ANCILIA

KiloEx AUDITING REPORT

June 2023

Prepared for

KiloEx

Prepared by

Ancilia, Inc.

Index

Version History

Version	Description	Date
1.3(public)	Public version	July, 13 2023
1.2	Explained more for initialize() issue	July, 06 2023
1.1	Lower two cases severity, added the fix section	July, 05 2023
1.0	First draft version.	Jun, 28 2023

Executive Summary

The KiloEx team (Kilo) shared their smart contract source code via github. We have listed hashes of smart contracts to ensure the entirety of the audit can be tied to a given contract version. The Ancilia research team has worked with the Kilo team on all potential findings and issues. The audit scope includes checking for smart contracts with attack vulnerabilities such as re-entry attacks, logic flaws, authentication bypasses, DoS attacks, etc. Our researchers primarily focused on Kilo's trading core functionalities: trading and staking.

Disclaimer

Note that security audit services do not guarantee to find all possible security issues in the given smart contracts. A repeating code audit or incremental code audit is encouraged. Multiple audits with several auditors are recommended. Product owners are still required to have their own test cases and regular code review process. A threat intelligence system may help to discover or prevent a potential attack which can further reduce risk. Additionally, a bug bounty program for the community will help improve the security of products. Last but not least, Security is complicated! A strong smart contract does not guarantee your product is safe from all cybersecurity attacks.

Contracts overview

After compilation with Hardhat(2.2.1, solc version 0.8.17), there are a total of 22 smart contracts which are listed below(test and interface files are omitted) .

OperatorOwnerGovernableUpgradeable	access/OperatorOwnerGovernableUpgradeable.sol	6bb80ddd2f8c7d3beb4d15ca2ea3d9dcb071f7687b9cb51406f4394d90e36351
OwnerGovernableUpgradeable	access/OwnerGovernableUpgradeable.sol	703ce085a74b8c1248482210a392d4fcf968616b656355efd05ab2a6d585b3c2
OperatorOwnerGovernable	access/OperatorOwnerGovernable.sol	babb4d6a2d678245a60c475afba5c379a50852a998cae8d4350c5f5720b3999d
OwnerGovernable	access/OwnerGovernable.sol	f77af2f9d4aa71d7a048cfbe58bb15987196da3ed02bea88782e5fd68e41ea37
KiloPriceFeed	core/KiloPriceFeed.sol	2d9b8533636ed369c2d6350f3670b9efd286537115f95a9f40553c7d48eef975
KiloStorageManager	core/KiloStorageManager.sol	07688927f9c22f5439abcbde44c8f5848b1748a2cc484e8ad990b231b30fad28
MarginFeeManager	core/MarginFeeManager.sol	f304b5009ea3f4983546d02e370a2f7db554c7e53a0e8713b48125bf96caac63
OrderBook	core/OrderBook.sol	bff64f64e58fb38477ace2d012a62849ddc67b41def19b751b3e0da3caf3f5e5
PendingReward	core/PendingReward.sol	3a24cc22b8e642b278fe168f96e4895415049b621dd328008e469c0975a79c64
PerpTrade	core/PerpTrade.sol	a85242948cf671e18fc08d81d90a278108500a8e94323381e9469482612e9ad4
PositionRouter	core/PositionRouter.sol	67c24dd8bbe3f8e938fdb331df4d0640f70e8880e2768926b1a3fd9041e3783a
ProductManager	core/ProductManager.sol	4d1b44997537fd332dc9463c36fc7dc03a5338701380985b1e175ffc84eb4be2
VaultStakeReward	core/VaultStakeReward.sol	221adf8d2d74f61ddd0f136aa5d41221bc377f612c5c2f89b4238d2b115e8e55
KiloPassCard	passcard/KiloPassCard.sol	045edf259f88b66095a63625e184731260e70691477f7a87fc7fc1c6b5700caf
KeeperReader	peripherals/KeeperReader.sol	c142d739a7a03d7c3b3f4509012363c4093d53e9f1a3d55e6f3604313c62d7c7
KiloPerpView	peripherals/KiloPerpView.sol	9d4992195679375e27643fec2a33cb9396a9fa2c368d12860220a427e6053df0
LiquidationPriceReader	peripherals/LiquidationPriceReader.sol	b2cfea313cc4e018a16fa2d8871f89466908989af4eb6a9f43698594d482113b
ReferralReader	referrals/ReferralReader.sol	4b241fe33af6319d040f92b7d1ba8c5552d096d95df39a8217dfc59073d2f004
ReferralStorageManager	referrals/ReferralStorageManager.sol	f3ee2aa4a035a1daa28ff76c123e1c55fcd6f44036e20bfb27790d51fa0e72ab
ProtocolReward	tradereward/ProtocolReward.sol	1a2158ad0bd79f05f9f45d0b96ab14d65cad8386282feb138bbc09afe0ae4174
TradeRewardDistributor	tradereward/TradeRewardDistributor.sol	8c7afe54e3a212c96da96b82cd9129cff3b695694aca13ba9a36facb31927da6
PerpTradeUtil	libraries/PerpTradeUtil.sol	1e52aaa9729927350f9ff58f6f2aedce4faa937e3b608d7d897b1de6c7ebc0dc

The findings

Results

ID	Description	Severity	Product Impact	Status
Kilo-A-01	ERC 4626 Staking inflation	Critical	Medium	Fixed
Kilo-A-02	Partial decreasePosition leads to Full Liquidation	Medium	Medium	Fixed
Kilo-A-03	OpenInterest partial update leads protocol anomaly	High	High	Fixed
Kilo-A-05	Liquidation 2% price may lead to a profit gain	Critical	Critical	Fixed
Kilo-A-06	Implementation contracts have not been initialized	Medium	Medium	Fixed
Kilo-A-07	isTradeEnabled stop only for increasePosition	Info	Info	Fixed

Details

Kilo-A-01 [Critical] ERC 4626 Staking inflation

** We marked the product impact as "medium" is because of the already deployed contract which has initial _totalsupply = 1000*1e8 which makes the attack harder. The vulnerability still exists on this contract if Kilo deployed the vulnerable version to other chains.*

The function `deposit()` in the contract `VaultStakeReward` is vulnerable for the staking inflation attack. The initial deposit could be front-run and then the user's staking funds would be vulnerable to theft. This article explains the attack.

```
function deposit(uint256 amount, address user) public override
nonReentrant returns (uint256) {
  require(kiloConfig.canUserStake(), "Vault: not allowed");
  require(msg.sender == user, "Vault: not staker");
  uint256 _totalAssets = totalAssets();

    //KEX-1: make the attacker can not profit from future users'
deposits
  if (_totalAssets == 0 && amount < 1000e8) {
      revert("Vault: amount too small");
  }
  ....
}
```

The code has an initial amount check, which could make the attack harder. However, the check condition could be easily bypassed. For example, during a front-run, the attacker only needs to transfer 1 wei to the Vault and to make _totalAssets non-zero. Then the attacker would just need to deposit 1 USDT to make the denominator a smaller number. Thus causing an ERC 4626 inflation attack.

Suggestion: Call the first deposit in the initialize() function with 1000 USDT to prevent a front-run and the denominator number is big enough.

Fix: Check totalSupply() == 0 rather than check _totalAssets which cannot bypass.

Kilo-A-02 [Medium] Partial decreasePosition leads to Full Liquidation

The function decreasePositionWithId() in the contract core/PerpTrade.sol does not check the minimal value of the margin parameter, in certain cases the function PerpTradeUtil._getPnl() will return 0. Ultimately this leads to a full liquidation of the current position order.

```
function _getPnl(
.....
) internal pure returns(int256 _pnl) {
  bool pnlIsNegative;
  uint256 pnl;
  ....
  if (isLong) {
    if (price >= positionPrice) {
      pnl = margin * positionLeverage * (price - positionPrice) / positionPrice
/ BASE;
    } else {
      pnl = margin * positionLeverage * (positionPrice - price) / positionPrice
/ BASE;
      pnlIsNegative = true;
    }
  } else {
    .....
  }

  if (pnlIsNegative) {
    _pnl = -1 * int256(pnl);
  } else {
    _pnl = int256(pnl);
  }
  return _pnl;
}
```

In the function decreasePositionWithId(), the final pnl value will be calculated based on the _getPnl() result and the accumulated funding value:

```
vars.pnl = PerpTradeUtil._getPnl(position.isLong, uint256(position.price),
uint256(position.leverage), margin, vars.price) - vars.fundingPayment -
int256(vars.borrowingFee);
```

A well constructed margin value could lead vars.pnl to be a negative value. This negative value will fully liquidate the current position order.

```
if (vars.pnl < 0 && uint256(- 1 * vars.pnl) >= margin *
    kiloConfig.liquidationThreshold / (10 ** 4)) {
```

```
  margin = uint256(position.margin);
  vars.pnl = - 1 * int256(uint256(position.margin));
  vars.isLiquidatable = true;
}
```

The full amount of `position.margin` will be part of `pendingPnl` and it will be transferred to the Vault. The total assets number of the vault will be updated and cause a more profitable `redeem()` price.

Suggestion: check return value of _getPnl(). Furthermore, it is suggested that checking of a minimum margin value is performed.

Fix: Added require condition on the minimal margin value.

Kilo-A-03 [High] OpenInterest partial update leads protocol anomaly

In the Kilo-A-02 case, we described a partial margin decrease that will trigger full liquidation. In the function decreasePositionWithId(), it will call updateDecreaseOpenInterest() on the provided margin * leverage. The margin could be a partial amount and thus the decrease in OpenInterest is on the partial amount as well.

```
kiloStorage.updateDecreaseOpenInterest(product.productId, margin *
uint256(position.leverage) / BASE, position.isLong);
```

Since the entire position amount will be liquidated, there is no way to reset the OpenInterest value accordingly. Thus an unexpected balance in longOpenInterests or shortOpenInterests leads to the protocol being in an anomalous state.

Suggestion: Once full liquidation is decided, OpenInterest must be updated accordingly

Fix: Added the updating code.

Kilo-A-05 [Critical] Liquidation 2% price may lead to a profit gain

The function adlDecreasePosition() in the contract core/PerpTrade.sol allows the liquidator to set an arbitrary price on the token. Although there is a 2% cap on the price difference, this still provides potential profits for an attacker. If an attacker sees a 2% price update change in the mempool, the contract PositionRouter can be called to set a position before the price change. It could be a long or short position depending on the price difference. After the price has been set, it could call decreasePosition() to guarantee the profit gain.

Suggestion: Don't put liquidation price into price feed, use it as a one time price in memory only.

Fix: Removed the permission which allows the user to create a position. Added check that only allows the contract itself to call the function and guarantee there must be a price override before creating a new position.

Kilo-A-06 [Medium] Implementation contracts not been initialized

For all Kilo's upgradable contracts, the _disableInitializers() is missing. It can potentially leave the contract uninitialized. An attacker could initialize the contract and everage this on a phishing attack.

More details on this: if a contract inherited class OwnerGovernableUpgradeable and if you called __owner_governable_init() in initialize() function, then there is no issue because __owner_governable_init will check if it is running in the constructor(). Kilo's contracts do not ALL follow this pattern and they are still vulnerable.

Suggestion: add _disableInitializers() each upgradable contracts:

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

Fix: Added the code.

Kilo-A-07 [Info] isTradeEnabled stop only for increasePosition

The global variable `isTradeEnabled` will control the `increasePosition()` function only. The `decreasePositionWithId()` could still work even if trading is stopped. In an urgent case, `decreasePositionWithId()` may need to be controlled by this variable as well.

Suggestions: As this will be part of business strategy, please be clear about the potential risk.

Fix: Added the check in `decreasePositionWithId()`

Summary

Ancilia team has performed both an automated and manual code audit on the Kilo smart contracts mentioned above. All issues have been shared with the Kilo team through a telegram channel before this report. Overall, 2 critical, 1 high, 2 medium impact issues have been discovered through this audit. Kilo team reacted pretty quickly and fixed all the issues. Ancilia team verified and confirmed the fixes are in the github.