price is 1e8 precision, and spread is small, so this problem will not happen

KEX-19:User can be unfairly liquidated when addMargin is not active

Category	Severity	Status	Contributor
Logical	Medium	Fixed	ladboy233

Code Reference

- code/src/core/PerpTrade.sol#L286

```
286:    function liquidatePositionsWithPrices(address[] memory tokens, uint256[] memory prices, uint
256[] calldata positionIds) external {
```

Description

ladboy233 : User can be unfairly liquidated when addMargin is not active

In PerpTrader.sol, there is a function addMargin

```
  function addMargin(uint256 productId, bool isLong, uint256 addedMargin) external nonReentrant {
        IERC20Upgradeable(token).safeTransferFrom(msg.sender, address(kiloStorage), addedMargin * to
kenBase / BASE);
        IProductManager.Product memory product  = IProductManager(productManager).getProduct(product
Id);
        // @audit when product is not active, can liquidate
        require(product.isActive, "not active");
        uint256 positionId = getPositionId(msg.sender, productId, isLong);
        (uint256 newMargin, uint256 newLeverage) = kiloStorage.addMargin(positionId, addedMargin, pr
oduct.minLeverage);
        emit MarginAdded(positionId, msg.sender, addedMargin, newMargin, newLeverage);
    }
```

However, when product.isActive is set to false, add margin is disabled

but liquidationPosition related function is always active, then User can be unfairly liquidated when addMargin is not active because when user does not have enough margin to make sure the position is sufficiently collateralized, and user cannot add margin (collateral)

user has no option but to watch their position get liquidated

```
    function liquidatePositionsWithPrices(address[] memory tokens, uint256[] memory prices, uint256
[] calldata positionIds) external {
        require(liquidators[msg.sender], "not liquidator");

        IOracle(oracle).setPrices(tokens, prices);
        IKiloStorage.KiloConfig memory kiloConfig = kiloStorage.getKiloConfig();

        uint256 totalLiquidatorReward;
        uint256 totalRemainReward;
        for (uint256 i = 0; i < positionIds.length; i++) {
            uint256 positionId = positionIds[i];
            (uint256 liquidatorReward, uint256 remainingReward) = _liquidatePosition(positionId, kil
oConfig);
            totalLiquidatorReward += liquidatorReward;
            totalRemainReward += remainingReward;
        }
        if (totalLiquidatorReward > 0) {
            IPendingReward(pendingReward).updateLiquidationReward(totalLiquidatorReward);
        }
        if (totalRemainReward > 0) {
            IPendingReward(pendingReward).updatePendingRewards(totalRemainReward);
        }
    }
```

Recommendation

ladboy233 : The recommendation is the protocol should consider disable the liquidation when the product is not in an active state

Client Response

Fixed

KEX-20: `setTraderReferralCodeByUser` function should use `tx.origin` instead of `msg.sender`.

Category	Severity	Status	Contributor
DOS	Medium	Fixed	LiRiu

Code Reference

- code/src/referrals/ReferralStorageManager.sol#L83-L85

```
83:     function setTraderReferralCodeByUser(bytes32 _code) external override {
84:         _setTraderReferralCode(msg.sender, _code);
85:     }
```

Description

LiRiu : To bind the `trader` and `code` in the `setTraderReferralCodeByUser` function correctly, you can use the `tx.origin` instead of `msg.sender`. This is because `tx.origin` refers to the original sender of the transaction, which will represent the user rather than the immediate calling contract.

The user is unable to bind their own referral code, which ultimately leads to the DoS of the referral code mechanism.

Recommendation

LiRiu : It is recommended to replace `msg.sender` with `tx.origin` in the `setTraderReferralCodeByUser` function.

Consider below fix in the `setTraderReferralCodeByUser` function

```
    function setTraderReferralCodeByUser(bytes32 _code) external override {
        _setTraderReferralCode(tx.origin, _code);
    }
```

Client Response

Fixed, only original sender can call this function

KEX-21: `tradeRewardDistributor` can potentially withdraw any amount of `xkiloToken` due to missing accounting

Category	Severity	Status	Contributor
Privilege Related	Medium	Declined	0xzoobi

Code Reference

- code/src/tradereward/ProtocolReward.sol#L40-L44

```
40:     function claimTradeReward(uint256 _amount) external whenNotPaused {
41:         require(msg.sender == tradeRewardDistributor, "ProtocolReward: not allowed");
42:         IPendingReward(pendingReward).withdrawProtocolReward();
43:         rewardToken.safeTransfer(msg.sender, _amount);
44:     }
```

Description

0xzoobi : The `claimXkiloReward` function can be called by `tradeRewardDistributor` who can claim the `XKilo` reward token which is 1:1 with their $KILO tokens. Currently as part of the function, the `tradeRewardDistributor` calls it with `_amount` and these tokens are transferred to the caller.

If the contract holds `XKilo` reward token then they can be easily stolen by `tradeRewardDistributor` since they is no condition limiting them from doing it.

Recommendation

0xzoobi : There are two recommendations to fix this

1. if the contract does not contain any other users funds and all the funds belong to `tradeRewardDistributor`, then add a check to see if the contract holds the specified balance to prevent unwanted reverts. Example: `require (xkiloToken.balanceOf(address(this)) >= _amount, "ProtocolReward: _amount > currentXKiloBalance");`.

A similar check can be added in `withdrawReward` function as well - `require (rewardToken.balanceOf(address(this)) >= _amount, "ProtocolReward: _amount > currentRewardToken");`

2. If not then use a mapping that holds the eligible rewards of all the users.

Client Response

Declined, The claimTradeReward function is called by tradeRewardDistributor. Normally, the IPendingReward(pendingReward).withdrawProtocolReward() function should retrieve a reward amount that is greater than _amount, so it should not be possible for the transfer amount to be insufficient. However, if the transfer amount is

insufficient, the safeTransfer function will perform a check and cause the transaction to fail, and manual intervention will be required to resolve the issue.

KEX-22: `var.pnl` Is Not Checked Against Total Short Size when Exiting Position Size, Potentially Resulting in Widespread Protocol Error

Category	Severity	Status	Contributor
Logical	Medium	Acknowledged	xfu

Code Reference

- code/src/core/PerpTrade.sol#L218-L283

```
218:    function decreasePositionWithId(
219:        uint256 positionId,
220:        uint256 margin,
221:        uint256 orderType,
222:        uint256 oraclePrice
223:    )  internal {
224:        _validateRouter();
225:        IKiloStorage.Position memory position = kiloStorage.getPositionById(positionId);
226:        IProductManager.Product memory product  = IProductManager(productManager).getProduct(pos
ition.productId);
227:        DecreasePositionVars memory vars;
228:        if (margin >= uint256(position.margin)) {
229:            margin = uint256(position.margin);
230:            vars.isFullClose = true;
231:        }
232:        IKiloStorage.KiloConfig memory kiloConfig = kiloStorage.getKiloConfig();
233:        if (orderType == ADL_ORDER_TYPE) {
234:            vars.price = oraclePrice;
235:        } else {
236:            vars.price = _calculatePrice(product.productId, product.productToken, !position.isLo
ng,
237:                _getMaxExposure(uint256(product.weight), kiloConfig.exposureMultiplier),
238:                uint256(product.reserve), margin * position.leverage / BASE, kiloConfig.maxShif
t);
239:        }
240:        (int256 funding, uint256 borrowing) = IMarginFeeManager(marginFeeManager).updateMarginFe
e(position.productId);
241:        kiloStorage.updateDecreaseOpenInterest(product.productId, margin * uint256(position.leve
rage) / BASE, position.isLong);
242:
243:        (vars.fundingPayment, vars.borrowingFee) = PerpTradeUtil._getMarginFee(marginFeeManager,
position.productId, position.isLong, position.leverage, margin, position.funding, position.borrowin
g);
244:        vars.pnl = PerpTradeUtil._getPnl(position.isLong, uint256(position.price), uint256(posit
ion.leverage), margin, vars.price) - vars.fundingPayment - int256(vars.borrowingFee);
245:        if (vars.pnl < 0 && uint256(- 1 * vars.pnl) >= margin * kiloConfig.liquidationThreshold
/ (10 ** 4)) {
246:            margin = uint256(position.margin);
247:            vars.pnl = - 1 * int256(uint256(position.margin));
```

```
248:             vars.isLiquidatable = true;
249:         } else {
250:             if (vars.pnl > 0 && !PerpTradeUtil._canTakeProfit(position.isLong, uint256(position.
lastIncreasedTime), uint256(position.oraclePrice),
251:                 IOracle(oracle).getPrice(product.productToken), product.minPriceChange, kiloConf
ig.minProfitTime)) {
252:                 vars.pnl = 0;
253:             }
254:         }
255:
256:         (vars.totalFee, vars.remainMargin) = IPendingReward(pendingReward).updatePendingPnlAndGe
tFee(vars.pnl, position, margin, product.productToken, positionId, productManager);
257:         if (vars.remainMargin > 0) {
258:             kiloStorage.transferRemainMargin(position.account, uint256(vars.remainMargin) * toke
nBase / BASE);
259:         }
260:
261:         emit DecreasePosition(
262:             positionId,
263:             position.account,
264:             uint256(position.productId),
265:             vars.price,
266:             uint256(position.price),
267:             margin,
268:             uint256(position.leverage),
269:             vars.totalFee,
270:             vars.pnl,
271:             vars.fundingPayment,
272:             vars.isLiquidatable,
273:             orderType,
274:             vars.borrowingFee
275:         );
276:
277:         if (vars.isFullClose) {
278:             kiloStorage.clearPosition(positionId);
279:         } else {
280:             position.margin -= uint64(margin);
281:             kiloStorage.updatePositionMargin(positionId, position.margin);
282:         }
283:     }
```

Description

xfu : When decreasing from the pool via `decreasePosition()`, an internal call is made to `decreasePositionWithId()` and call internal function `_validateRouter()` which validate execution permission and reverting if caller is not allowed. However, it does not check whether the value of the `var.pnl` in the tranche can accommodate the total open short position size. If this is not the case, then it is possible for the short profits on a token to exceed the value of the new value of the token in the tranche. This would lead to a negative AUM, causing multiple functions to revert in the future.

Recommendation

xfu : Implement an error for the case where the new value of the tranche (with respect to an asset) is lower than total short size when decreasing position.

```
function decreasePositionWithId(
        uint256 positionId,
        uint256 margin,
        uint256 orderType,
        uint256 oraclePrice
    )  internal {
        // ...

        vars.pnl = PerpTradeUtil._getPnl(position.isLong, uint256(position.price), uint256(position.
leverage), margin, vars.price) - vars.fundingPayment - int256(vars.borrowingFee);
        if (vars.pnl < 0 && uint256(- 1 * vars.pnl) >= margin * kiloConfig.liquidationThreshold / (1
0 ** 4)) {
            margin = uint256(position.margin);
            vars.pnl = - 1 * int256(uint256(position.margin));
            vars.isLiquidatable = true;
        } else {
            if (vars.pnl > 0 && !PerpTradeUtil._canTakeProfit(position.isLong, uint256(position.last
IncreasedTime), uint256(position.oraclePrice),
                IOracle(oracle).getPrice(product.productToken), product.minPriceChange, kiloConfig.m
inProfitTime)) {
                vars.pnl = 0;
            }
        }

        // revert if pnl is less than total position short size.

        (vars.totalFee, vars.remainMargin) = IPendingReward(pendingReward).updatePendingPnlAndGetFee
(vars.pnl, position, margin, product.productToken, positionId, productManager);

        // ...
    }
```

Client Response

Acknowledged

KEX-23:access control validation did not serve its purpose in "vaultstakeaward" contract "deposit" function

Category	Severity	Status	Contributor
Logical	Medium	Declined	ginlee

Code Reference

- code/src/core/VaultStakeReward.sol#L90

```
90:         require(msg.sender == user, "Vault: not staker");
```

Description

ginlee : `function deposit(uint256 amount, address user) public override nonReentrant returns (uint256) { require(kiloConfig.canUserStake(), "Vault: not allowed"); require(msg.sender == user, "Vault: not staker");` An attacker can simply use his own address as user parameter and bypass the validation

Recommendation

ginlee : To protect user and system security, it is recommended to implement appropriate access control mechanisms to ensure that only authorized users can invoke the function and perform necessary validation and authorization checks on user actions.

Client Response

Declined, Anyone can make a deposit, but the user variable is inherited from the parent class. To ensure that users can only make a deposit for themselves, so add a check for msg.sender == user

KEX-24:sanity check missing in `setMinBorrowingRate` and `setMaxBorrowingRate`

Category	Severity	Status	Contributor
Logical	Medium	Fixed	0xzoobi

Code Reference

- code/src/core/MarginFeeManager.sol#L128-L136

```
128:    function setMaxBorrowingRate(uint256 _maxBorrowingRate) external onlyOwner {
129:        maxBorrowingRate = _maxBorrowingRate;
130:        emit OwnerSetMaxBorrowingRate(_maxBorrowingRate);
131:    }
132:
133:    function setMinBorrowingRate(uint256 _minBorrowingRate) external onlyOwner {
134:        minBorrowingRate = _minBorrowingRate;
135:        emit OwnerSetMinBorrowingRate(_minBorrowingRate);
136:    }
```

Description

0xzoobi : The current implementation of `setMinBorrowingRate` does not check if the new `_minBorrowingRate` is greater than the current `maxBorrowingRate`.

Similarly, the `setMaxBorrowingRate` does not check if the new `_maxBorrowingRate` is less than the current `minBorrowingRate`.

Setting a wrong value in either of the function may lead to unexpected behaviour.

The important functions using these values include `getBorrowingRate` which is in turn called by `updateMarginFee`., which may result in calculating wrong values or revert.

Recommendation

0xzoobi : Add the below sanity check for `setMinBorrowingRate` and `setMaxBorrowingRate`

```
function setMaxBorrowingRate(uint256 _maxBorrowingRate) external onlyOwner {
        require(_maxBorrowingRate > minBorrowingRate, "_maxBorrowingRate  should be greater than min
BorrowingRate");
        maxBorrowingRate = _maxBorrowingRate; //@audit-issue -> should check if greater than minBorr
owingRate
        emit OwnerSetMaxBorrowingRate(_maxBorrowingRate);
    }

    function setMinBorrowingRate(uint256 _minBorrowingRate) external onlyOwner {
        require(_minBorrowingRate  < maxBorrowingRate , "_minBorrowingRate  should be less than maxB
orrowingRate ");
        minBorrowingRate = _minBorrowingRate; //@audit-issue -> should check if lesser than maxBorro
wingRate
        emit OwnerSetMinBorrowingRate(_minBorrowingRate);
    }
```

Client Response

Fixed

KEX-25: Division before Multiplication in `PerpTradeUtil.sol`

Category	Severity	Status	Contributor
Logical	Low	Fixed	0xzoobi

Code Reference

- code/src/libraries/PerpTradeUtil.sol#L108-L117

```
108:     function _getTradeFee(
109:         uint256 margin,
110:         uint256 leverage,
111:         uint256 productId,
112:         address user,
113:         address productManager
114:     ) internal view returns(uint256) {
115:         uint256 feeRate = IProductManager(productManager).getTradeFeeRate(productId, user);
116:         return margin * leverage / BASE * feeRate / 10**4;
117:     }
```

Description

0xzoobi : The current implementation of `_getTradeFee` uses the following formula to calculate trading fee - `margin * leverage / BASE * feeRate / 10**4;`. Because Solidity integer division may truncate, it is often preferable to do multiplication before division to prevent precision loss.

The impact of this would be trader paying less trading fee, In long run, it is a loss of trading fee from all the users to the KiloEx Platform.

Run the following PoC in Remix with Given sample values

```
contract MathCheckPoCOne {

    uint256 private constant BASE = 10**8;

    //Test with your known values to verify
    uint256 public margin = 0.242342342234 ether;
    uint256 public leverage = 3;
    uint256 public feeRate = 0.0023424123223 ether;

    function testOne() public returns(uint256) { //loss of precision
        return margin * leverage / BASE * feeRate / 10**4;
    }

    function testTwo() public returns(uint256) {//recommended approach
        return margin * leverage * feeRate / (10**4 * BASE);
    }
}

Output
testOne ->170299706598721110 5410
testTwo ->170299706599189593 0054

---

contract MathCheckPoCTwo {

    uint256 private constant BASE = 10**8;

    uint256 public margin = 0.111423424342342234 ether;
    uint256 public leverage = 3;
    uint256 public feeRate = 0.00003424123223 ether;

    function testOne() public returns(uint256) {
        return margin * leverage / BASE * feeRate / 10**4;
    }

    function testTwo() public returns(uint256) {
        return margin * leverage * feeRate / (10**4 * BASE);
    }
}

Output
testOne ->11445826045378498790
```

```
  testTwo ->1144582604630392636 9
```

Recommendation

0xzoobi : Multiple-before-Divide. The updated formula to calculate the `TradingFee` should look like `margin * leverage * feeRate / (10**4 * BASE);`

Client Response

Fixed

KEX-26: Division before Multiplication in `VaultStakeReward.sol`

Category	Severity	Status	Contributor
Code Style	Low	Fixed	Yaodao

Code Reference

- code/src/core/VaultStakeReward.sol#L134

```
134:         require(kiloConfig.totalOpenInterest() <= (IERC20Upgradeable(token).balanceOf(address(th
is)) - assets) * BASE / tokenBase * kiloConfig.getKiloConfig().utilizationMultiplier / (10 ** 4), "!
utilized");
```

Description

Yaodao : Performing integer division before multiplication truncates the low bits, losing the precision of calculation.

Consider below codes

```
        require(kiloConfig.totalOpenInterest() <= (IERC20Upgradeable(token).balanceOf(address(this))
- assets) * BASE / tokenBase * kiloConfig.getKiloConfig().utilizationMultiplier / (10 ** 4), "!utili
zed");
```

Recommendation

Yaodao : Recommend applying multiplication before division to avoid loss of precision.

Client Response

Fixed

KEX-27:Array length if not validated when calling a function can cause reverts

Category	Severity	Status	Contributor
Code Style	Low	Fixed	0xzoobi

Code Reference

- code/src/peripherals/LiquidationPriceReader.sol#L55-L59
- code/src/peripherals/LiquidationPriceReader.sol#L95-L99
- code/src/core/PerpTrade.sol#L286

```
55:    function getLiquidationPrice(
56:        address[] memory account,
57:        uint256[] memory productId,
58:        bool[] memory isLong
59:    ) external view returns(

95:    function getLiquidationPrices(
96:        address[] memory account,
97:        uint256[] memory productId,
98:        bool[] memory isLong
99:    ) external view returns (

286:    function liquidatePositionsWithPrices(address[] memory tokens, uint256[] memory prices, uint
256[] calldata positionIds) external {
```

Description

0xzoobi : When calling a function which has multiple arrays as input and it is expected that all the arrays have the same size, we need a add a check for this.

These types of checks are already present in the existing code. Example: code/src/core/KiloPriceFeed.sol#L147-L148

Adding a check will make the function call less error prone.

Recommendation

0xzoobi : Add a check to see if the length of all the three input arrays are equal.

Sample Fix

```
require(account.length == productId.length, "Invalid Length");
require(account.length == isLong.length, "Invalid Length");
```

Client Response

Fixed

KEX-28:Consider using OpenZeppelin's SafeCast library to prevent unexpected overflows in `KiloStorageManager`::`storeIncreasePosition()`

Category	Severity	Status	Contributor
Integer Overflow and Underflow	Low	Fixed	ginlee

Code Reference

- code/src/core/VaultStakeReward.sol#L97-L98
- code/src/core/VaultStakeReward.sol#L136
- code/src/core/VaultStakeReward.sol#L195-L198
- code/src/core/KiloStorageManager.sol#L272-L279
- code/src/core/PerpTrade.sol#L280
- code/src/core/KiloStorageManager.sol#L285
- code/src/core/KiloStorageManager.sol#L296-L297

```
97:         _stake.timestamp = uint128(block.timestamp);
98:         _stake.amount += uint96(newAmount);

136:         _stake.amount -= uint96(amount);

195:         _fromStakeItem.amount -= uint96(changedAmounts);
196:         _toStakeItem.amount += uint96(changedAmounts);
197:         _toStakeItem.owner = to;
198:         _toStakeItem.timestamp = uint128(Math.max(_fromStakeItem.timestamp, _toStakeItem.timesta
mp));

272:             productId:uint88(productId),
273:             margin : uint128(margin),
274:             leverage : uint128(leverage),
275:             price : uint128(price),
276:             oraclePrice : uint128(oraclePrice),
277:             funding : int64(funding),
278:             borrowing : uint64(borrowing),
279:             lastIncreasedTime : uint128(block.timestamp)

280:             position.margin -= uint64(margin);

285:         position.margin = uint64(newMargin);

296:         position.margin = uint64(newMargin);
297:         position.leverage = uint64(newLeverage);
```

Description

ginlee : Consider using OpenZeppelin's SafeCast library to prevent unexpected overflows when casting from uint256, the SafeCast library provides safe casting functions that ensure that the result of a cast does not overflow or underflow the destination type

Recommendation

ginlee : import SafeCast library and Use productId.toUint88() instead of uint88(productId), change others in the same pattern

Client Response

Fixed

KEX-29:Division before Multiplication in `MarginFeeManager.sol` 、`OrderBook.sol` and `PerpTradeUtil.sol`

Category	Severity	Status	Contributor
Logical	Low	Fixed	Hacker007

Code Reference

- code/src/core/MarginFeeManager.sol#L85
- code/src/libraries/PerpTradeUtil.sol#L115
- code/src/core/OrderBook.sol#L388

```
85:         uint256 borrowFeeRate = (openInterestLong + openInterestShort) * BASE / (kiloConfig.maxEx
posureMultiplier * maxExposure) * product.borrowingFactor * FEE_BASE / BASE / (10 ** 4);

115:         uint256 feeRate = IProductManager(productManager).getTradeFeeRate(productId, user);

388:             uint256 margin = (order.margin + order.tradeFee) * BASE / (BASE + getTradeFeeRate(or
der.productId, order.account) * _leverage / 10**4);
```

Description

Hacker007 : Mathematical operations in the aforementioned function perform divisions before multiplications. Performing multiplication before division can sometimes avoid loss of precision.

Recommendation

Hacker007 : We advise the client to apply multiplications before divisions if integer overflow would not happen in functions.

Client Response

Fixed

KEX-30:Follow `checks-effects-interaction` pattern

Category	Severity	Status	Contributor
Logical	Low	Mitigated	0xzoobi

Code Reference

- code/src/core/PendingReward.sol#L79-L88
- code/src/core/PendingReward.sol#L90-L99
- code/src/core/PendingReward.sol#L101-L110

```
79:     function withdrawProtocolReward() external nonReentrant returns (uint256) {
80:         require(msg.sender == protocolRewardReceiver, "PendingReward: not allowed");
81:         uint256 _pendingReward = rewardReserves[RewardReserveType.PROTOCOL] * tokenBase / BASE;
82:         if (_pendingReward > 0) {
83:             kiloStorage.transferPendingReward(protocolRewardReceiver, _pendingReward);
84:             rewardReserves[RewardReserveType.PROTOCOL] = 0;
85:             emit WithdrawReward(RewardReserveType.PROTOCOL, protocolRewardReceiver, _pendingRewar
d);
86:         }
87:         return _pendingReward;
88:     }

90:     function withdrawLiquidationReward() external nonReentrant returns (uint256) {
91:         require(msg.sender == liquidationRewardReceiver, "PendingReward: not allowed");
92:         uint256 _pendingReward = rewardReserves[RewardReserveType.LIQUIDATION] * tokenBase / BAS
E;
93:         if (_pendingReward > 0) {
94:             kiloStorage.transferPendingReward(liquidationRewardReceiver, _pendingReward);
95:             rewardReserves[RewardReserveType.LIQUIDATION] = 0;
96:             emit WithdrawReward(RewardReserveType.LIQUIDATION, liquidationRewardReceiver, _pendin
gReward);
97:         }
98:         return _pendingReward;
99:     }

101:     function withdrawVaultReward() external nonReentrant returns (uint256) {
102:         require(msg.sender == vaultRewardReceiver, "PendingReward: not allowed");
103:         uint256 _pendingReward = rewardReserves[RewardReserveType.VAULT] * tokenBase / BASE;
104:         if (_pendingReward > 0) {
105:             kiloStorage.transferPendingReward(vaultRewardReceiver, _pendingReward);
106:             rewardReserves[RewardReserveType.VAULT] = 0;
107:             emit WithdrawReward(RewardReserveType.VAULT, vaultRewardReceiver, _pendingReward);
108:         }
109:         return _pendingReward;
110:     }
```

Description

0xzoobi : The `withdrawProtocolReward` has a `nonReentrant` to prevent any possible re-entrancy risks while calling the function. But just adding a Re-entrancy guard does not solve the issue.

The function does not follow `checks-effects-interaction` pattern, As a result, the external call is made first followed by the update of state variables.

This may pose a cross-function re-entrancy or cross-contract re-entrancy risks.

Recommendation

0xzoobi : Use the `Checks-Effects-Interactions` best practice and make all state changes before calling external contracts.

Client Response

Mitigated, These methods are called by the protocol or EOA accounts from Kilo and there is no re-entrancy risk.

KEX-31:For upgrading contracts, add _disableInitializers() to the constructor that implements the contract

Category	Severity	Status	Contributor
Logical	Low	Fixed	Hacker007, seek.guo

Code Reference

- code/src/peripherals/LiquidationPriceReader.sol#L18-L24

```
18:     constructor(
19:         address _kiloStorageAddr
20:     ) {
21:         kiloStorage = IKiloStorage(_kiloStorageAddr);
22:         marginFeeManager = kiloStorage.marginFeeManagerAddr();
23:         oracle = IOracle(kiloStorage.kiloPriceFeedAddr());
24:     }
```

Description

Hacker007 : All the logic contracts with the `initialize()` function have the same issue. An attacker can call the function `initialize()` and assume ownership of the logic contract. As a result, she can perform privileged operations that can trick the investors that she is the owner of the upgradeable contract.

```
    function initialize(

    ) public initializer {
        discountEnabled = false;
        __owner_governable_init();
    }
```

seek.guo : Do not leave an implementation contract uninitialized. An uninitialized implementation contract can be taken over by an attacker, which may impact the proxy.

Recommendation

Hacker007 : Call the function `_disableInitializers` in the constructor of every contract to prevent any future reinitialization.

```
constructor() {
  _disableInitializers();
}
```

seek.guo : To prevent the implementation contract from being used, you should invoke the _disableInitializers function in the constructor to automatically lock it when it is deployed

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() {
    _disableInitializers();
}
```

Client Response

Fixed, The LiquidationPriceReader contract has been modified to be an immutable contract that cannot be upgraded.

KEX-32:Incompatibility With Deflationary Tokens

Category	Severity	Status	Contributor
Logical	Low	Declined	Yaodao, Hacker007

Code Reference

- code/src/core/OrderBook.sol#L15
- code/src/core/PositionRouter.sol#L15
- code/src/core/VaultStakeReward.sol#L16
- code/src/core/PerpTrade.sol#L16
- code/src/core/VaultStakeReward.sol#L88-L104

```
15:contract OrderBook is OwnerGovernableUpgradeable, ReentrancyGuardUpgradeable {

15:contract PositionRouter is OwnerGovernableUpgradeable, ReentrancyGuardUpgradeable {

16:contract PerpTrade is IPerpTrade, OwnerGovernableUpgradeable, ReentrancyGuardUpgradeable {

16:contract VaultStakeReward is OwnerGovernableUpgradeable, ReentrancyGuardUpgradeable, PausableUpgr
adeable, ERC20Upgradeable, ERC4626Upgradeable {

88:    function deposit(uint256 amount, address user) public override nonReentrant returns (uint256)
{
89:        require(kiloConfig.canUserStake(), "Vault: not allowed");
90:        require(msg.sender == user, "Vault: not staker");
91:        require(IERC20Upgradeable(token).balanceOf(address(this)) * BASE / tokenBase + amount <=
cap, "over cap");
92:        _claimVaultPendingReward();
93:
94:        Stake storage _stake = stakes[user];
95:        uint256 newAmount = amount * tokenBase / BASE;
96:        _stake.owner = user;
97:        _stake.timestamp = uint128(block.timestamp);
98:        _stake.amount += uint96(newAmount);
99:        staked += uint256(newAmount);
100:        uint256 shares = previewDeposit(newAmount);
101:        _deposit(user, user, newAmount, shares);
102:        emit StakedRedeemItem(user, StakeType.Stake, int256(newAmount), int256(shares), _stake.a
mount, IERC20Upgradeable(address(this)).balanceOf(user), 0);
103:        return shares * BASE / tokenBase;
104:    }
```

Description

Yaodao : When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. As a result, an inconsistency in the amount will occur and the transaction may fail due to the validation checks. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee) to the target contract, only 90 tokens actually arrive to the contract.

Hacker007 : When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. For example, if a user stakes 100 deflationary tokens (with a 10% transaction

fee) in a VaultStakeReward contract, only 90 tokens actually arrive in the contract. However, the user can still withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Reference:

- https://thoreum-finance.medium.com/what-exploit-happened-today-for-gocerberus-and-garuda-also-for-lokum-ybear-piggy-caramelswap-3943ee23a39f

Recommendation

Yaodao : Recommend regulating the set of tokens supported and adding necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.
Hacker007 : Regulate the set of pool tokens supported and adding necessary mitigation mechanisms to keep track of accurate balances, if there is a need to support deflationary tokens.

Client Response

Declined, Will not support deflationary tokens

KEX-33:Lack of `sequencer` logic for L2 network

Category	Severity	Status	Contributor
Logical	Low	Declined	Yaodao

Code Reference

- code/src/core/KiloPriceFeed.sol#L122-L136

```
122:    function getChainlinkPrice(address token) public view returns (uint256 priceToReturn, uint25
6 chainlinkTimestamp) {
123:        require(token != address(0), '!feed-error');
124:
125:        (,int256 price,,uint256 timeStamp,) = AggregatorV3Interface(token).latestRoundData();
126:
127:        require(price > 0, '!price');
128:        require(timeStamp > 0, '!timeStamp');
129:        uint8 decimals = AggregatorV3Interface(token).decimals();
130:        chainlinkTimestamp = timeStamp;
131:        if (decimals != 8) {
132:            priceToReturn = uint256(price) * (10**8) / (10**uint256(decimals));
133:        } else {
134:            priceToReturn = uint256(price);
135:        }
136:    }
```

Description

Yaodao : In the function `getChainlinkPrice()`, `sequencer` is not checked.

Optimistic rollup protocols move all execution off the layer 1 (L1) Ethereum chain, complete execution on a layer 2 (L2) chain, and return the results of the L2 execution back to the L1. These protocols have a sequencer that executes and rolls up the L2 transactions by batching multiple transactions into a single transaction.

If a sequencer becomes unavailable, it is impossible to access read/write APIs that consumers are using and applications on the L2 network will be down for most users without interacting directly through the L1 optimistic rollup contracts. The L2 has not stopped, but it would be unfair to continue providing service on your applications when only a few users can use them.

To help your applications identify when the sequencer is unavailable, you can use a data feed that tracks the last known status of the sequencer at a given point in time. This helps you prevent mass liquidations by providing a grace period to allow customers to react to such an event.

Recommendation

Yaodao : Recommend creating the consumer contract for sequencer uptime feeds similarly to the contracts that you use for other Chainlink Data Feeds.

Consider below fix

```
    uint256 private constant GRACE_PERIOD_TIME = 3600;

    error SequencerDown();
    error GracePeriodNotOver();

    function getChainlinkPrice(address token) public view returns (uint256 priceToReturn, uint256 ch
ainlinkTimestamp) {
        require(token != address(0), '!feed-error');

        (,int256 answer,uint256 startedAt,,) = AggregatorV3Interface(sequencerUptimeFeed).latestRoun
dData();

        // Answer == 0: Sequencer is up
        // Answer == 1: Sequencer is down
        bool isSequencerUp = answer == 0;
        if (!isSequencerUp) {
            revert SequencerDown();
        }

        uint256 timeSinceUp = block.timestamp - startedAt;
        if (timeSinceUp <= GRACE_PERIOD_TIME) {
            revert GracePeriodNotOver();
        }

        require(price > 0, '!price');
        require(timeStamp > 0, '!timeStamp');
        uint8 decimals = AggregatorV3Interface(token).decimals();
        chainlinkTimestamp = timeStamp;
        if (decimals != 8) {
            priceToReturn = uint256(price) * (10**8) / (10**uint256(decimals));
        } else {
            priceToReturn = uint256(price);
        }
    }
```

Client Response

Declined, We currently do not have any plans to deploy on Layer 2. If we decide to do so in the future, we will create a new price feed implementation similar to the existing one.

KEX-34:Lack of check the `_leverage` in `OrderBook` contract `updateIncreaseOrder` function

Category	Severity	Status	Contributor
Logical	Low	Fixed	danielt

Code Reference

- code/src/core/OrderBook.sol#L303-L335

```
303:    function createIncreaseOrder(
304:        uint256 _productId,
305:        uint256 _margin,
306:        uint256 _leverage,
307:        bool _isLong,
308:        uint256 _triggerPrice,
309:        bool _triggerAboveThreshold,
310:        uint256 _executionFee,
311:        bytes32 _referralCode
312:    ) external payable nonReentrant {
313:        require(_executionFee >= minExecutionFee, "OrderBook: insufficient execution fee");
314:        require(_margin >= kiloStorage.getKiloConfig().minMargin, "OrderBook: insufficient colla
teral");
315:
316:        uint256 tradeFee = getTradeFeeRate(_productId, msg.sender) * _margin * _leverage / (FEE_
BASE * BASE);
317:        require(msg.value == _executionFee, "OrderBook: incorrect execution fee transferred");
318:
319:        if (_referralCode != bytes32(0) && referralStorage != address(0)) {
320:            IReferralStorage(referralStorage).setTraderReferralCodeByUser(_referralCode);
321:        }
322:
323:        IERC20Upgradeable(token).safeTransferFrom(msg.sender, address(this), (_margin + tradeFe
e) * tokenBase / BASE);
324:        _createIncreaseOrder(
325:            msg.sender,
326:            _productId,
327:            _margin,
328:            tradeFee,
329:            _leverage,
330:            _isLong,
331:            _triggerPrice,
332:            _triggerAboveThreshold,
333:            _executionFee
334:        );
335:    }
```

Description

danielt : In the `createIncreaseOrder` function and the `updateIncreaseOrder` function, there is no zero value check on `_leverage` to ensure it is greater than zero.

If `_leverage` is zero, then the `tradeFee` will be zero, which is an unexpected result:

```
uint256 tradeFee = getTradeFeeRate(_productId, msg.sender) * _margin * _leverage / (FEE_BASE * BAS
E);
```

Recommendation

danielt : Adding zero value checks on `_leverage`, in the `createIncreaseOrder` function and the `updateIncreaseOrder` function.

Client Response

Fixed

KEX-35:Lack of logic to deal with insufficient tokens in the function `decrVaultBalance()`

Category	Severity	Status	Contributor
Logical	Low	Mitigated	Yaodao

Code Reference

- code/src/core/VaultStakeReward.sol#L151-L154

```
151:    function decrVaultBalance(uint256 decreasedBalance) external {
152:        require(msg.sender == pendingReward, "Vault: not allowed");
153:        IERC20Upgradeable(token).safeTransfer(address(kiloConfig), decreasedBalance * tokenBase
/ BASE);
154:    }
```

Description

Yaodao : The function `decrVaultBalance()` is called by the `pendingReward` contract to transfer out the users' profit. However, there is no logic to deal with the balance of vault is insufficient to transfer. As a result, the users can't get profit immediately in this condition, and the profit may even decrease as the time goes.

Consider below codes

```
    function decrVaultBalance(uint256 decreasedBalance) external {
        require(msg.sender == pendingReward, "Vault: not allowed");
        IERC20Upgradeable(token).safeTransfer(address(kiloConfig), decreasedBalance * tokenBase / BA
SE);
    }
```

Recommendation

Yaodao : Recommend adding logic to deal with insufficient tokens.

Client Response

Mitigated, At present, there are monitoring and alarms offline, and manual intervention will be performed to deal with this situation.

KEX-36:Lack of repeatability check in the function `addProduct()`

Category	Severity	Status	Contributor
Logical	Low	Fixed	Yaodao

Code Reference

- code/src/core/ProductManager.sol#L52-L68

```
52:     function addProduct(address token, Product memory _product) external onlyOwner {
53:         require(_product.maxLeverage > _product.minLeverage && token != address(0));
54:         require(_product.minLeverage >= 1 * BASE);
55:         id[token] = _product.productId;
56:         productToken[_product.productId] = _product.productToken;
57:         maxLeverage[token] = _product.maxLeverage;
58:         minLeverage[token] = _product.minLeverage;
59:         fee[token] = _product.fee;
60:         isActive[token] = _product.isActive;
61:         minPriceChange[token] = _product.minPriceChange;
62:         weight[token] = _product.weight;
63:         reserve[token] = _product.reserve;
64:         borrowingFactor[token] = _product.borrowingFactor;
65:         maxPositionSize[token] = _product.maxPositionSize;
66:         totalWeight = totalWeight + _product.weight;
67:         emit ProductAdded(_product.productId, token, this.getProduct(id[token]));
68:     }
```

Description

Yaodao : According to the following codes, the function `addProduct()` is used to add the parameters for the given token to add a new product. However, the parameters updated the data with the given token and the given token is not checked whether equal to the `_product.productToken`. As a result, the two tokens can be different and the call will not fail.

Consider below codes

```
    function addProduct(address token, Product memory _product) external onlyOwner {
        require(_product.maxLeverage > _product.minLeverage && token != address(0));
        require(_product.minLeverage >= 1 * BASE);
        id[token] = _product.productId;
        productToken[_product.productId] = _product.productToken;
        maxLeverage[token] = _product.maxLeverage;
        minLeverage[token] = _product.minLeverage;
        fee[token] = _product.fee;
        isActive[token] = _product.isActive;
        minPriceChange[token] = _product.minPriceChange;
        weight[token] = _product.weight;
        reserve[token] = _product.reserve;
        borrowingFactor[token] = _product.borrowingFactor;
        maxPositionSize[token] = _product.maxPositionSize;
        totalWeight = totalWeight + _product.weight;
        emit ProductAdded(_product.productId, token, this.getProduct(id[token]));
    }
```

Recommendation

Yaodao : Recommend adding check of the token.

Consider below fix in the `addProduct()` function

```
require(token == _product.productToken, "incorrect token");
```

Client Response

Fixed

KEX-37:MarginFeeManager miss inheritance

Category	Severity	Status	Contributor
Logical	Low	Fixed	seek.guo

Code Reference

- code/src/tradereward/interfaces/IProtocolReward.sol#L4-L10
- code/src/interfaces/IMarginFeeManager.sol#L5-L18
- code/src/tradereward/ProtocolReward.sol#L11-L82
- code/src/core/MarginFeeManager.sol#L12-L158

```
4:interface IProtocolReward {
5:    function claimProtocolReward() external;
6:    function claimTradeReward(uint256 _amount) external;
7:    function claimXkiloReward(uint256 _amount) external;
8:    function withdrawReward(uint256 _amount) external;
9:}

5:interface IMarginFeeManager {
6:    function updateMarginFee(uint256) external returns(int256,uint256);
7:
8:    function getFunding(uint256) external view returns(int256);
9:    function getFundingRate(uint256 _productId) external view returns(int256);
10:    function getCurrentCumulativeFunding(uint256 _productId) external view returns(int256 funding
Rate, int256 cumulativeFunding);
11:
12:    function getBorrowing(uint256) external view returns(uint256);
13:    function getBorrowingRate(uint256 _productId) external view returns(uint256);
14:    function getCurrentCumulativeBorrowing(uint256 _productId) external view returns(uint256 borr
owingRate, uint256 cumulativeBorrowing);
15:
16:    function getFundingBorrowing(uint256 _productId) external view returns(int256 funding, uint25
6 borrowing);
17:}

11:contract ProtocolReward is OwnerGovernableUpgradeable, PausableUpgradeable, ReentrancyGuardUpgrad
eable {
12:    using SafeERC20Upgradeable for IERC20Upgradeable;
13:    uint256 public totalClaimed;
14:    address private pendingReward;
15:    address private protocolRewardReceiver;
16:    address private tradeRewardDistributor;
17:    IERC20Upgradeable public rewardToken;
18:    IERC20Upgradeable public xkiloToken;
19:
20:    event TokenWithdrawnOwner(address to, uint256 amount);
21:    event OwnerSetProtocolRewardReceiver(address protocolRewardReceiver);
22:    event OwnerSetTradeRewardDistributor(address tradeRewardDistributor);
23:
24:    function initialize(
25:        address _rewardToken, address _pendingReward, address _protocolRewardReceiver
26:    ) public initializer {
27:        __owner_governable_init();
```

```
28:         __ReentrancyGuard_init();
29:         __Pausable_init();
30:         rewardToken = IERC20Upgradeable(_rewardToken);
31:         pendingReward = _pendingReward;
32:         protocolRewardReceiver = _protocolRewardReceiver;
33:     }
34:
35:
36:     function setXkiloToken(address _xkiloToken) external onlyOwner {
37:         xkiloToken = IERC20Upgradeable(_xkiloToken);
38:     }
39:
40:     function claimTradeReward(uint256 _amount) external whenNotPaused {
41:         require(msg.sender == tradeRewardDistributor, "ProtocolReward: not allowed");
42:         IPendingReward(pendingReward).withdrawProtocolReward();
43:         rewardToken.safeTransfer(msg.sender, _amount);
44:     }
45:
46:     function claimXkiloReward(uint256 _amount) external whenNotPaused {
47:         require(msg.sender == tradeRewardDistributor, "ProtocolReward: not allowed");
48:         require (address(xkiloToken) != address(0), "ProtocolReward: xkiloToken zero");
49:         xkiloToken.safeTransfer(msg.sender, _amount);
50:     }
51:
52:     //anyone can claim, just from pendingReward to address(this)
53:     function claimProtocolReward() external whenNotPaused {
54:         IPendingReward(pendingReward).withdrawProtocolReward();
55:     }
56:
57:     function setProtocolRewardReceiver(address _receiver) external onlyOwner {
58:         protocolRewardReceiver = _receiver;
59:         emit OwnerSetProtocolRewardReceiver(_receiver);
60:     }
61:
62:     function setTradeRewardDistributor(address _tradeRewardDistributor) external onlyOwner {
63:         tradeRewardDistributor = _tradeRewardDistributor;
64:         emit OwnerSetTradeRewardDistributor(_tradeRewardDistributor);
65:     }
66:
67:     function withdrawReward(uint256 _amount) external onlyOwner {
68:         require(protocolRewardReceiver != address(0), "ProtocolReward: no receiver");
```

```
69:         rewardToken.safeTransfer(protocolRewardReceiver, _amount);
70:         emit TokenWithdrawnOwner(protocolRewardReceiver, _amount);
71:     }
72:
73:     function pause() external onlyOwner whenNotPaused {
74:         _pause();
75:     }
76:
77:     function unpause() external onlyOwner whenPaused {
78:         _unpause();
79:     }
80:
81:}
```

```
12:contract MarginFeeManager is OwnerGovernableUpgradeable {
13:     address private perpTrade;
14:     IKiloStorage private kiloStorage;
15:     address private productManager;
16:
17:     uint256 private constant BASE = 10 ** 8;
18:     uint256 private constant FEE_BASE = 10**12;
19:
20:     uint256 public maxBorrowingRate;
21:     uint256 public minBorrowingRate;
22:     mapping(uint256 => uint256) public cumulativeBorrowings;
23:     mapping(uint256 => uint256) public borrowFeeLastUpdateTimes;
24:
25:     uint256 public maxFundingRate;
26:     uint256 public minFundingMultiplier;
27:     mapping(uint256 => uint256) public fundingMultipliers;
28:     mapping(uint256 => int256) private cumulativeFundings;
29:     mapping(uint256 => uint256) private lastUpdateTimes;
30:
31:     uint256[50] private __gap;
32:
33:     event BorrowingUpdated(uint256 productId, uint256 borrowFeeRate, uint256 borrowFeeRateChange,
uint256 cumulativeBorrowing);
34:     event OwnerSetPerpTrade(address perpTrade);
35:     event OwnerSetMaxBorrowingRate(uint256 maxBorrowingRate);
36:     event OwnerSetMinBorrowingRate(uint256 minBorrowingRate);
37:     event FundingUpdated(uint256 productId, int256 fundingRate, int256 fundingChange, int256 cumu
lativeFunding);
38:     event OwnerSetMinFundingMultiplier(uint256 minFundingMultiplier);
```

```
39:     event OwnerSetFundingMultiplier(uint256 productId, uint256 fundingMultiplier);
40:     event OwnerSetMaxFundingRate(uint256 maxFundingRate);
41:
42:     function initialize(
43:         address _productManager,
44:         address _kiloStorageAddr
45:     ) public initializer {
46:         __owner_governable_init();
47:         owner = msg.sender;
48:         maxBorrowingRate = 10 * FEE_BASE;//10=1000%
49:         minBorrowingRate = FEE_BASE / 10;//0.01=1%
50:         maxFundingRate = 10 * FEE_BASE;
51:         minFundingMultiplier = 2 * FEE_BASE;
52:         productManager = _productManager;
53:         kiloStorage = IKiloStorage(_kiloStorageAddr);
54:     }
55:
56:     function updateMarginFee(uint256 _productId) external returns(int256, uint256){
57:         require(msg.sender == perpTrade, "FundingManager: not allowed");
58:         if (lastUpdateTimes[_productId] == 0 && borrowFeeLastUpdateTimes[_productId] == 0) {
59:             lastUpdateTimes[_productId] = block.timestamp;
60:             borrowFeeLastUpdateTimes[_productId] = block.timestamp;
61:             return (0,0);
62:         }
63:         int256 fundingRate = getFundingRate(_productId);
64:         int256 fundingChange = fundingRate * int256(block.timestamp - lastUpdateTimes[_productI
d]) / int256(365 days);
65:         int256 cumulativeFunding = cumulativeFundings[_productId] + fundingChange;
66:         cumulativeFundings[_productId] = cumulativeFunding;
67:         lastUpdateTimes[_productId] = block.timestamp;
68:
69:         uint256 borrowFeeRate = getBorrowingRate(_productId);
70:         uint256 borrowFeeRateChange = borrowFeeRate * uint256(block.timestamp - borrowFeeLastUpda
teTimes[_productId]) / uint256(365 days);
71:         uint256 cumulativeBorrowing = cumulativeBorrowings[_productId] + borrowFeeRateChange;
72:         cumulativeBorrowings[_productId] = cumulativeBorrowing;
73:         borrowFeeLastUpdateTimes[_productId] = block.timestamp;
74:
75:         emit FundingUpdated(_productId, fundingRate, fundingChange, cumulativeFunding);
76:         emit BorrowingUpdated(_productId, borrowFeeRate, borrowFeeRateChange, cumulati-
```

```
veBorrowing);
77:         return (cumulativeFunding, cumulativeBorrowing);
78:     }
79:
80:     function getBorrowingRate(uint256 _productId) public view returns (uint256) {
81:         IProductManager.Product memory product = IProductManager(productManager).getProduct(_prod
uctId);
82:         IKiloStorage.KiloConfig memory kiloConfig = kiloStorage.getKiloConfig();
83:         uint256 maxExposure = IPerpTrade(perpTrade).getMaxExposure(product.weight);
84:         (uint256 openInterestLong, uint256 openInterestShort) = kiloStorage.openInterest(_product
Id);
85:         uint256 borrowFeeRate = (openInterestLong + openInterestShort) * BASE / (kiloConfig.maxEx
posureMultiplier * maxExposure) * product.borrowingFactor * FEE_BASE / BASE / (10 ** 4);
86:         borrowFeeRate = Math.min(borrowFeeRate, maxBorrowingRate);
87:         borrowFeeRate = Math.max(borrowFeeRate, minBorrowingRate);
88:         return borrowFeeRate;
89:     }
90:
91:     function getBorrowing(uint256 _productId) external view returns (uint256) {
92:         return cumulativeBorrowings[_productId];
93:     }
94:
95:     function getCurrentCumulativeBorrowing(uint256 _productId) external view returns(uint256 borr
owingRate, uint256 cumulativeBorrowing) {
96:         borrowingRate = getBorrowingRate(_productId);
97:         uint256 borrowFeeRateChange = borrowingRate * uint256(block.timestamp - lastUpdateTimes[_
productId]) / uint256(365 days);
98:         cumulativeBorrowing = cumulativeBorrowings[_productId] + borrowFeeRateChange;
99:     }
100:
101:     function getFundingRate(uint256 _productId) public view returns(int256) {
102:         IProductManager.Product memory product = IProductManager(productManager).getProduct(_pro
ductId);
103:         uint256 maxExposure = IPerpTrade(perpTrade).getMaxExposure(product.weight);
104:         uint256 fundingMultiplier = Math.max(fundingMultipliers[_productId], minFundingMultiplie
r);
105:         (uint256 openInterestLong, uint256 openInterestShort) = kiloStorage.openInterest(_produc
tId);
106:         if (openInterestLong > openInterestShort) {
107:             return int256(Math.min((openInterestLong - openInterestShort) * fundingMultiplier /
maxExposure, maxFundingRate));
108:         } else {
```

```
109:             return -1 * int256(Math.min((openInterestShort - openInterestLong) * fundingMultipli
er / maxExposure, maxFundingRate));
110:         }
111:     }
112:
113:     function getFunding(uint256 _productId) external view returns(int256) {
114:         return cumulativeFundings[_productId];
115:     }
116:
117:     function getCurrentCumulativeFunding(uint256 _productId) external view returns(int256 fundin
gRate, int256 cumulativeFunding) {
118:         fundingRate = getFundingRate(_productId);
119:         int256 fundingChange = fundingRate * int256(block.timestamp - lastUpdateTimes[_productI
d]) / int256(365 days);
120:         cumulativeFunding = cumulativeFundings[_productId] + fundingChange;
121:     }
122:
123:     function setPerpTrade(address _perpTrade) external onlyOwner {
124:         perpTrade = _perpTrade;
125:         emit OwnerSetPerpTrade(_perpTrade);
126:     }
127:
128:     function setMaxBorrowingRate(uint256 _maxBorrowingRate) external onlyOwner {
129:         maxBorrowingRate = _maxBorrowingRate;
130:         emit OwnerSetMaxBorrowingRate(_maxBorrowingRate);
131:     }
132:
133:     function setMinBorrowingRate(uint256 _minBorrowingRate) external onlyOwner {
134:         minBorrowingRate = _minBorrowingRate;
135:         emit OwnerSetMinBorrowingRate(_minBorrowingRate);
136:     }
137:
138:     function setMinFundingMultiplier(uint256 _minFundingMultiplier) external onlyOwner {
139:         minFundingMultiplier = _minFundingMultiplier;
140:         emit OwnerSetMinFundingMultiplier(_minFundingMultiplier);
141:     }
142:
143:     function setFundingMultiplier(uint256 _productId, uint256 _fundingMultiplier) external onlyO
wner {
144:         fundingMultipliers[_productId] = _fundingMultiplier;
145:         emit OwnerSetFundingMultiplier(_productId, _fundingMultiplier);
146:     }
```

```
147:
148:     function setMaxFundingRate(uint256 _maxFundingRate) external onlyOwner {
149:         maxFundingRate = _maxFundingRate;
150:         emit OwnerSetMaxFundingRate(_maxFundingRate);
151:     }
152:
153:     function getFundingBorrowing(uint256 _productId) external view returns(int256 funding, uint2
56 borrowing) {
154:         funding = cumulativeFundings[_productId];
155:         borrowing = cumulativeBorrowings[_productId];
156:     }
157:}
```

Description

seek.guo : MarginFeeManager should inherit from IMarginFeeManager ProtocolReward should inherit from IProtocolReward

Recommendation

seek.guo : Fix code as follow for MarginFeeManager

```
import "../interfaces/IMarginFeeManager.sol"

contract MarginFeeManager is OwnerGovernableUpgradeable, IMarginFeeManager {
  ...
}
```

Fix code as follow for MarginFeeManager

```
import "./interfaces/IProtocolReward.sol";

contract ProtocolReward is OwnerGovernableUpgradeable, PausableUpgradeable, ReentrancyGuardUpgradeab
le, IProtocolReward {
.....
}
```

Client Response

Fixed

KEX-38:Missing calls to __ReentrancyGuard_init() function in inherited contracts

Category	Severity	Status	Contributor
Logical	Low	Fixed	Hacker007

Code Reference

- code/src/core/PendingReward.sol#L44-L52
- code/src/core/PositionRouter.sol#L178-L184

```
44:    function initialize (
45:        address _kiloStorageAddr
46:    ) public initializer {
47:        __owner_governable_init();
48:        protocolRewardRatio = 5000;
49:        kiloStorage = IKiloStorage(_kiloStorageAddr);
50:        token = kiloStorage.token();
51:        tokenBase = kiloStorage.tokenBase();
52:    }

178:     function initialize(
179:         address _collateralToken,
180:         uint256 _minExecutionFee,
181:         address _kiloStorageAddr
182:     ) public initializer {
183:         __owner_governable_init();
184:
```

Description

Hacker007 : Most contracts use the delegateCall proxy pattern and hence their implementations require the use of initialize() functions instead of constructors. This requires derived contracts to call the corresponding init functions of their inherited base contracts. However, the derived contracts `PendingReward` and `PositionRouter` do not call `__ReentrancyGuard_init()` function in the `initialize` function, which may lead to undefined behavior.

Recommendation

Hacker007 : Add missing calls to init functions of inherited contracts `PendingReward` and `PositionRouter`. An Example.

```
    function initialize (
        address _kiloStorageAddr
    ) public initializer {
        __owner_governable_init();
        __ReentrancyGuard_init();
        //...
    }
```

Client Response

Fixed

KEX-39:Missing checks for zero address in `Governable.sol`、`GovernableUpgradeable.sol` and `OwnerUpgradeable.sol`

Category	Severity	Status	Contributor
Logical	Low	Fixed	Yaodao

Code Reference

- code/src/access/Governable.sol#L20-L22
- code/src/access/GovernableUpgradeable.sol#L24-L26
- code/src/access/OwnerUpgradeable.sol#L24-L27

```
20:     function setGov(address _gov) external onlyGov {
21:         gov = _gov;
22:     }

24:     function setGov(address _gov) external onlyGov {
25:         gov = _gov;
26:     }

24:     function setOwner(address _owner) external onlyOwner {
25:         owner = _owner;
26:         emit SetOwner(_owner);
27:     }
```

Description

Yaodao : These privileged functions lack checks for `address(0)`. If the given parameter is address(0), it will cause the owner/gov address to be `address(0)` and can't gain the ownmr/gov role again.

Recommendation

Yaodao : Recommend adding the check of `address(0)`.

Client Response

Fixed

KEX-40:Missing checks for zero address in `PendingReward.sol`

Category	Severity	Status	Contributor
Code Style	Low	Fixed	0xzoobi

Code Reference

- code/src/core/PendingReward.sol#L58-L67

```
58:     function setRewardReceivers(
59:         address _protocolRewardReceiver,
60:         address _vaultRewardReceiver,
61:         address _liquidationRewardReceiver
62:     ) external onlyOwner {
63:         protocolRewardReceiver = _protocolRewardReceiver;
64:         vaultRewardReceiver = _vaultRewardReceiver;
65:         liquidationRewardReceiver = _liquidationRewardReceiver;
66:         emit OwnerSetRewardReceivers(protocolRewardReceiver, vaultRewardReceiver, liquidationRewa
rdReceiver);
67:     }
```

Description

0xzoobi : Checking addresses against zero-address during initialization or during setting is a security best-practice. However, such checks are missing in address variable initializations/changes in many places.

Impact: Allowing zero-addresses will lead to contract reverts and force redeployments if there are no setters for such address variables.

Recommendation

0xzoobi : add a check for address(0) as shown below.

Example Fix

```
require(protocolRewardReceiver != address(0), "Cannot be zero address");
```

Client Response

Fixed

KEX-41:Missing checks for zero address in `ProtocolReward.sol`

Category	Severity	Status	Contributor
Logical	Low	Fixed	Hacker007

Code Reference

- code/src/tradereward/ProtocolReward.sol#L30-L38

```
30:         rewardToken = IERC20Upgradeable(_rewardToken);
31:         pendingReward = _pendingReward;
32:         protocolRewardReceiver = _protocolRewardReceiver;
33:     }
34:
35:
36:     function setXkiloToken(address _xkiloToken) external onlyOwner {
37:         xkiloToken = IERC20Upgradeable(_xkiloToken);
38:     }
```

Description

Hacker007 : The functions `initialize()` and `setXkiloToken()` from contract `ProtocolReward` does not check if the input addresses are zero.

```
    function initialize(
        address _rewardToken, address _pendingReward, address _protocolRewardReceiver
    ) public initializer {
        __owner_governable_init();
        __ReentrancyGuard_init();
        __Pausable_init();
        rewardToken = IERC20Upgradeable(_rewardToken);
        pendingReward = _pendingReward;
        protocolRewardReceiver = _protocolRewardReceiver;
    }
    function setXkiloToken(address _xkiloToken) external onlyOwner {
        xkiloToken = IERC20Upgradeable(_xkiloToken);
    }
```

A zero address may induce transaction failure or get reward tokens lost forever.

Recommendation

Hacker007 : Check if the input parameter is zero address.

Client Response

Fixed

KEX-42:Missing input validation in `ProductManager`::`setLeverage()`

Category	Severity	Status	Contributor
Logical	Low	Fixed	Hacker007

Code Reference

- code/src/core/ProductManager.sol#L105-L110

```
105:    function setLeverage(uint256 id, uint256 _maxLeverage, uint256 _minLeverage) external onlyOw
ner {
106:        address token = productToken[id];
107:        maxLeverage[token] = _maxLeverage;
108:        minLeverage[token] = _minLeverage;
109:        emit OwnerSetLeverage(id, token, _maxLeverage, _minLeverage);
110:    }
```

Description

Hacker007 : The functions `addProduct()` and `updateProduct()` ensure that `_product.minLeverage` is smaller than `_product.maxLeverage` and `_product.minLeverage` is greater than or equal to 1*BASE. However, the function `setLeverage` does not check if the token corresponding to the input id is zero address and `_maxLeverage` is greater than `_minLeverage`, which may lead to unexpected errors.

```
    function setLeverage(uint256 id, uint256 _maxLeverage, uint256 _minLeverage) external onlyOwner
{
        address token = productToken[id];
        maxLeverage[token] = _maxLeverage;
        minLeverage[token] = _minLeverage;
        emit OwnerSetLeverage(id, token, _maxLeverage, _minLeverage);
    }
```

Recommendation

Hacker007 : Add proper checks to ensure the input parameters are legitimate.

Client Response

Fixed, same as kex-81

KEX-43:Ownable: Does not implement 2-Step-Process for transferring ownership

Category	Severity	Status	Contributor
Code Style	Low	Fixed	xfu, 0xzoobi, ginlee

Code Reference

- code/src/access/Governable.sol#L1-L6
- code/src/access/GovernableUpgradeable.sol#L1-L6
- code/src/access/OperatorOwnerGovernable.sol#L1-L6
- code/src/access/OperatorOwnerGovernableUpgradeable.sol#L1-L6
- code/src/access/OwnerGovernable.sol#L1-L6
- code/src/access/OwnerGovernableUpgradeable.sol#L1-L6
- code/src/access/OwnerUpgradeable.sol#L1-L6
- code/src/access/Governable.sol#L20-L22
- code/src/access/GovernableUpgradeable.sol#L24-L26
- code/src/access/OwnerUpgradeable.sol#L24-L27
- code/src/access/OwnerGovernable.sol#L27-L37
- code/src/access/OwnerGovernableUpgradeable.sol#L31-L41
- code/src/access/OperatorOwnerGovernable.sol#L35-L51
- code/src/access/OperatorOwnerGovernableUpgradeable.sol#L38-L54

```
1:// SPDX-License-Identifier: MIT
2:
3:pragma solidity 0.8.17;
4:
5:contract Governable {
6:    address public gov;
```

```
1:// SPDX-License-Identifier: MIT
2:
3:import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
4:
5:pragma solidity 0.8.17;
6:
```

```
1:// SPDX-License-Identifier: MIT
2:pragma solidity 0.8.17;
3:
4:
5:contract OperatorOwnerGovernable {
6:
```

```
1:// SPDX-License-Identifier: MIT
2:pragma solidity 0.8.17;
3:
4:import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
5:
6:contract OperatorOwnerGovernableUpgradeable is Initializable {
```

```
1:// SPDX-License-Identifier: MIT
2:pragma solidity 0.8.17;
3:import "hardhat/console.sol";
4:
5:contract OwnerGovernable {
6:    address public gov;
```

```
1:// SPDX-License-Identifier: MIT
2:pragma solidity 0.8.17;
3:
4:import "@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol";
5:
6:contract OwnerGovernableUpgradeable is Initializable {
```

```
1:// SPDX-License-Identifier: MIT
2:pragma solidity 0.8.17;
3:
4:import "./GovernableUpgradeable.sol";
5:
6:contract OwnerUpgradeable is Initializable{

20:     function setGov(address _gov) external onlyGov {
21:         gov = _gov;
22:     }

24:     function setGov(address _gov) external onlyGov {
25:         gov = _gov;
26:     }

24:     function setOwner(address _owner) external onlyOwner {
25:         owner = _owner;
26:         emit SetOwner(_owner);
27:     }

27:     function setGov(address _gov) external onlyGov {
28:         require(_gov != address(0), "OwnerGovernable: zero address");
29:         gov = _gov;
30:         emit SetGov(_gov);
31:     }
32:
33:     function setOwner(address _owner) external onlyGov {
34:         require(_owner != address(0), "OwnerGovernable: zero address");
35:         owner = _owner;
36:         emit SetOwner(_owner);
37:     }

31:     function setGov(address _gov) external onlyGov {
32:         require(_gov != address(0), "OwnerGovernable: zero address");
33:         gov = _gov;
34:         emit SetGov(_gov);
35:     }
36:
37:     function setOwner(address _owner) external onlyGov {
38:         require(_owner != address(0), "OwnerGovernable: zero address");
39:         owner = _owner;
40:         emit SetOwner(_owner);
41:     }
```

```
35:     function setGov(address _gov) external onlyGov {
36:         require(_gov != address(0), "OwnerGovernable: zero address");
37:         gov = _gov;
38:         emit SetGov(_gov);
39:     }
40:
41:     function setOwner(address _owner) external onlyGov {
42:         require(_owner != address(0), "OwnerGovernable: zero address");
43:         owner = _owner;
44:         emit SetOwner(_owner);
45:     }
46:
47:     function setOperator(address _operator, bool _isOperator) external onlyOwner {
48:         require(_operator != address(0), "OwnerGovernable: zero address");
49:         operators[_operator] = _isOperator;
50:         emit OwnerSetOperator(_operator, _isOperator);
51:     }
```

```
38:     function setGov(address _gov) external onlyGov {
39:         require(_gov != address(0), "OwnerGovernable: zero address");
40:         gov = _gov;
41:         emit SetGov(_gov);
42:     }
43:
44:     function setOwner(address _owner) external onlyGov {
45:         require(_owner != address(0), "OwnerGovernable: zero address");
46:         owner = _owner;
47:         emit SetOwner(_owner);
48:     }
49:
50:     function setOperator(address _operator, bool _isOperator) external onlyOwner {
51:         require(_operator != address(0), "OwnerGovernable: zero address");
52:         operators[_operator] = _isOperator;
53:         emit OwnerSetOperator(_operator, _isOperator);
54:     }
```

Description

xfu : In multiple contracts, roles can be transferred to another address simply by calling the corresponding `*.setGov()` function. These functions immediately transfer a high-level privilege to a new address in a single transaction, which can

be risky from a security perspective, as providing a faulty address may lockout that role from future calls, rendering the contract potentially useless.

A more secure pattern for such privilege transfers requires the new pending addresses to issue an `acceptAdmin()` function call before finalizing the transfer. Note that this pattern is common even with the use of timelocks, such as the Compound Timelock contract (see functions `setPendingAdmin()` and `acceptAdmin()`).

0xzoobi : The contracts under `src/access` does not implement a 2-Step-Process for transferring ownership. So ownership of the contract can easily be lost when making a mistake when transferring ownership.

Since the privileged roles have critical function roles assigned to them. Assigning the ownership to a wrong user can be disastrous. So Consider using the `Ownable2Step` contract from OZ (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol) instead.

The way it works is there is a `transferOwnership` to transfer the ownership and `acceptOwnership` to accept the ownership. Refer the above Ownable2Step.sol for more details.

ginlee : a general issue spotted in access file, consider implementing a Two-Step Transfer pattern when transferring critical role in the protocol. Otherwise it's possible that the role mistakenly transfers ownership to the wrong address, resulting in a loss of the role

Recommendation

xfu : Require the pending new role address to make an `acceptAdmin()` function call before fully transferring the privilege.

0xzoobi : Implement 2-Step-Process for transferring ownership.

ginlee : Use transferGov and acceptGov function to manage transfer

Client Response

Fixed

KEX-44:Positions cannot be executed by external user when `allowPublicKeeper` is True

Category	Severity	Status	Contributor
Logical	Low	Fixed	xfu

Code Reference

- code/src/core/PositionRouter.sol#L173
- code/src/core/PositionRouter.sol#L574

```
173:    modifier onlyPositionKeeper() {

574:        bool isKeeperCall = msg.sender == address(this) || isPositionKeeper[msg.sender];
```

Description

xfu : - core/PositionRouter.sol

if state `allowPublicKeeper` set to True, it is possible to use `executeNPositionsWithPrices` to execute postion. when executing `executeIncreasePositions` or `executeDecreasePositions` functions, the position is validated using `_validateExecution()` function. This function will always revert if the external user not in `isPositionKeeper` (`bool isKeeperCall = msg.sender == address(this) || isPositionKeeper[msg.sender];`).

Recommendation

xfu : Using allowPublicKeeper as condition to `modifier onlyPositionKeeper()`

- core/PositionRouter.sol:173

```
modifier onlyPositionKeeper() {
        require(isPositionKeeper[msg.sender] || allowPublicKeeper, "not allowed");
        _;
}
```

Remove `isKeeperCall` as condition to revert

- core/PositionRouter.sol:574

```
bool isKeeperCall = msg.sender == address(this) || isPositionKeeper[msg.sender];
if (!isUserExecuteEnabled) {
    revert("PositionRouter: forbidden");
}
```

Client Response

Fixed

KEX-45:Potential balance over cap possibility

Category	Severity	Status	Contributor
Logical	Low	Fixed	Yaodao

Code Reference

- code/src/core/VaultStakeReward.sol#L83-L86
- code/src/core/VaultStakeReward.sol#L88-L104
- code/src/core/PendingReward.sol#L101-L110

```
83:     function _claimVaultPendingReward() private {
84:         IPendingReward(pendingReward).withdrawVaultReward();
85:         IPendingReward(pendingReward).distributePendingPnl();
86:     }

88:     function deposit(uint256 amount, address user) public override nonReentrant returns (uint256)
{
89:         require(kiloConfig.canUserStake(), "Vault: not allowed");
90:         require(msg.sender == user, "Vault: not staker");
91:         require(IERC20Upgradeable(token).balanceOf(address(this)) * BASE / tokenBase + amount <=
cap, "over cap");
92:         _claimVaultPendingReward();
93:
94:         Stake storage _stake = stakes[user];
95:         uint256 newAmount = amount * tokenBase / BASE;
96:         _stake.owner = user;
97:         _stake.timestamp = uint128(block.timestamp);
98:         _stake.amount += uint96(newAmount);
99:         staked += uint256(newAmount);
100:         uint256 shares = previewDeposit(newAmount);
101:         _deposit(user, user, newAmount, shares);
102:         emit StakedRedeemItem(user, StakeType.Stake, int256(newAmount), int256(shares), _stake.a
mount, IERC20Upgradeable(address(this)).balanceOf(user), 0);
103:         return shares * BASE / tokenBase;
104:     }

101:     function withdrawVaultReward() external nonReentrant returns (uint256) {
102:         require(msg.sender == vaultRewardReceiver, "PendingReward: not allowed");
103:         uint256 _pendingReward = rewardReserves[RewardReserveType.VAULT] * tokenBase / BASE;
104:         if (_pendingReward > 0) {
105:             kiloStorage.transferPendingReward(vaultRewardReceiver, _pendingReward);
106:             rewardReserves[RewardReserveType.VAULT] = 0;
107:             emit WithdrawReward(RewardReserveType.VAULT, vaultRewardReceiver, _pendingReward);
108:         }
109:         return _pendingReward;
110:     }
```

Description

Yaodao : According to the following codes, the check of cap will check the sum of balance and amount whether over the `cap`.

```
    function deposit(uint256 amount, address user) public override nonReentrant returns (uint256) {
        require(kiloConfig.canUserStake(), "Vault: not allowed");
        require(msg.sender == user, "Vault: not staker");
        require(IERC20Upgradeable(token).balanceOf(address(this)) * BASE / tokenBase + amount <= ca
p, "over cap");
        _claimVaultPendingReward();

        Stake storage _stake = stakes[user];
        uint256 newAmount = amount * tokenBase / BASE;
        _stake.owner = user;
        _stake.timestamp = uint128(block.timestamp);
        _stake.amount += uint96(newAmount);
        staked += uint256(newAmount);
        uint256 shares = previewDeposit(newAmount);
        _deposit(user, user, newAmount, shares);
        emit StakedRedeemItem(user, StakeType.Stake, int256(newAmount), int256(shares), _stake.amoun
t, IERC20Upgradeable(address(this)).balanceOf(user), 0);
        return shares * BASE / tokenBase;
    }
```

However, the function `_claimVaultPendingReward()` will be called after the check and the function will transfer tokens to `address(this)`.

```
    function _claimVaultPendingReward() private {
        IPendingReward(pendingReward).withdrawVaultReward();
        IPendingReward(pendingReward).distributePendingPnl();
    }
```

```
    function withdrawVaultReward() external nonReentrant returns (uint256) {
        require(msg.sender == vaultRewardReceiver, "PendingReward: not allowed");
        uint256 _pendingReward = rewardReserves[RewardReserveType.VAULT] * tokenBase / BASE;
        if (_pendingReward > 0) {
            kiloStorage.transferPendingReward(vaultRewardReceiver, _pendingReward);
            rewardReserves[RewardReserveType.VAULT] = 0;
            emit WithdrawReward(RewardReserveType.VAULT, vaultRewardReceiver, _pendingReward);
        }
        return _pendingReward;
    }
```

As a result, when `IERC20Upgradeable(token).balanceOf(address(this)) * BASE / tokenBase + amount == cap`, the call of `_claimVaultPendingReward()` will make the `IERC20Upgradeable(token).balanceOf(address(this))` over the cap after this call.

Recommendation

Yaodao : Recommend calling the function `_claimVaultPendingReward()` before the check of `cap`.

Client Response

Fixed

KEX-46:Potential denial-of-service issues caused by accuracy conversion processes.

Category	Severity	Status	Contributor
Logical	Low	Acknowledged	LiRiu

Code Reference

- code/src/core/VaultStakeReward.sol#L88-L104

```
88:    function deposit(uint256 amount, address user) public override nonReentrant returns (uint256)
{
89:        require(kiloConfig.canUserStake(), "Vault: not allowed");
90:        require(msg.sender == user, "Vault: not staker");
91:        require(IERC20Upgradeable(token).balanceOf(address(this)) * BASE / tokenBase + amount <=
cap, "over cap");
92:        _claimVaultPendingReward();
93:
94:        Stake storage _stake = stakes[user];
95:        uint256 newAmount = amount * tokenBase / BASE;
96:        _stake.owner = user;
97:        _stake.timestamp = uint128(block.timestamp);
98:        _stake.amount += uint96(newAmount);
99:        staked += uint256(newAmount);
100:        uint256 shares = previewDeposit(newAmount);
101:        _deposit(user, user, newAmount, shares);
102:        emit StakedRedeemItem(user, StakeType.Stake, int256(newAmount), int256(shares), _stake.a
mount, IERC20Upgradeable(address(this)).balanceOf(user), 0);
103:        return shares * BASE / tokenBase;
104:    }
```

Description

LiRiu : There might be a situation at line 100 of the code where the shares calculated based on "newAmount" could be lower than "newAmount" itself due to "totalSupply() + 10 ** _decimalsOffset()" being less than "totalAssets() + 1" during the actual calculation. For example, there could be a situation where "share" is less than "amount * tokenBase / BASE." Therefore, while the calculation of "shares * BASE / tokenBase" would result in 0, "share" itself is not necessarily 0.

```
    function deposit(uint256 amount, address user) public override nonReentrant returns (uint256) {
        ...
        Stake storage _stake = stakes[user];
        uint256 newAmount = amount * tokenBase / BASE;
        _stake.owner = user;
        _stake.timestamp = uint128(block.timestamp);
        _stake.amount += uint96(newAmount);
        staked += uint256(newAmount);
        uint256 shares = previewDeposit(newAmount);
        _deposit(user, user, newAmount, shares);
        ...
    }
```

Recommendation

LiRiu : Please modify the precision transformation according to the business requirements to avoid exceptional situations.

Client Response

Acknowledged

KEX-47:Potential divided by zero error

Category	Severity	Status	Contributor
Logical	Low	Fixed	xfu

Code Reference

- code/src/core/MarginFeeManager.sol#L85

```
85:         uint256 borrowFeeRate = (openInterestLong + openInterestShort) * BASE / (kiloConfig.maxExposureMultiplier * maxExposure) * product.borrowingFactor * FEE_BASE / BASE / (10 ** 4);
```

Description

xfu : - core/MarginFeeManager.sol:85

```
        uint256 borrowFeeRate = (openInterestLong + openInterestShort) * BASE / (kiloConfig.maxExposureMultiplier * maxExposure) * product.borrowingFactor * FEE_BASE / BASE / (10 ** 4);
```

the rre calculation `(openInterestLong + openInterestShort) * BASE / (kiloConfig.maxExposureMultiplier * maxExposure)` may be < 1 and > 0, If this condition holds, the computed value will always be 0.

Recommendation

xfu : multiply before divide:

```
        uint256 borrowFeeRate = (openInterestLong + openInterestShort) * BASE  * product.borrowingFactor * FEE_BASE/ (kiloConfig.maxExposureMultiplier * maxExposure) / BASE / (10 ** 4);
```

Client Response

Fixed

KEX-48:Pull-Over-Push Pattern In `setGov()` and `setOwner()` Functions

Category	Severity	Status	Contributor
Logical	Low	Fixed	Hacker007

Code Reference

- code/src/access/Governable.sol#L20-L22
- code/src/access/GovernableUpgradeable.sol#L24-L26
- code/src/access/OwnerUpgradeable.sol#L24-L27
- code/src/access/OwnerGovernable.sol#L27-L31
- code/src/access/OwnerGovernableUpgradeable.sol#L31-L35
- code/src/access/OperatorOwnerGovernable.sol#L35-L39
- code/src/access/OperatorOwnerGovernableUpgradeable.sol#L38-L42

```
20:     function setGov(address _gov) external onlyGov {
21:         gov = _gov;
22:     }

24:     function setGov(address _gov) external onlyGov {
25:         gov = _gov;
26:     }

24:     function setOwner(address _owner) external onlyOwner {
25:         owner = _owner;
26:         emit SetOwner(_owner);
27:     }

27:     function setGov(address _gov) external onlyGov {
28:         require(_gov != address(0), "OwnerGovernable: zero address");
29:         gov = _gov;
30:         emit SetGov(_gov);
31:     }

31:     function setGov(address _gov) external onlyGov {
32:         require(_gov != address(0), "OwnerGovernable: zero address");
33:         gov = _gov;
34:         emit SetGov(_gov);
35:     }

35:     function setGov(address _gov) external onlyGov {
36:         require(_gov != address(0), "OwnerGovernable: zero address");
37:         gov = _gov;
38:         emit SetGov(_gov);
39:     }

38:     function setGov(address _gov) external onlyGov {
39:         require(_gov != address(0), "OwnerGovernable: zero address");
40:         gov = _gov;
41:         emit SetGov(_gov);
42:     }
```

Description

Hacker007 : The change of gov by function setGov() overrides the previously set gov with the new one without guaranteeing the new _gov is able to actuate transactions on-chain.

```
    function setGov(address _gov) external onlyGov {
        gov = _gov;
    }
```

The change of gov by function setOwner() overrides the previously set owner with the new one without guaranteeing the new _owner is able to actuate transactions on-chain.

```
    function setOwner(address _owner) external onlyOwner {
        owner = _owner;
        emit SetOwner(_owner);
    }
```

Recommendation

Hacker007 : We advise the pull-over-push pattern to be applied here whereby a new gov is first proposed and consequently needs to accept the gov status ensuring that the account can actuate transactions on-chain. An example:

```
address public newGov;

function setGov(address pendingGov) external onlyGov {
    require(pendingGov != address(0), "new gov can not be the zero address.")
    newGov = pendingGov;
}

function acceptGov() external {
    require(msg.sender == newGov, 'You must be nominated as new gov before you can accept gov');
    gov = newGov;
    newGov = address(0);
}
```

The same pull-over-push pattern can be applied to the `setOwner()` function.

Client Response

Fixed

KEX-49:Redundant modifier `payable`

Category	Severity	Status	Contributor
Language Specific	Low	Fixed	Hacker007

Code Reference

- code/src/interfaces/IPerpTrade.sol#L9-L16
- code/src/core/PerpTrade.sol#L118-L125

```
9:    function increasePosition(
10:         address user,
11:         uint256 productId,
12:         uint256 margin,
13:         bool isLong,
14:         uint256 leverage,
15:         uint256 orderType
16:     ) external payable;

118:    function increasePosition(
119:        address user,
120:        uint256 productId,
121:        uint256 margin,
122:        bool isLong,
123:        uint256 leverage,
124:        uint256 orderType
125:    ) public payable nonReentrant {
```

Description

Hacker007 : The function that is declared `payable` can be receiver ether. The function `increasePosition()` seems not designed for receiving ether to have the `payable` modifier.

Recommendation

Hacker007 : Remove the `payable` modifier from the function `increasePosition()`.

Client Response

Fixed

KEX-50:Unbounded loops may cause "OrderBook" contract "cancelMultiple" function to fail

Category	Severity	Status	Contributor
DOS	Low	Fixed	ginlee

Code Reference

- code/src/core/OrderBook.sol#L217-L237
- code/src/core/OrderBook.sol#L239-L249

```
217:     function executeOrders(
218:         address[] memory _openAddresses,
219:         uint256[] memory _increaseOrderIndexes,
220:         address[] memory _closeAddresses,
221:         uint256[] memory _decreaseOrderIndexes,
222:         address payable _feeReceiver
223:     ) public onlyKeeper {
224:         require(_openAddresses.length == _increaseOrderIndexes.length && _closeAddresses.length
== _decreaseOrderIndexes.length, "OrderBook: not same length");
225:         for (uint256 i = 0; i < _openAddresses.length; i++) {
226:             try this.executeIncreaseOrder(_openAddresses[i], _increaseOrderIndexes[i], _feeRecei
ver) {
227:             } catch Error(string memory executionError) {
228:                 emit ExecuteIncreaseOrderError(_openAddresses[i], _increaseOrderIndexes[i], exec
utionError);
229:             } catch (bytes memory /*lowLevelData*/) {}
230:         }
231:         for (uint256 i = 0; i < _closeAddresses.length; i++) {
232:             try this.executeDecreaseOrder(_closeAddresses[i], _decreaseOrderIndexes[i], _feeRece
iver) {
233:             } catch Error(string memory executionError) {
234:                 emit ExecuteDecreaseOrderError(_closeAddresses[i], _decreaseOrderIndexes[i], exe
cutionError);
235:             } catch (bytes memory /*lowLevelData*/) {}
236:         }
237:     }

239:     function cancelMultiple(
240:         uint256[] memory _increaseOrderIndexes,
241:         uint256[] memory _decreaseOrderIndexes
242:     ) external {
243:         for (uint256 i = 0; i < _increaseOrderIndexes.length; i++) {
244:             cancelIncreaseOrder(_increaseOrderIndexes[i]);
245:         }
246:         for (uint256 i = 0; i < _decreaseOrderIndexes.length; i++) {
247:             cancelDecreaseOrder(_decreaseOrderIndexes[i]);
248:         }
249:     }
```

Description

ginlee : There are no upper bounds on the number of _increaseOrderIndexes and _decreaseOrderIndexes, may use up gas in the loop. Same issue as for executeOrders

Recommendation

ginlee : Have an upper bound on

Client Response

Fixed, Already processed, limited by external keeper, up to 20 pieces of data each time

KEX-51:Uncapped Fee `minExecutionFee`

Category	Severity	Status	Contributor
Governance Manipulation	Low	Fixed	xfu

Code Reference

- code/src/core/OrderBook.sol#L189-L192

```
189:    function setMinExecutionFee(uint256 _minExecutionFee) external onlyOwner {
190:        minExecutionFee = _minExecutionFee;
191:        emit OwnerSetMinExecutionFee(_minExecutionFee);
192:    }
```

Description

xfu : - core/OrderBook.sol

Function `OrderBook.setMinExecutionFee()` does not constrain the input. Consequently, users can be subject to very high fees. This can subject users to high execution fees without prior announcements or make the protocol unusable.

Recommendation

xfu : Consider having an upper bound for the execution fee, communicate it via public facing documentation and accordingly check against it in functions `setMinExecutionFee()` and `initialize()` (constructor() PositionRouter.sol).

Client Response

Fixed

KEX-52:Uncapped product `ProductManager.setFee()`

Category	Severity	Status	Contributor
Governance Manipulation	Low	Fixed	xfu

Code Reference

- code/src/core/ProductManager.sol#L99-L103

```
99:    function setFee(uint256 id, uint256 _fee) external onlyOwner {
100:        address token = productToken[id];
101:        fee[token] = _fee;
102:        emit OwnerSetFee(id, token, _fee);
103:    }
```

Description

xfu : - core/ProductManager.sol

Function `ProductManager.setFee()` does not constrain the input `_fee`. Consequently, users can be subject to very high fees (effectively up to 100%, RATE_BASE=10000). This can subject users to high fees with no prior announcements. Note: Only the admin address can execute this function.

Recommendation

xfu : Consider having an upper bound for the deposit fee, communicate it via public-facing documentation, and accordingly check against it in function `setFee()`.

Client Response

Fixed

KEX-53:Unchecked Return Value When Calling `IPendingReward.withdrawProtocolReward()`

Category	Severity	Status	Contributor
Logical	Low	Declined	Hacker007

Code Reference

- code/src/tradereward/ProtocolReward.sol#L40-L44
- code/src/tradereward/ProtocolReward.sol#L53-L55

```
40:     function claimTradeReward(uint256 _amount) external whenNotPaused {
41:         require(msg.sender == tradeRewardDistributor, "ProtocolReward: not allowed");
42:         IPendingReward(pendingReward).withdrawProtocolReward();
43:         rewardToken.safeTransfer(msg.sender, _amount);
44:     }

53:     function claimProtocolReward() external whenNotPaused {
54:         IPendingReward(pendingReward).withdrawProtocolReward();
55:     }
```

Description

Hacker007 : The withdrawProtocolReward is used to transfer pending reward to the `protocolRewardReceiver`.

```
//from code/src/core/PendingReward.sol
    function withdrawProtocolReward() external nonReentrant returns (uint256) {
        require(msg.sender == protocolRewardReceiver, "PendingReward: not allowed");
        uint256 _pendingReward = rewardReserves[RewardReserveType.PROTOCOL] * tokenBase / BASE;
        if (_pendingReward > 0) {
            kiloStorage.transferPendingReward(protocolRewardReceiver, _pendingReward);
            rewardReserves[RewardReserveType.PROTOCOL] = 0;
            emit WithdrawReward(RewardReserveType.PROTOCOL, protocolRewardReceiver, _pendingReward);
        }
        return _pendingReward;
    }
```

The real reward amount `_pendingReward` will be returned. However, In the contract `ProtocolReward`, the following function calling `withdrawProtocolReward()` won't check the return value, which may lead to a failure of token

transfer.

```
    function claimTradeReward(uint256 _amount) external whenNotPaused {
        require(msg.sender == tradeRewardDistributor, "ProtocolReward: not allowed");
        IPendingReward(pendingReward).withdrawProtocolReward();
        rewardToken.safeTransfer(msg.sender, _amount);
    }
    function claimProtocolReward() external whenNotPaused {
        IPendingReward(pendingReward).withdrawProtocolReward();
    }
```

Recommendation

Hacker007 : Check the return value of `withdrawProtocolReward()` before doing a token transfer.

Client Response

Declined, The claimTradeReward function is called by tradeRewardDistributor. Normally, the IPendingReward(pendingReward).withdrawProtocolReward() function should retrieve a reward amount that is greater than _amount, so it should not be possible for the transfer amount to be insufficient. However, if the transfer amount is insufficient, the safeTransfer function will perform a check and cause the transaction to fail, and manual intervention will be required to resolve the issue.

KEX-54:Wrong initial value of `minBorrowingRate`

Category	Severity	Status	Contributor
Logical	Low	Fixed	danielt

Code Reference

- code/src/core/MarginFeeManager.sol#L48-L49

```
48:         maxBorrowingRate = 10 * FEE_BASE;//10=1000%
49:         minBorrowingRate = FEE_BASE / 10;//0.01=1%
```

Description

danielt : In the `MarginFeeManager` contract, the `minBorrowingRate` is initialized to be `FEE_BASE / 10` shown below:

```
    function initialize(
        address _productManager,
        address _kiloStorageAddr
    ) public initializer {
        __owner_governable_init();
        owner = msg.sender;
        maxBorrowingRate = 10 * FEE_BASE;//10=1000%
        minBorrowingRate = FEE_BASE / 10;//0.01=1%
        ...
    }
```

Per the comments of the statements, we knew that `FEE_BASE` stands for `100%`, and the `maxBorrowingRate` is initialized to be `10 * FEE_BASE`, which matches its comment `10=1000%`.

However, the comment `0.01=1%` tells us that the `minBorrowingRate` should be `FEE_BASE / 100` instead of `FEE_BASE / 10`.

The wrong initial value assigned to the `minBorrowingRate` will incur side effects on the on-chain transactions.

Recommendation

danielt : Update the initial value assigned to the `minBorrowingRate`, and make the statement and the comment consistent.

Client Response

Fixed

KEX-55:initialize function can be called by anybody in KiloPriceFeed contract

Category	Severity	Status	Contributor
Governance Manipulation	Low	Mitigated	ginlee

Code Reference

- code/src/core/MarginFeeManager.sol#L42
- code/src/core/PendingReward.sol#L44
- code/src/core/ProductManager.sol#L45
- code/src/core/KiloPriceFeed.sol#L48
- code/src/core/VaultStakeReward.sol#L64
- code/src/core/KiloStorageManager.sol#L85
- code/src/core/PerpTrade.sol#L99
- code/src/core/OrderBook.sol#L162
- code/src/core/PositionRouter.sol#L178

```
42:    function initialize(

44:    function initialize (

45:    function initialize(

48:    function initialize() public initializer {

64:    function initialize(

85:    function initialize(

99:    function initialize(

162:    function initialize(

178:    function initialize(
```

Description

ginlee : initialize() function can be called anybody when the contract is not initialized.More importantly, if someone else runs this function, they will have full authority because of the owner state variable. Also exists in other contracts

Recommendation

ginlee : Add a control that makes init() only call the Deployer Contract or EOA; `if (msg.sender != DEPLOYER_ADDRESS) { revert NotDeployer(); }`

Client Response

Mitigated, The initialize method can only be called once, and our deployment script will execute this method. Otherwise, the contract cannot be used properly, so no modifications will be made for now.

KEX-56:staking amount is unsafely downcasted from uint256 to uint96

Category	Severity	Status	Contributor
Integer Overflow and Underflow	Low	Fixed	ladboy233

Code Reference

- code/src/core/VaultStakeReward.sol#L87-L105

```
87:
88:     function deposit(uint256 amount, address user) public override nonReentrant returns (uint256)
{
89:         require(kiloConfig.canUserStake(), "Vault: not allowed");
90:         require(msg.sender == user, "Vault: not staker");
91:         require(IERC20Upgradeable(token).balanceOf(address(this)) * BASE / tokenBase + amount <=
cap, "over cap");
92:         _claimVaultPendingReward();
93:
94:         Stake storage _stake = stakes[user];
95:         uint256 newAmount = amount * tokenBase / BASE;
96:         _stake.owner = user;
97:         _stake.timestamp = uint128(block.timestamp);
98:         _stake.amount += uint96(newAmount);
99:         staked += uint256(newAmount);
100:         uint256 shares = previewDeposit(newAmount);
101:         _deposit(user, user, newAmount, shares);
102:         emit StakedRedeemItem(user, StakeType.Stake, int256(newAmount), int256(shares), _stake.a
mount, IERC20Upgradeable(address(this)).balanceOf(user), 0);
103:         return shares * BASE / tokenBase;
104:     }
105:
```

Description

ladboy233 : staking amount is unsafely downcasted from uint256 to uint96

In VaultStakeReward.sol, the deposit function can be called

```
    function deposit(uint256 amount, address user) public override nonReentrant returns (uint256) {
        require(kiloConfig.canUserStake(), "Vault: not allowed");
        require(msg.sender == user, "Vault: not staker");
        require(IERC20Upgradeable(token).balanceOf(address(this)) * BASE / tokenBase + amount <= ca
p, "over cap");
        _claimVaultPendingReward();

        Stake storage _stake = stakes[user];
        uint256 newAmount = amount * tokenBase / BASE;
        _stake.owner = user;
        _stake.timestamp = uint128(block.timestamp);
        // @audit safe downcasting
        _stake.amount += uint96(newAmount);
        staked += uint256(newAmount);
        // @audit inflation attack
        // @audit 0 shares
        uint256 shares = previewDeposit(newAmount);
        _deposit(user, user, newAmount, shares);
        emit StakedRedeemItem(user, StakeType.Stake, int256(newAmount), int256(shares), _stake.amoun
t, IERC20Upgradeable(address(this)).balanceOf(user), 0);
        // @audit what are you returning?
        return shares * BASE / tokenBase;
    }
```

as we can see, the amount is uint256 type, but the staked amount can be sliently downcast to uint96

```
  _stake.amount += uint96(newAmount);
```

where new amount is a unt256 type

```
uint256 newAmount = amount * tokenBase / BASE;
```

this means if the staking amount is more than uint96 number, the staking amount is wrongly downcasted and truncated to uint96, leading to loss of staking token

Recommendation

ladboy233 : The recommendation is use openzepplein's safe cast to make sure the cast does not over-flow or under-flow

Client Response

Fixed

KEX-57:Code is never used and should be removed

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	seek.guo

Code Reference

- code/src/access/OwnerGovernable.sol#L3
- code/src/access/GovernableUpgradeable.sol#L15-L17
- code/src/access/OwnerUpgradeable.sol#L15-L17
- code/src/libraries/PerpTradeUtil.sol#L138-L169
- code/src/core/PerpTrade.sol#L368-L382

```
3:import "hardhat/console.sol";

15:    function __governable_init() internal initializer {
16:        gov = msg.sender;
17:    }

15:    function __owner_init() internal initializer {
16:        owner = msg.sender;
17:    }

138:    function _calculatePrice(
139:        address oracle,
140:        address productToken,
141:        bool isLong,
142:        uint256 openInterestLong,
143:        uint256 openInterestShort,
144:        uint256 maxExposure,
145:        uint256 reserve,
146:        uint256 amount,
147:        uint256 maxShift
148:    ) internal view returns (uint256) {
149:        require(amount < reserve, "amount more than reserve");
150:        uint256 oraclePrice = isLong ? IOracle(oracle).getPrice(productToken, true) : IOracle(or
acle).getPrice(productToken, false);
151:        int256 shift = (int256(openInterestLong) - int256(openInterestShort)) * int256(maxShift)
/ int256(maxExposure);
152:        if (isLong) {
153:            uint256 slippage = (reserve * reserve / (reserve - amount) - reserve) * BASE / amoun
t;
154:            // slippage = shift >= 0 ? slippage + uint256(shift) : slippage - (uint256(-1 * shif
t) / shiftDivider);
155:            // shift > 0, When shift is positive, the long position slippage is increased based
on the shift value. This is no longer incentivized.
156:            if (shift > 0) {
157:                slippage = slippage + uint256(shift);
158:            }
159:            return oraclePrice * slippage / BASE;
160:        } else {
161:            uint256 slippage = (reserve - (reserve * reserve) / (reserve + amount)) * BASE / amo
unt;
162:            // slippage = shift >= 0 ? slippage + (uint256(shift) / shiftDivider) : slippage - u
int256(-1 * shift);
```

```
163:            // shift < 0, When shift is negative, the short position slippage is increased based
on the shift value. This is no longer incentivized.
164:            if (shift < 0) {
165:                slippage = slippage - uint256(-1 * shift);
166:            }
167:            return oraclePrice * slippage / BASE;
168:        }
169:    }

368:    function _updateOpenInterest(
369:        uint256 productId,
370:        uint256 addedOpenInterest,
371:        bool isLong,
372:        bool isIncrease,
373:        uint256 productWeight,
374:        IKiloStorage.KiloConfig memory kiloConfig
375:    ) private {
376:        if (isIncrease) {
377:            uint256 maxExposure = _getMaxExposure(productWeight, kiloConfig.exposureMultiplier);
378:            kiloStorage.updateIncreaseOpenInterest(productId, addedOpenInterest, isLong, _getVau
ltBalance(), maxExposure);
379:        } else {
380:            kiloStorage.updateDecreaseOpenInterest(productId, addedOpenInterest, isLong);
381:        }
382:    }
```

Description

seek.guo : Unuseless code `import "hardhat/console.sol"` in OwnerGovernable.sol
seek.guo : Code is never used

Recommendation

seek.guo : remove this code

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;
import "hardhat/console.sol";
```

seek.guo : Should be removed

Client Response

Fixed

KEX-58:Events not emitted or indexed for important state changes

Category	Severity	Status	Contributor
Language Specific	**Informational**	**Mitigated**	**danielt, 0xzoobi, Yaodao, ginlee**

Code Reference

- code/src/tradereward/ProtocolReward.sol#L20-L22
- code/src/access/Governable.sol#L20-L22
- code/src/access/GovernableUpgradeable.sol#L24-L26
- code/src/referrals/ReferralStorageManager.sol#L28-L36
- code/src/core/ProductManager.sol#L30-L42
- code/src/core/KiloPriceFeed.sol#L32-L45
- code/src/core/MarginFeeManager.sol#L33-L40
- code/src/tradereward/ProtocolReward.sol#L36-L38
- code/src/core/VaultStakeReward.sol#L43
- code/src/core/KiloStorageManager.sol#L49-L83
- code/src/core/PendingReward.sol#L54-L56
- code/src/core/VaultStakeReward.sol#L55-L61
- code/src/tradereward/TradeRewardDistributor.sol#L58-L64
- code/src/core/PerpTrade.sol#L114-L116
- code/src/core/KiloPriceFeed.sol#L147-L156
- code/src/core/OrderBook.sol#L180-L182
- code/src/core/ProductManager.sol#L181-L183
- code/src/core/PositionRouter.sol#L199-L201
- code/src/core/PerpTrade.sol#L448-L462

```
20:     function setGov(address _gov) external onlyGov {
21:         gov = _gov;
22:     }

20:     event TokenWithdrawnOwner(address to, uint256 amount);
21:     event OwnerSetProtocolRewardReceiver(address protocolRewardReceiver);
22:     event OwnerSetTradeRewardDistributor(address tradeRewardDistributor);

24:     function setGov(address _gov) external onlyGov {
25:         gov = _gov;
26:     }

28:     event SetTraderReferralCode(address account, bytes32 code);
29:     event SetTier(uint256 tierId, uint256 totalRebate, uint256 discountShare);
30:     event SetReferrerTier(address referrer, uint256 tierId);
31:     event SetReferrerDiscountShare(address referrer, uint256 discountShare);
32:     event RegisterCode(address account, bytes32 code);
33:     event SetCodeOwner(address account, address newAccount, bytes32 code);
34:     event GovSetCodeOwner(bytes32 code, address newAccount);
35:
36:     event OwnerSetDiscountShareLimiter(uint256 minDiscountShares, uint256 maxDiscountShares);

30:     event ProductAdded(uint256 productId, address token, Product product);
31:     event ProductUpdated(uint256 productId, address token, Product product);
32:     event OwnerSetBorrowingFactor(uint256 productId, address token, uint256 borrowingFactor);
33:     event OwnerSetMaxPositionSize(uint256 productId, address token, uint256 maxPositionSize);
34:     event OwnerSetFee(uint256 productId, address token, uint256 fee);
35:     event OwnerSetLeverage(uint256 productId, address token, uint256 maxLeverage, uint256 minLeve
rage);
36:     event OwnerSetIsActive(uint256 productId, address token, bool isActive);
37:     event OwnerSetMinPriceChange(uint256 productId, address token, uint256 minPriceChange);
38:     event OwnerSetWeight(uint256 productId, address token, uint256 weight);
39:     event OwnerSetReserve(uint256 productId, address token, uint256 reserve);
40:     error ProductNotExisted(address productId);
41:     event OwnerEnableFeeDiscount(bool discountEnabled);
42:     event OwnerSetAccountDiscount(address account, uint256 discount);

32:     event PriceSet(address token, uint256 price, uint256 timestamp);
```

```
33:
34:     event OwnerSetPriceDuration(uint256 priceDuration);
35:     event OwnerSetUpdateInterval(uint256 updateInterval);
36:     event OwnerSetDefaultMaxPriceDiff(uint256 maxPriceDiff);
37:     event OwnerSetMaxPriceDiff(address token, uint256 maxPriceDiff);
38:     event OwnerSetKeeper(address keeper, bool isActive);
39:
40:     event OwnerSetSpreadEnabled(bool isSpreadEnabled);
41:     event OwnerSetDefaultSpread(uint256 defaultSpread);
42:     event OwnerSetSpread(address token, uint256 spread);
43:     event OwnerSetIsChainlinkOnly(bool isChainlinkOnlySet);
44:     event OwnerSetIsKiloOracleOnly(bool isKiloOracleOnlySet);
45:     event OwnerEnableFastOracle(bool enabled);

33:     event BorrowingUpdated(uint256 productId, uint256 borrowFeeRate, uint256 borrowFeeRateChange,
uint256 cumulativeBorrowing);
34:     event OwnerSetPerpTrade(address perpTrade);
35:     event OwnerSetMaxBorrowingRate(uint256 maxBorrowingRate);
36:     event OwnerSetMinBorrowingRate(uint256 minBorrowingRate);
37:     event FundingUpdated(uint256 productId, int256 fundingRate, int256 fundingChange, int256 cumu
lativeFunding);
38:     event OwnerSetMinFundingMultiplier(uint256 minFundingMultiplier);
39:     event OwnerSetFundingMultiplier(uint256 productId, uint256 fundingMultiplier);
40:     event OwnerSetMaxFundingRate(uint256 maxFundingRate);

36:     function setXkiloToken(address _xkiloToken) external onlyOwner {
37:         xkiloToken = IERC20Upgradeable(_xkiloToken);
38:     }

43:     event OwnerSetPerpTrade(address perpTrade);

49:     event OwnerSetMinMargin(uint256 minMargin);
50:     event OwnerSetMinProfitTime(uint256 minProfitTime);
51:     event OwnerSetExposureMultiplier(uint256 exposureMultiplier);
52:     event OwnerSetUtilizationMultiplier(uint256 utilizationMultiplier);
53:     event OwnerSetMaxExposureMultiplier(uint256 maxExposureMultiplier);
54:     event OwnerSetMaxShift(uint256 maxShift);
55:     event OwnerSetLiquidationParams(uint256 liquidationBounty, uint256 liquidationThreshold);
56:     event OwnerSetAllowPublicLiquidator(bool allowPublicLiquidator);
57:
58:
59:     event OwnerSetTradeEnabled(bool isTradeEnabled);
```

```
60:     event OwnerSetCanUserStake(bool canUserStake);
61:
62:     event OwnerSetParameters(uint256 maxShift,
63:         uint256 minProfitTime,
64:         bool canUserStake,
65:         bool allowPublicLiquidator,
66:         uint256 exposureMultiplier,
67:         uint256 utilizationMultiplier,
68:         uint256 maxExposureMultiplier,
69:         uint256 liquidationBounty,
70:         uint256 liquidationThreshold);
71:     event OwnerPauseTrading(bool isTradeEnabled, bool canUserStake);
72:     event OwnerSetAdlMultiplier(uint256 adlMultiplier);
73:     event OwnerSetToken(address token, uint256 decimals);
74:
75:     event OwnerSetMarginFeeManager(address marginFeeManagerAddr);
76:     event OwnerSetPerpTrade(address perpTradeAddr);
77:     event OwnerSetPendingReward(address pendingRewardAddr);
78:     event OwnerSetKiloPriceFeed(address kiloPriceFeedAddr);
79:     event OwnerSetOrderBook(address orderBookAddr);
80:     event OwnerSetPositionRouter(address positionRouterAddr);
81:     event OwnerSetStakeReward(address stakeRewardAddr);
82:     event OwnerSetProductManager(address productManagerAddr);
83:     event OwnerSetBorrowingManager(address borrowingManagerAddr);
```

```
54:     function setPerpTrade(address _perpTrade) external onlyOwner {
55:         perpTrade = _perpTrade;
56:     }
```

```
55:     event OwnerSetVaultCapAndPeriod(
56:         uint256 cap,
57:         uint256 stakingPeriod
58:     );
59:
60:     event SetApy(uint256 apy,uint256 originalApy);
61:     event OwnerSetKeeper(address keeper);
```

```
58:     function setProtocolReward(address _protocolReward) external onlyOwner {
59:         protocolReward = _protocolReward;
60:     }
61:
62:     function setXkiloToken(address _xkiloToken) external onlyOwner {
63:         xkiloToken = IERC20Upgradeable(_xkiloToken);
```

```
64:     }

114:    function setVaultStakeReward(address _vaultStakeReward) external onlyOwner {
115:        vaultStakeReward = _vaultStakeReward;
116:    }

147:    function setPrices(address[] memory tokens, uint256[] memory prices) external onlyKeeper {
148:        require(tokens.length == prices.length, "!length");
149:        for (uint256 i = 0; i < tokens.length; i++) {
150:            address token = tokens[i];
151:            priceMap[token] = prices[i];
152:            lastUpdatedTimes[token] = block.timestamp;
153:            emit PriceSet(token, prices[i], block.timestamp);
154:        }
155:        lastUpdatedTime = block.timestamp;
156:    }

180:    function setOracle(address _oracle) external onlyOwner {
181:        oracle = _oracle;
182:    }

181:    function setPerpTrade(address _perpTrade) external onlyOwner {
182:        perpTrade = _perpTrade;
183:    }

199:    function setOracle(address _oracle) external onlyOwner {
200:        oracle = _oracle;
201:    }

448:    function setApprovedRouter(address _manager, bool _isActive) external onlyOwner {
449:        approvedRouters[_manager] = _isActive;
450:    }
451:
452:    function setOracle(address _oracle) external onlyOwner {
453:        oracle = _oracle;
454:    }
455:
456:    function setMarginFeeManager(address _marginFeeManager) external onlyOwner {
457:        marginFeeManager = _marginFeeManager;
458:    }
459:
460:    function setLiquidator(address _liquidator, bool _isActive) external onlyOwner {
```

```
461:         liquidators[_liquidator] = _isActive;
462:     }
```

Description

danielt : In the `ProtocolReward` contract, the below key function lack of missing event:

- `setXkiloToken`

In the `TradeRewardDistributor` contract, the below key function lack of missing event:

- `setProtocolReward`
- `setXkiloToken`

In the `Governable` contract, the below key function lack of missing event:

- `setGov`

In the `GovernableUpgradeable` contract, the below key function lack of missing event:

- `setGov`

In the `KiloStorageManager` contract, the below key function lack of missing event:

- `clearPosition`

Emitting an event when updating the state variable is important to track the state of the contract.
0xzoobi : Throughout the code, there are parameters in events that are not `indexed`. Consider indexing event parameters to avoid hindering the task of off-chain services searching and filtering for specific events.
0xzoobi : When changing state variables events are not emitted.
Events are an important feature in smart contracts and are primarily used to facilitate the communication and interaction between smart contracts and external entities, such as user interfaces and off-chain applications
Yaodao : Functions that update state variables should emit relevant events as notifications.
ginlee : Index event fields make the field more quickly accessible to off-chain.Each event should use three indexed fields if there are three or more fields.

Recommendation

danielt : Emit events for key functions that update the state variables.
0xzoobi : add `indexed` parameter for critical or important events.
0xzoobi : Emit an event when setting or updating an important state variable. Please make sure they are `indexed` if needed.
Yaodao : Recommend adding events for state-changing actions, and emitting them in their relevant functions.
ginlee : Use three indexed for each event if there are three or more fields

Client Response

Mitigated, Part of it has been optimized, and we are currently synchronizing to the database

KEX-59:FEATURE SUGGESTION - Provide incentives to user's for calling `claim` on behalf of a user

Category	Severity	Status	Contributor
Logical	Informational	Declined	0xzoobi

Code Reference

- code/src/tradereward/TradeRewardDistributor.sol#L71-L87

```
71:     function claim(uint256 _amount, uint256 _xkiloAmount, bytes32[] calldata _merkleProof) extern
al whenNotPaused nonReentrant {
72:         // Verify the reward round is not claimed already
73:         require(!hasUserClaimedForRewardRound[currentRewardRound][msg.sender], "Rewards: Already
claimed");
74:         (uint256 adjustedAmount, uint256 adjustedXkiloAmount)  = _pendingRewards(msg.sender, _amo
unt, _xkiloAmount, _merkleProof);
75:         require(adjustedAmount > 0 || adjustedXkiloAmount > 0, "Rewards: no claimable rewards");
76:
77:         hasUserClaimedForRewardRound[currentRewardRound][msg.sender] = true;
78:         if (adjustedAmount > 0) {
79:             amountClaimedByUser[msg.sender] += adjustedAmount;
80:             rewardToken.safeTransfer(msg.sender, adjustedAmount);
81:         }
82:         if (adjustedXkiloAmount > 0 && address(xkiloToken) != address(0)) {
83:             xkiloClaimedByUser[msg.sender] += adjustedXkiloAmount;
84:             xkiloToken.safeTransfer(msg.sender, adjustedXkiloAmount);
85:         }
86:         emit RewardClaimed(msg.sender, currentRewardRound, adjustedAmount, adjustedXkiloAmount);
87:     }
```

Description

0xzoobi : The current implementation of `claim` is such that, if a user does not call the function for a particular `curren tRewardRound` and if the round is completed and a new round has started, the user has effectively lost the reward. Looking at the code, the `onlyGov` can later withdraw these tokens rewards for themselves.

This implementation is good enough but what if an incentive is added to for user to call the `claim` function of behalf of a user. The caller can probably like 10-20% of the total reward as commission and remainder is transferred to the user

instead of `onlyGov` user.

Recommendation

0xzoobi : Add an incentive for caller to claim rewards on behalf of other users.

Consider below implmentation

```
function claimNew(address _user, uint256 _amount, uint256 _xkiloAmount, bytes32[] calldata _merklePr
oof) external whenNotPaused nonReentrant {
        // Verify the reward round is not claimed already
        require(!hasUserClaimedForRewardRound[currentRewardRound][_user], "Rewards: Already claime
d");
        (uint256 adjustedAmount, uint256 adjustedXkiloAmount)  = _pendingRewards(_user, _amount, _xk
iloAmount, _merkleProof);
        require(adjustedAmount > 0 || adjustedXkiloAmount > 0, "Rewards: no claimable rewards");
        hasUserClaimedForRewardRound[currentRewardRound][_user] = true;

        if(msg.sender == _user) {
            //do what is already implemented
        }
        else {
            //transfer 10-20% of the rewards to caller
            //transfer remaining rewards to the actual user
        }
    }
```

Client Response

Declined, Users can withdraw their full reward at any round and will not lose their reward

KEX-60:Gas Optimization: ++i costs less gas than i++, especially when it's used in for-loops (--i/i-- too)

Category	Severity	Status	Contributor
Gas Optimization	Informational	Declined	LiRiu

Code Reference

- code/src/referrals/ReferralReader.sol#L9
- code/src/peripherals/LiquidationPriceReader.sol#L70
- code/src/peripherals/KiloPerpView.sol#L72
- code/src/core/KiloPriceFeed.sol#L80
- code/src/peripherals/KiloPerpView.sol#L98
- code/src/peripherals/LiquidationPriceReader.sol#L103
- code/src/peripherals/LiquidationPriceReader.sol#L130
- code/src/peripherals/KiloPerpView.sol#L133
- code/src/core/KiloPriceFeed.sol#L140
- code/src/core/KiloPriceFeed.sol#L149
- code/src/core/OrderBook.sol#L225
- code/src/core/OrderBook.sol#L231
- code/src/core/OrderBook.sol#L243
- code/src/core/OrderBook.sol#L246
- code/src/core/PerpTrade.sol#L294

```
9:        for (uint256 i = 0; i < _codes.length; i++) {
70:        for (uint256 i; i < account.length; i++) {
72:        for (uint256 i; i < ids.length; i++) {
80:        return lastUpdatedTime + updateInterval < block.timestamp;
98:        for (uint256 i; i < ids.length; i++) {
103:        for (uint256 i; i < account.length; i++) {
130:        for (uint256 i; i < tokens.length; i++) {
133:        for (uint256 i; i < ids.length; i++) {
140:        for (uint256 i = 0; i < tokens.length; i++) {
149:        for (uint256 i = 0; i < tokens.length; i++) {
225:        for (uint256 i = 0; i < _openAddresses.length; i++) {
231:        for (uint256 i = 0; i < _closeAddresses.length; i++) {
243:        for (uint256 i = 0; i < _increaseOrderIndexes.length; i++) {
246:        for (uint256 i = 0; i < _decreaseOrderIndexes.length; i++) {
294:        for (uint256 i = 0; i < positionIds.length; i++) {
```

Description

LiRiu : `++i` costs less gas than `i++`, especially when it's used in for-loops (`--i` / `i--` too)

Recommendation

LiRiu : Recommend using `++i` instead of `i++`, `--i` / `i--` too

Client Response

Declined, i++ is more in line with reading and usage habits, and will not be processed for now

KEX-61:Gas Optimization: Don’t initialize variables with default value

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	ginlee

Code Reference

- code/src/referrals/ReferralReader.sol#L9
- code/src/core/KiloPriceFeed.sol#L89
- code/src/core/KiloPriceFeed.sol#L140
- code/src/core/KiloPriceFeed.sol#L149
- code/src/core/OrderBook.sol#L225
- code/src/core/OrderBook.sol#L231
- code/src/core/OrderBook.sol#L243
- code/src/core/OrderBook.sol#L246
- code/src/core/PerpTrade.sol#L294

```
9:        for (uint256 i = 0; i < _codes.length; i++) {

89:            if (lastUpdatedTimes[tokens[i]] + updateInterval < block.timestamp) {

140:        for (uint256 i = 0; i < tokens.length; i++) {

149:        for (uint256 i = 0; i < tokens.length; i++) {

225:        for (uint256 i = 0; i < _openAddresses.length; i++) {

231:        for (uint256 i = 0; i < _closeAddresses.length; i++) {

243:        for (uint256 i = 0; i < _increaseOrderIndexes.length; i++) {

246:        for (uint256 i = 0; i < _decreaseOrderIndexes.length; i++) {

294:        for (uint256 i = 0; i < positionIds.length; i++) {
```

Description

ginlee : No need to initialize variables with default value

Recommendation

ginlee : change "uint256 i = 0" to "uint256 i"

Client Response

Fixed

KEX-62:Gas Optimization: Double initialize `owner`

Category	Severity	Status	Contributor
Logical	Informational	Fixed	Hacker007

Code Reference

- code/src/core/MarginFeeManager.sol#L42-L47

```
42:     function initialize(
43:         address _productManager,
44:         address _kiloStorageAddr
45:     ) public initializer {
46:         __owner_governable_init();
47:         owner = msg.sender;
```

Description

Hacker007 : The variable `owner` is already initialized in the function `__owner_governable_init()`.

```
    function __owner_governable_init() internal initializer {
        gov = msg.sender;
        owner = msg.sender;
    }
```

So there is no need to initialize `owner` again in the function `initialize()`.

```
    function initialize(
        address _productManager,
        address _kiloStorageAddr
    ) public initializer {
        __owner_governable_init();
        owner = msg.sender;
```

Recommendation

Hacker007 : Remove the redundant assignment.

Client Response

Fixed

KEX-63:Gas Optimization: Function `_calculatePrice` repeatedly implementation

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	xfu

Code Reference

- code/src/libraries/PerpTradeUtil.sol#L138
- code/src/core/PerpTrade.sol#L145

```
138:    function _calculatePrice(

145:        (vars.funding, vars.borrowing) = IMarginFeeManager(marginFeeManager).updateMarginFee(pro
ductId);
```

Description

xfu : - libraries/PerpTradeUtil.sol:138

- core/PerpTrade.sol:415

```
function _calculatePrice(
        address oracle,
        address productToken,
        bool isLong,
        uint256 openInterestLong,
        uint256 openInterestShort,
        uint256 maxExposure,
        uint256 reserve,
        uint256 amount,
        uint256 maxShift
    ) internal view returns (uint256)
```

Recommendation

xfu : function `_calculatePrice` has been repeatedly implemented in PerpTradeUtil and PerpTrade, remove function `_calculatePrice` in PerpTradeUtil

Client Response

Fixed, same as kex-57

KEX-64:Gas Optimization: State variables set in the constructor can be declared `immutable`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	0xzoobi

Code Reference

- code/src/peripherals/LiquidationPriceReader.sol#L14-L16

```
14:     address private marginFeeManager;
15:     IKiloStorage private kiloStorage;
16:     IOracle private oracle;
```

Description

0xzoobi : The following variables `kiloStorage`, `marginFeeManager` and `oracle` are set once the constructor is called and never updated. Its better of to declare the state variables as `immutable` to save some gas.

Recommendation

0xzoobi : Declare the state variables defined in constructor as `immutable`.

Sample Fix

```
    address private immutable marginFeeManager;
    IKiloStorage private immutable kiloStorage;
    IOracle private immutable oracle;
```

Client Response

Fixed

KEX-65:Gas Optimization: Unused Local return Variable in `PerpTrade.sol`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	xfu

Code Reference

- code/src/core/PerpTrade.sol#L240
- code/src/core/PerpTrade.sol#L320

```
240:        (int256 funding, uint256 borrowing) = IMarginFeeManager(marginFeeManager).updateMarginFe
e(position.productId);

320:        (int256 funding, uint256 borrowing) = IMarginFeeManager(marginFeeManager).updateMarginFe
e(position.productId);
```

Description

xfu : - core/PerpTrade.sol:240

```
(int256 funding, uint256 borrowing) = IMarginFeeManager(marginFeeManager).updateMarginFee(position.p
roductId);
```

- core/PerpTrade.sol:320

```
(int256 funding, uint256 borrowing) = IMarginFeeManager(marginFeeManager).updateMarginFee(position.p
roductId);
```

Recommendation

xfu : remove local return variable:

```
IMarginFeeManager(marginFeeManager).updateMarginFee(position.productId);
```

Client Response

Fixed

KEX-66:Gas Optimization: Unused event

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	Hacker007

Code Reference

- code/src/core/KiloStorageManager.sol#L83

```
83:     event OwnerSetBorrowingManager(address borrowingManagerAddr);
```

Description

Hacker007 : The event `OwnerSetBorrowingManager` is declared but never used in the contract.

Recommendation

Hacker007 : remove the unused event.

Client Response

Fixed

KEX-67:Gas Optimization: Unused statements in contract `ProductManager`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	Hacker007

Code Reference

- code/src/core/ProductManager.sol#L9
- code/src/core/ProductManager.sol#L181-L188

```
9:    address private perpTrade;

181:    function setPerpTrade(address _perpTrade) external onlyOwner {
182:        perpTrade = _perpTrade;
183:    }
184:
185:    modifier onlyPerpTrade() {
186:        require(msg.sender == perpTrade, "not perpTrade");
187:        _;
188:    }
```

Description

Hacker007 : The following function `setPerpTrade`, state `perpTrade`, and modifier `onlyPerpTrade` are defined in the `ProductManager` contract, but appear to be unused and irrelevant to the main logic:

```
contract ProductManager is IProductManager, OwnerGovernableUpgradeable {

    address private perpTrade;
    //...
    function setPerpTrade(address _perpTrade) external onlyOwner {
        perpTrade = _perpTrade;
    }

    modifier onlyPerpTrade() {
        require(msg.sender == perpTrade, "not perpTrade");
        _;
    }
}
```

Recommendation

Hacker007 : To improve the codebase, consider either using these functions or removing them. Removing unused functions can help to reduce clutter and make the code easier to understand.

Client Response

Fixed

KEX-68:Gas Optimization: Using calldata instead of memory for read-only arguments in external functions saves gas

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	danielt, Yaodao

Code Reference

- code/src/core/KiloPriceFeed.sol#L138
- code/src/core/KiloPriceFeed.sol#L147
- code/src/core/PerpTrade.sol#L196
- code/src/core/OrderBook.sol#L204-L216
- code/src/core/OrderBook.sol#L204
- code/src/core/OrderBook.sol#L217
- code/src/core/OrderBook.sol#L239
- code/src/core/PositionRouter.sol#L269
- code/src/core/PositionRouter.sol#L278

```
138:    function getPrices(address[] memory tokens) external view returns (uint256[] memory){

147:    function setPrices(address[] memory tokens, uint256[] memory prices) external onlyKeeper {

196:    function adlDecreasePosition(

204:    function executeOrdersWithPrices(

204:    function executeOrdersWithPrices(
205:        address[] memory tokens,
206:        uint256[] memory prices,
207:        address[] memory _openAddresses,
208:        uint256[] memory _increaseOrderIndexes,
209:        address[] memory _closeAddresses,
210:        uint256[] memory _decreaseOrderIndexes,
211:        address payable _feeReceiver
212:    ) external onlyKeeper {
213:        IOracle(oracle).setPrices(tokens, prices);
214:        executeOrders(_openAddresses, _increaseOrderIndexes, _closeAddresses, _decreaseOrderInde
xes, _feeReceiver);
215:    }
216:

217:    function executeOrders(

239:    function cancelMultiple(

269:    function executeNPositionsWithPrices(

278:    function executePositionsWithPrices(
```

Description

danielt : When a function with a memory array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. Each iteration of this for-loop costs at least 60 gas (i.e. 60 * <mem_array>.length). Using `calldata` directly, obliviates the need for such a loop in the contract code and runtime execution.

If the array is passed to an internal function which passes the array to another internal function where the array is modified and therefore memory is used in the external call, it's still more gas-efficient to use `calldata` when the external function uses modifiers, since the modifiers may prevent the internal functions from being called.
Here are three examples:

```
    function executeOrdersWithPrices(
        address[] memory tokens,
        uint256[] memory prices,
        address[] memory _openAddresses,
        uint256[] memory _increaseOrderIndexes,
        address[] memory _closeAddresses,
        uint256[] memory _decreaseOrderIndexes,
        address payable _feeReceiver
    ) external onlyKeeper {
        ...
    }

    function executeOrders(
        address[] memory _openAddresses,
        uint256[] memory _increaseOrderIndexes,
        address[] memory _closeAddresses,
        uint256[] memory _decreaseOrderIndexes,
        address payable _feeReceiver
    ) public onlyKeeper {
    ...
    }

    function cancelMultiple(
        uint256[] memory _increaseOrderIndexes,
        uint256[] memory _decreaseOrderIndexes
    ) external {
       ...
    }
```

Yaodao : It's better to use calldata instead of memory for function parameters that represent variables that will not be modified.

Recommendation

danielt : Using calldata instead of memory for read-only arguments in external functions saves gas
Yaodao : Consider using calldata instead of memory to save gas.

Client Response

Fixed

KEX-69:Gas Optimization: Using private rather than public for constants, saves gas

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	LiRiu

Code Reference

- code/src/core/KiloPriceFeed.sol#L28-L29

```
28:     uint256 public constant MAX_PRICE_DURATION = 30 minutes;
29:     uint256 public constant SPREAD_PRICE_BASE = 10000;
```

Description

LiRiu : Using the 'private' modifier to declare global constants can reduce gas consumption.

Recommendation

LiRiu : Using the 'private' modifier to declare global constants

```
    uint256 public constant MAX_PRICE_DURATION = 30 minutes;
    uint256 public constant SPREAD_PRICE_BASE = 10000;
```

Client Response

Fixed

KEX-70:Gas Optimization: `setReferrerDiscountShare` require conditions can be combined into a single statement

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	0xzoobi

Code Reference

- code/src/referrals/ReferralStorageManager.sol#L74-L75

```
74:         require(_discountShare <= maxDiscountShares, "ReferralStorage: invalid _discountShare");
75:         require(_discountShare >= minDiscountShares, "ReferralStorage: invalid _discountShare");
```

Description

0xzoobi : The `setReferrerDiscountShare` has two require statements in place to check if `_discountShare` is between the range of `minDiscountShares` and `maxDiscountShares`. A better approach would be to combine both the checks into a single check.

Recommendation

0xzoobi : Change code to check if `_discountShare` is between the range of `minDiscountShares` and `maxDiscountShares`.

Consider below fix

```
require(_discountShare >= minDiscountShares && _discountShare <= maxDiscountShares, "ReferralStorage: invalid _discountShare");
```

Client Response

Fixed

KEX-71:Gas Optimization: function `_updateOpenInterest` never used

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	xfu

Code Reference

- code/src/core/PerpTrade.sol#L368-L382

```
368:    function _updateOpenInterest(
369:        uint256 productId,
370:        uint256 addedOpenInterest,
371:        bool isLong,
372:        bool isIncrease,
373:        uint256 productWeight,
374:        IKiloStorage.KiloConfig memory kiloConfig
375:    ) private {
376:        if (isIncrease) {
377:            uint256 maxExposure = _getMaxExposure(productWeight, kiloConfig.exposureMultiplier);
378:            kiloStorage.updateIncreaseOpenInterest(productId, addedOpenInterest, isLong, _getVau
ltBalance(), maxExposure);
379:        } else {
380:            kiloStorage.updateDecreaseOpenInterest(productId, addedOpenInterest, isLong);
381:        }
382:    }
```

Description

xfu : - core/PerpTrade.sol:368

```
function _updateOpenInterest(
        uint256 productId,
        uint256 addedOpenInterest,
        bool isLong,
        bool isIncrease,
        uint256 productWeight,
        IKiloStorage.KiloConfig memory kiloConfig
    ) private {
        if (isIncrease) {
            uint256 maxExposure = _getMaxExposure(productWeight, kiloConfig.exposureMultiplier);
            kiloStorage.updateIncreaseOpenInterest(productId, addedOpenInterest, isLong, _getVaultBa
lance(), maxExposure);
        } else {
            kiloStorage.updateDecreaseOpenInterest(productId, addedOpenInterest, isLong);
        }
    }
```

this function never used in PerpTrade.sol

Recommendation

xfu : remove function `_updateOpenInterest` in PerpTrade.sol

Client Response

Fixed, same as kex-57

KEX-72:Gas Optimization: in `IKiloStorage.sol`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	0xzoobi

Code Reference

- code/src/interfaces/IKiloStorage.sol#L5-L18

```
5:    struct KiloConfig {
6:        uint256 minMargin;//Minimum margin for opening a position
7:        uint256 maxShift;// = 0.002e8; // max shift (shift is used adjust the price to balance the longs and shorts)
8:        uint256 minProfitTime;// = 6 hours;
9:        uint256 exposureMultiplier;// = 10000;
10:        uint256 utilizationMultiplier;// = 10000; //Vault utilization rate multiplier
11:        uint256 maxExposureMultiplier;// = 3;
12:        uint256 liquidationBounty;// = 5000; //Liquidation reward percentage in basis points. 5000 = 50%
13:        uint256 liquidationThreshold;// = 8000; //Liquidation threshold in basis points. 8000 = 80%
14:        bool canUserStake;
15:        bool allowPublicLiquidator;
16:        bool isTradeEnabled;
17:        uint256 adlMultiplier;
18:    }
```

Description

0xzoobi : The current implementation needs about 10 slots of storage for each `KiloConfig` struct.

image

The below shown changes can save significant amount of gas, instead of 10 storage slots only 1 storage slot is needed. I am aware of the fact that BSC had cheap gas costs, but 10x saving for each time the struct is used can save a enormous gas costs in long run.

image

A similar change can be done to `IProductManager's` `Product` struct.

Recommendation

0xzoobi : Make the above changes to reduce gas costs and improve UX.

Client Response

Fixed, same as kex-57

KEX-73:Gas Optimization: in `KiloStorageManager`::`updateIncreaseOpenInterest()`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	danielt

Code Reference

- code/src/core/KiloStorageManager.sol#L304-L323

```
304:    function updateIncreaseOpenInterest(
305:        uint256 productId,
306:        uint256 amount,
307:        bool isLong,
308:        uint256 _vaultBalance,
309:        uint256 maxExposure
310:    ) external override onlyPerpTrade {
311:        totalOpenInterest = totalOpenInterest + amount;
312:
313:        require(totalOpenInterest <= _vaultBalance * utilizationMultiplier / 10 ** 4 &&
314:            longOpenInterests[productId] + shortOpenInterests[productId] + amount < maxExposureM
ultiplier * maxExposure, "!maxOI");
315:
316:        if (isLong) {
317:            longOpenInterests[productId] = longOpenInterests[productId] + amount;
318:            require(longOpenInterests[productId] <= maxExposure + shortOpenInterests[productId],
"!exposure-long");
319:        } else {
320:            shortOpenInterests[productId] = shortOpenInterests[productId] + amount;
321:            require(shortOpenInterests[productId] <= maxExposure + longOpenInterests[productId],
"!exposure-short");
322:        }
323:    }
```

Description

danielt : In `KiloStorageManager` contract, the `updateIncreaseOpenInterest` function updates `longOpenInterests` and `shortOpenInterests`.

However, we can reduce the read of the storage variables of `longOpenInterests` and `shortOpenInterests` with usages of the memory variable.

Example:

```
        uint256 longOpenInterest = longOpenInterests[productId];
        uint256 shortOpenInterest = shortOpenInterests[productId]
        require(totalOpenInterest <= _vaultBalance * utilizationMultiplier / 10 ** 4 &&
            longOpenInterest + shortOpenInterest + amount < maxExposureMultiplier * maxExposure, "!m
axOI");

        if (isLong) {
            longOpenInterests[productId] = longOpenInterest + amount;
            require(longOpenInterest + amount <= maxExposure + shortOpenInterest, "!exposure-long");
        }  else {
            shortOpenInterests[productId] = shortOpenInterest+ amount;
            require(shortOpenInterest + amount <= maxExposure + longOpenInterest, "!exposure-shor
t");
        }
```

Recommendation

danielt : Reducing the read of the storage variable by using the memory variables

Client Response

Fixed

KEX-74:Gas Optimization: in `ReferralReader`::`getCodeOwners()`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	newway55

Code Reference

- code/src/referrals/ReferralReader.sol#L7-L17

```
7:    function getCodeOwners(IReferralStorage _referralStorage, bytes32[] memory _codes) public view
returns (address[] memory) {
8:        address[] memory owners = new address[](_codes.length);
9:        for (uint256 i = 0; i < _codes.length; i++) {
10:            bytes32 code = _codes[i];
11:            owners[i] = _referralStorage.codeOwners(code);
12:        }
13:        return owners;
14:    }
15:
16:
17:}
```

Description

newway55 : Storing codes into a variable is adding more gas consumption. In this case removing variables codes makes more sense while it's not really accurate and can be directly accessed from the param of codeOwners();

Consider below POC contract

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.0;

import "../src/referrals/ReferralReader.sol";
import "../src/referrals/interfaces/IReferralStorage.sol";
import "forge-std/Test.sol";

interface IStorage {
    function codeOwners(bytes32 _code) external view returns (address);

}

contract ReferralReaderTest is Test {
    ReferralReader public referralReader;
    IReferralStorage public referralStorage;
    function setUp() public {
        referralReader = new ReferralReader();
    }

    function testGetCodeOwners() public {

        bytes32[] memory codes = new bytes32[](3);
        codes[0] = bytes32("0x123");
        codes[1] = bytes32("0x456");
        codes[2] = bytes32("0x789");

        address[] memory expectedOwners = new address[](3);
        address _storageEx = address(40000);

        address storing0 = IReferralStorage(_storageEx).codeOwners(codes[0]);
        address storing1 = IReferralStorage(_storageEx).codeOwners(codes[1]) ;
        address storing2 = IReferralStorage(_storageEx).codeOwners(codes[2]);

        expectedOwners[0] = storing0;
        expectedOwners[1] = storing1;
        expectedOwners[2] = storing2;

        address[] memory result = referralReader.getCodeOwners(referralStorage, codes);

        for (uint i = 0; i < expectedOwners.length ; i++){
            emit log_address(result[i]);
            assertEq(expectedOwners[i] ,  result[i]);
```

```
}

    }
}
```

Recommendation

newway55 : Remove L12

Consider below fix in the `sample.test()` function

```
// SPDX-License-Identifier: MIT
pragma solidity 0.8.17;

import "./interfaces/IReferralStorage.sol";

contract ReferralReader {
    function getCodeOwners(
        IReferralStorage _referralStorage,
        bytes32[] memory _codes
    ) public view returns (address[] memory) {
        address[] memory owners = new address[](_codes.length);
        for (uint256 i = 0; i < _codes.length; i++) {
            owners[i] = _referralStorage.codeOwners(_codes[i]);
        }
        return owners;
    }
}
```

Client Response

Fixed

KEX-75:Gas Optimization: in `ReferralStorageManager.sol`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	0xzoobi

Code Reference

- code/src/referrals/ReferralStorageManager.sol#L9-L16

```
9:    struct Tier {
10:        uint256 totalRebate; // e.g. 2400 for 24%
11:        uint256 discountShare; // 5000 for 50%/50%, 7000 for 30% rebates/70% discount
12:        bool canChangeDiscountShare; //discountShare changed by user
13:    }
14:
15:    uint256 public constant DEFAULT_REFERRAL_TIER = 1;
16:    uint256 public constant BASIS_POINTS = 10000;
```

Description

0xzoobi : The below shown changes can save significant amount of gas, since you expect to store this information for every user.

Note: 10000 BPS == 100%.

 image

Recommendation

0xzoobi : Implement the suggested changes to reduce gas costs exponentially.

Client Response

Fixed

KEX-76:Gas Optimization: state mutability of `getPrices` can be restricted to `view`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	0xzoobi

Code Reference

- code/src/peripherals/LiquidationPriceReader.sol#L126

```
126:    function getPrices(address[] memory tokens) external returns (uint256[] memory prices, uint256[] memory timestamps) {
```

Description

0xzoobi : The `getPrices` function doesn't update any state of the blockchain but just calculates the prices of tokens via oracles.

Recommendation

0xzoobi : change state mutability to `view`.

Client Response

Fixed

KEX-77:Gas optimization: in `VaultStakeReward.sol`

Category	Severity	Status	Contributor
Gas Optimization	Informational	Declined	xfu

Code Reference

- code/src/core/VaultStakeReward.sol#L35
- code/src/core/VaultStakeReward.sol#L230

```
35:    mapping(address => Stake) private stakes;

230:    function getStake(address stakeOwner) external view returns (Stake memory) {
```

Description

xfu : - core/VaultStakeReward.sol:35

```
mapping(address => Stake) private stakes;
```

- core/VaultStakeReward.sol:230

```
function getStake(address stakeOwner) external view returns (Stake memory) {
        return stakes[stakeOwner];
}
```

Recommendation

xfu : set state variable `stakes` as public, remove getStake function

severity: Informational

Client Response

Declined, The test that has been touched, and there are too many dependencies, so I will not deal with it for the time being

KEX-78:Incomplete input validation in `KiloStorageManager`

Category	Severity	Status	Contributor
Logical	Informational	Fixed	Hacker007

Code Reference

- code/src/core/KiloStorageManager.sol#L119-L123
- code/src/core/KiloStorageManager.sol#L134-L148

```
119:     function setMinProfitTime(uint256 _minProfitTime) external onlyOwner {
120:         minProfitTime = _minProfitTime;
121:         emit OwnerSetMinProfitTime(_minProfitTime);
122:     }
123:

134:     function setMaxExposureMultiplier(uint256 _maxExposureMultiplier) external onlyOwner {
135:         maxExposureMultiplier = _maxExposureMultiplier;
136:         emit OwnerSetMaxExposureMultiplier(_maxExposureMultiplier);
137:     }
138:
139:     function setMaxShift(uint256 _maxShift) external onlyOwner {
140:         maxShift = _maxShift;
141:         emit OwnerSetMaxShift(_maxShift);
142:     }
143:
144:     function setLiquidationParams(uint256 _liquidationBounty, uint256 _liquidationThreshold) ext
ernal onlyOwner {
145:         liquidationBounty = _liquidationBounty;
146:         liquidationThreshold = _liquidationThreshold;
147:         emit OwnerSetLiquidationParams(_liquidationBounty, _liquidationThreshold);
148:     }
```

Description

Hacker007 : The function `setParameters()` checks the input parameters as below:

```
    function setParameters(
        uint256 _maxShift,
        uint256 _minProfitTime,
        bool _canUserStake,
        bool _allowPublicLiquidator,
        uint256 _exposureMultiplier,
        uint256 _utilizationMultiplier,
        uint256 _maxExposureMultiplier,
        uint256 _liquidationBounty,
        uint256 _liquidationThreshold
    ) external onlyOwner {
        require(_maxShift <= 0.01e8 && _minProfitTime <= 24 hours && _liquidationThreshold > 5000 &&
_maxExposureMultiplier > 0);
        //...
    }
```

However, the standalone setting functions do not have a sanity check as `setParameters()` do.

```
    function setMinProfitTime(uint256 _minProfitTime) external onlyOwner {
        minProfitTime = _minProfitTime;
        emit OwnerSetMinProfitTime(_minProfitTime);
    }

    function setMaxExposureMultiplier(uint256 _maxExposureMultiplier) external onlyOwner {
        maxExposureMultiplier = _maxExposureMultiplier;
        emit OwnerSetMaxExposureMultiplier(_maxExposureMultiplier);
    }
    function setMaxShift(uint256 _maxShift) external onlyOwner {
        maxShift = _maxShift;
        emit OwnerSetMaxShift(_maxShift);
    }

    function setLiquidationParams(uint256 _liquidationBounty, uint256 _liquidationThreshold) externa
l onlyOwner {
        liquidationBounty = _liquidationBounty;
        liquidationThreshold = _liquidationThreshold;
        emit OwnerSetLiquidationParams(_liquidationBounty, _liquidationThreshold);
    }
```

Recommendation

Hacker007 : Add a proper check to the corresponding function as `setParameters()` do.

Client Response

Fixed

KEX-79:Incorrect commets

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	Yaodao

Code Reference

- code/src/core/KiloPriceFeed.sol#L25
- code/src/core/KiloPriceFeed.sol#L57

```
25:    uint256 public defaultMaxPriceDiff; // 2e16/2e18=2%

57:        defaultMaxPriceDiff = 2e16; // 2e16/2e18=2%
```

Description

Yaodao : The following comments in the contract `KiloPriceFeed` are incorrect.

Consider below codes:

```
defaultMaxPriceDiff = 2e16;; // 2e16/2e18=2%
```

Recommendation

Yaodao : Recommend updating the comment.

Consider below fix

```
defaultMaxPriceDiff = 2e16;; // 2e16/1e18=2%
```

Client Response

Fixed

KEX-80:Invalid contract specified in override list: "IERC20MetadataUpgradeable".

Category	Severity	Status	Contributor
Compiler Error	Informational	Acknowledged	xfu

Code Reference

- code/src/core/VaultStakeReward.sol#L234-L236

```
234:    function decimals() public view override(ERC20Upgradeable, IERC20MetadataUpgradeable) return
s (uint8) {
235:        return IERC20MetadataUpgradeable(token).decimals();
236:    }
```

Description

xfu : - core/VaultStakeReward.sol:290:34

```
function decimals() public view override(ERC20Upgradeable, IERC20MetadataUpgradeable) returns (uint
8) {
        return IERC20MetadataUpgradeable(token).decimals();
}
```

Recommendation

xfu :

```
function decimals() public view override(ERC4626Upgradeable, ERC20Upgradeable) returns (uint8) {
        return IERC20MetadataUpgradeable(token).decimals();
}
```

Client Response

Acknowledged

KEX-81:Lack of upper and lower boundary checks in `ProductManager`::`setLeverage()`

Category	Severity	Status	Contributor
Logical	Informational	Fixed	Yaodao

Code Reference

- code/src/core/ProductManager.sol#L105-L110

```
105:    function setLeverage(uint256 id, uint256 _maxLeverage, uint256 _minLeverage) external onlyOw
ner {
106:        address token = productToken[id];
107:        maxLeverage[token] = _maxLeverage;
108:        minLeverage[token] = _minLeverage;
109:        emit OwnerSetLeverage(id, token, _maxLeverage, _minLeverage);
110:    }
```

Description

Yaodao : According to the following codes, the function `setLeverage()` is used to update the `maxLeverage` and `minLeverage` for the token. However, these given parameters are not checked like the function `updateProduct()`.

Consider below codes

```
    function setLeverage(uint256 id, uint256 _maxLeverage, uint256 _minLeverage) external onlyOwner
{
        address token = productToken[id];
        maxLeverage[token] = _maxLeverage;
        minLeverage[token] = _minLeverage;
        emit OwnerSetLeverage(id, token, _maxLeverage, _minLeverage);
    }
```

Recommendation

Yaodao : Recommend adding reasonable upper and low boundary checks.

Consider below fix

```
    require(_maxLeverage > _minLeverage);
    require(_minLeverage >= 1 * BASE);
```

Client Response

Fixed, same as kex-42

KEX-82:No need to use SafeMath in solidity version 0.8+

Category	Severity	Status	Contributor
Gas Optimization	Informational	Fixed	Yaodao, Hacker007, ginlee

Code Reference

- code/src/core/KiloPriceFeed.sol#L7
- code/src/core/KiloPriceFeed.sol#L10

```
7:import "@openzeppelin/contracts/utils/math/SafeMath.sol";

10:    using SafeMath for uint256;
```

Description

Yaodao : Solidity provides overflow checking for version above 0.8. The contract KiloPriceFeed does not need to import SafeMath library for overflow checking, which can save gas.
Hacker007 : Solidity versions >= 0.8.x perform checked arithmetic by default, so the SafeMath library is unnecessary in most cases.
ginlee : In Solidity version 0.8 and later, arithmetic operations have built-in overflow and underflow protection by default. The SafeMath library, which was commonly used in earlier versions of Solidity, is no longer necessary.

Recommendation

Yaodao : Consider removing SafeMath library.
Hacker007 : It is recommended to delete SafeMath from the contract.
ginlee : remove "using SafeMath for uint256" and "@openzeppelin/contracts/utils/math/SafeMath.sol"

Client Response

Fixed

KEX-83:Unused import files

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	Hacker007

Code Reference

- code/src/core/KiloStorageManager.sol#L7-L8

```
7:import "../interfaces/IOracle.sol";
8:import "../interfaces/IPerpTrade.sol";
```

Description

Hacker007 : The linked import statements include a contract or library which is not used in the file that imported such files.

Recommendation

Hacker007 : Remove the import statement that includes the library.

Client Response

Fixed

KEX-84: `require` check without messages

Category	Severity	Status	Contributor
Code Style	Informational	Fixed	Yaodao

Code Reference

- code/src/core/ProductManager.sol#L53
- code/src/core/ProductManager.sol#L54
- code/src/core/ProductManager.sol#L71
- code/src/core/ProductManager.sol#L72
- code/src/core/KiloStorageManager.sol#L166
- code/src/core/ProductManager.sol#L176

```
53:         require(_product.maxLeverage > _product.minLeverage && token != address(0));

54:         require(_product.minLeverage >= 1 * BASE);

71:         require(_product.maxLeverage > _product.minLeverage && token != address(0));

72:         require(_product.minLeverage >= 1 * BASE);

166:         require(_maxShift <= 0.01e8 && _minProfitTime <= 24 hours && _liquidationThreshold > 500
0 && _maxExposureMultiplier > 0);

176:         require(_discount <= MAX_ACCOUNT_DISCOUNT);
```

Description

Yaodao : These `require` checks only give the check condition without the error messages.

Recommendation

Yaodao : Recommend adding corresponding error messages.

Client Response

Fixed

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.