HAECHI AUDIT

PerpDEX

Smart Contract Security Analysis

Published on : July 6, 2022

Version v2.0

HAECHI AUDIT

Smart Contract Audit Certificate

PerpDEX

Security Report Published by HAECHI AUDIT
v1.0 June 29, 2022
v2.0 July 6, 2022

Auditor : Louis Noh

[illegible]

Executive Summary

Severity of Issues	Findings	Resolved	Unresolved	Acknowledged	Comment
Critical	2	-	2	2	-
Major	1	1	-	-	All issues resolved
Minor	1	-	1	-	-
Tips	2	-	-	-	-

TABLE OF CONTENTS

6 Issues (2 Critical, 1 Major, 1 Minor, 2 tips) Found

ABOUT US

HAECHI AUDIT believes in the power of cryptocurrency and the next paradigm it will bring.

We have the vision to *empower the next generation of finance*. By providing security and trust in the blockchain industry, we dream of a world where everyone has easy access to blockchain technology.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. HAECHI AUDIT provides specialized and professional smart contract security auditing and development services.

We are a team of experts with years of experience in the blockchain field and have been trusted by 300+ project groups. Our notable partners include Universe,1inch, Klaytn, Badger, etc.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io

Website: audit.haechi.io

INTRODUCTION

This report was prepared to audit the security of the smart contract created by the PerpDEX team. HAECHI AUDIT conducted the audit focusing on whether the smart contract created by the PerpDEX team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the smart contract.

CRITICAL	Critical issues must be resolved as critical flaws that can harm a wide range of users.
MAJOR	Major issues require correction because they either have security problems or are implemented not as intended.
MINOR	Minor issues can potentially cause problems and therefore require correction.
TIPS	Tips issues can improve the code usability or efficiency when corrected.

HAECHI AUDIT recommends that the Ithil team improve all issues discovered.

The following issue explanation uses the format of {file name}#{line number}, {contract name}#{function/variable name} to specify the code. For instance, *Sample.sol:20* points to the 20th line of Sample.sol file, and *Sample#fallback()* means the fallback() function of the Sample contract.

Please refer to the Appendix to check all results of the tests conducted for this report.

SUMMARY

The codes used in this Audit can be found on the following repository and commit hash.

Repository :
https://github.com/HAECHI-LABS/audit-PerpDEX/tree/e8ad278cd8ea9b624b8648e02ca8a8da10943e10

Commit hash : e8ad278cd8ea9b624b8648e02ca8a8da10943e10

Issues	HAECHI AUDIT found 2 critical issues, 1 major issue, and 1 minor issue. There are 2 Tips issues explained that would improve the code's usability or efficiency upon modification.
Update	[v2.0] - In a new commit "3be7f6655f4fd80cfae6a2e05b4f38e8dc1d84b9", 2 Critical Issues has been left as acknowledgment. 1 major issue has been resolved. 1 minor issue has been unresolved.

Severity	Issue	Status
CRITICAL	ShareMarkPricex96 which is calculated by the onchain data can be manipulated.	(Found - v1.0) (Ack - v2.0)
CRITICAL	Amount of the minted asset should not be decided by onchain data.	(Found - v1.0) (Ack - v2.0)
MAJOR	The accountValue can not be calculated properly, in case several markets use the different pairs of base and quote.	(Found - v1.0) (Resolved - v2.0)
MINOR	Some ERC20 tokens require a fee when transferring the balance.	(Found - v1.0) (Unresolved - v2.0)
TIPS	PerpdexLongToken cannot guarantee Stable.	(Found - v1.0)
TIPS	The settlement token should be setted to a safe token	(Found - v1.0)

OVERVIEW

Contracts subject to audit

- IPerpdexMarketMinimum.sol
- IERC20Metadata.sol
- IWETH9.sol
- IPerpdexMarket.sol
- IPerpdexExchange.sol
- IPerpdexPriceFeed.sol
- Math.sol
- PerpdexMarket.sol
- VaultLibrary.sol
- MakerLibrary.sol
- AccountLibrary.sol
- PerpMath.sol
- PerpdexStructs.sol
- FundingLibrary.sol
- TakerLibrary.sol
- PoolLibrary.sol
- MarketStructs.sol
- PriceLimitLibrary.sol
- PerpdexExchange.sol
- PerpdexTokenBase.sol
- IERC20Metadata.sol
- IERC4626.sol
- PerpdexLongToken.sol
- IStdReference.sol
- IDIAOracleV2.sol
- IPerpdexPriceFeed.sol
- UniswapV3PriceFeed.sol
- BandPriceFeed.sol
- ChainlinkPriceFeed.sol
- DiaPriceFeed.sol

The PerpDEX Smart contract has the following privileges.

- Owner
- OnlyExchange

The specifications of the control of each privileges are as follows.

Role	Functions
Owner	❖ *PerpdexExchange#setIsMarketAllowed()* ❖ *PerpdexExchange#setProtocolFeeRatio()* ❖ *PerpdexExchange#setLiquidationRewardConfig()* ❖ *PerpdexExchange#setMmRatio()* ❖ *PerpdexExchange#setImRatio()* ❖ *PerpdexExchange#setMaxMarketsPerAccount()* ❖ *PerpdexExchange#transferProtocolFee()* ❖ *PerpdexExchange#transferInsuranceFund()* ❖ *PerpdexMarket#setPriceLimitConfig()* ❖ *PerpdexMarket#setFundingRolloverSec()* ❖ *PerpdexMarket#setFundingMaxElapsedSec()* ❖ *PerpdexMarket#setFundingMaxPremiumRatio()* ❖ *PerpdexMarket#setPoolFeeRatio()* ❖ *PerpdexMarket#setBaseURI()*
OnlyExchange	❖ *PerpdexMarket#removeLiquidity()* ❖ *PerpdexMarket#addLiquidity()* ❖ *PerpdexMarket#swap()*

FINDINGS

CRITICAL

ShareMarkPricex96 which is calculated by the onchain data can be manipulated

(Found - v.1.0) (Ack - v.2.0)

```
function getShareMarkPriceX96(uint256 base, uint256 quote) internal pure returns (uint256) {
        return FullMath.mulDiv(quote, FixedPoint96.Q96, base);
}
```

[https://github.com/HAECHI-LABS/audit-PerpDEX/blob/083c7dc084181995c2c91e88e90e9504f5f07b35/contractv1/core/contracts/contracts/lib/PoolLibrary.sol#L268]

Issue

In *PerpdexExchange#trade()*, swap is made in a market depending on the amount of trade. The price in swap is decided by the ratio of the amount of base and quote in the pool using *PoolLibrary#getShareMarkPriceX96()*. That is, ShareMarkPriceX96 depends on the onchain data. It is vulnerable to massive asset trade.

Recommendation

The Price should not be dependent on the onchain data.

Acknowledgement

It has not been modified by the development team’s intention.

CRITICAL

Amount of the minted asset should not be decided by onchain data.

(Found - v.1.0) (Ack - v.2.0)

```
function mintETH(uint256 shares, address receiver) external payable onlyWeth nonReentrant returns
(uint256 assets) {
        assets = previewMint(shares);
        uint256 exceeded = msg.value.sub(assets);
        if (exceeded > 0) {
            msg.sender.transfer(exceeded);
        }
        _doMint(assets, shares, receiver);
}

function mint(uint256 shares, address receiver) external override nonReentrant returns (uint256
assets) {
        assets = previewMint(shares);
        _transferAssetFromSender(assets);
        _doMint(assets, shares, receiver);
}
```

[https://github.com/HAECHI-LABS/audit-PerpDEX/blob/e8ad278cd8ea9b624b8648e02ca8a8da10943e10/contractv1/stable-coin/contracts/PerpdexLongToken.sol#L41-L54]

Issue

In *PerpdexLongToken#mintETH()*, *PerpdexLongToken#mint()*, the amount of asset to receive is calculated by *PerpdexLongToken#previeMint()*. However, *PerpdexLongToken#previeMint()* decides the amount of asset to transfer based on the amount of base and quote in a pool. It means that it depends on onchain data. Thus, it can be affected by massive asset trading.

Recommendation

The amount of asset to receive should not be dependent on the onchain data.

Acknowledgement

It has not been modified by the development team's intention.

⚠ **MAJOR**

The accountValue can not be calculated properly, in case several markets use the different pairs of base and quote.

(Found - v.1.0) (Resolved - v.1.0)

```
function getTotalAccountValue(PerpdexStructs.AccountInfo storage accountInfo) internal view returns
(int256) {
        address[] storage markets = accountInfo.markets;
        int256 accountValue = accountInfo.vaultInfo.collateralBalance;
        uint256 length = markets.length;
        for (uint256 i = 0; i < length; ++i) {
            address market = markets[i];

            PerpdexStructs.MakerInfo storage makerInfo = accountInfo.makerInfos[market];
            int256 baseShare = accountInfo.takerInfos[market].baseBalanceShare;
            int256 quoteBalance = accountInfo.takerInfos[market].quoteBalance;

            if (makerInfo.liquidity != 0) {
                (uint256 poolBaseShare, uint256 poolQuoteBalance) =
                IPerpdexMarketMinimum(market).getLiquidityValue(makerInfo.liquidity);
                (int256 deleveragedBaseShare, int256 deleveragedQuoteBalance) =
                IPerpdexMarketMinimum(market).getLiquidityDeleveraged(
                        makerInfo.liquidity,
                        makerInfo.cumBaseSharePerLiquidityX96,
                        makerInfo.cumQuotePerLiquidityX96
                        );
                baseShare = baseShare.add(poolBaseShare.toInt256()).add(deleveragedBaseShare);
                quoteBalance =
quoteBalance.add(poolQuoteBalance.toInt256()).add(deleveragedQuoteBalance);
        }

        if (baseShare != 0) {
                uint256 sharePriceX96 = IPerpdexMarketMinimum(market).getShareMarkPriceX96();
                accountValue = accountValue.add(baseShare.mulDiv(sharePriceX96.toInt256(),
FixedPoint96.Q96));
        }
            accountValue = accountValue.add(quoteBalance);
    }
    return accountValue;
}
```

[https://github.com/HAECHI-LABS/audit-PerpDEX/blob/083c7dc084181995c2c91e88e90e9504f5f07b35/contractv1/core/contracts/contracts/lib/AccountLibrary.sol#L49]

Issue

AccountLibrary#getTotalAccountValue adds a pair of base and quote of each market allowed in the account. If there is a market using a pair of base and quote or every market takes the same pairs of base and quote, there won't be a big problem. But, pairs of base and quote in all markets cannot be the same.

Suppose that there are markets, one is BTC/USDT and the other is ETH/USDT. The accountValue is added up by BTC and ETH of the account. As you can see, simple addition of BTC and ETH does not represent the value of the account's holding.

Recommendation

Please revise the simple addition so that the holding value of the account can be properly reflected in markets with different pair of base and quote such as BTC/USDT and ETH/USDT.

Update

[v2.0] - The issue is resolved by adding the logic to validate market's exchange in *PerpdexExchange#setIsMarketAllowed()*.

MINOR

Some ERC20 tokens require a fee when transferring the balance.

(Found - v.1.0) (Unresolved - v.1.0)

```
function _transferTokenIn(
        address token,
        address from,
        uint256 amount
        ) private {
        // check for deflationary tokens by assuring balances before and after transferring to be
the same
        uint256 balanceBefore = IERC20Metadata(token).balanceOf(address(this));
        SafeERC20.safeTransferFrom(IERC20(token), from, address(this), amount);
        require(
            (IERC20Metadata(token).balanceOf(address(this)).sub(balanceBefore)) == amount,
            "VL_TTI: inconsistent balance"
            );
}
[https://github.com/HAECHI-LABS/audit-PerpDEX/blob/e8ad278cd8ea9b624b8648e02ca8a8da10943e10/contract
v1/core/contracts/contracts/lib/VaultLibrary.sol#L89]
```

Issue

In *VaultLibrary#_transferTokenIn*, the contract receives ERC20 tokens. In a case a ERC20 token does not require a transfer fee, 'IERC20Metadata(token).balanceOf(address(this)).sub(balanceBefore)) == amount' is right. However, some ERC20 tokens do not. That is, the require statement does not work properly.

Recommendation

Please revise the require statement to work properly when some ERC20 tokens require additional fee.

💡 **TIPS**

PerpdexLongToken cannot guarantee Stable.

(Found - v.1.0)

Issue

PerpdexLongToken may have properties similar to stable coin. But, these properties cannot guarantee that PerpdexLongToken is stable.

💡 **TIPS**

The settlement token should be setted to a safe token.

(Found - v.1.0)

```
function _getERC20Name(
        address marketArg,
        string memory prefix,
        string memory nativeTokenSymbol
        ) private view returns (string memory) {
        address settlementToken =
IPerpdexExchange(IPerpdexMarket(marketArg).exchange()).settlementToken();

        return
        string(
                abi.encodePacked(
                    prefix,
                    IPerpdexMarket(marketArg).symbol(),
                    settlementToken == address(0) ? nativeTokenSymbol :
IERC20Metadata(settlementToken).symbol()
                    )
            );
}
[https://github.com/HAECHI-LABS/audit-PerpDEX/blob/e8ad278cd8ea9b624b8648e02ca8a8da10943e10/contract
v1/stable-coin/contracts/PerpdexTokenBase.sol#L282]
```

Issue

PerpdeTokenBase#_getERC20Name returns the name of the token. However, some tokens have the name as bytes32. In case, it causes unexpected behavior to the contract. Thus, the settlement token should be setted to a safe token.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix A. Test Results

The following results show the unit test results covering the key logic of the smart contract subject to the security audit. Parts marked in red are test cases that failed to pass the test due to existing issues.

```
AccountLibrary
  #updateMarkets()
    ✔ should update Markets (445ms)
  #isLiquidationFree()
    ✔ should be done when liquidity is not 0 (45ms)

 FundingLibrary
  #_calcPremiumX96()
    ✔ should not be reverted

 PerpdexExchange
  #constructor()
    ✔ should be reverted by PE_C: token address invalid
    ✔ should set to usdc addr (112ms)

    #modifier checkDeadline()
      ✔ should be reverted by PE_CD: too late (228ms)
    #checkMarketAllowed()
      ✔ should be reverted by PE_CMA: market not allowed (218ms)
    #deposit()
      ✔ should be reverted by PE_D: amount not zero (103ms)
      ✔ should be emitted by Deposited(trader, msg.value) (102ms)
      ✔ should be reverted by PE_D: msg.value not zero
      ✔ should be emitted by Deposited(trader, amount)
      ✔ should be reverted by VL_D: zero amount
    #withdraw()
      ✔ should be equal to 800 (57ms)
      ✔ should be reverted by VL_W: zero amount
    #transferInsuranceFund()
      ✔ should be emitted by InsuranceFundTransferred
    #transferProtocolFee()
      ✔ shoud be emitted by ProtocolFeeTransferred
    #trade()
      ✔ should be emitted by PositionChanged (217ms)
      ✔ should be emitted by PositionChanged, true, true (205ms)
      ✔ should be emitted by PositionChanged, false, true (204ms)
      ✔ should be emitted by PositionChanged, true, false (211ms)
      ✔ should be emitted by PositionChanged return 0 when, protocolFeeRatio is not 0 (209ms)
      ✔ should be reverted by TL_OP: enough mm (197ms)
```

```
#addLiquidity()
  ✔ should be emitted by LiquidityAdded (81ms)
#removeLiquidity()
  ✔ should be emitted by LiquidityRemoved (188ms)
#setMaxMarketsPerAccount()
  ✔ should return 10
#setImRatio()
  ✔ should be reverted by PE_SIR: too large
  ✔ should be reverted by PE_SIR: smaller than mmRatio
  ✔ should set imRatio, to 10
#setMmRatio()
  ✔ should be reverted by PE_SMR: bigger than imRatio
  ✔ should be reverted by PE_SMR: zero
  ✔ should set mmRatio to 1
#setLiquidationRewardConfig()
  ✔ should be emitted by LiquidationRewardConfigChanged
  ✔ should be reverted by PE_SLRC: too large reward ratio
  ✔ should be reverted by PE_SLRC: ema time is zero
#setProtocolFeeRatio()
  ✔ should be reverted by PE_SPFR: too larg
  ✔ should set protocolFeeRatio to 1
#setIsMarketAllowed()
  ✔ should be reverted by PE_SIMA: market address invalid
  ✔ should be emitted by IsMarketAllowedChanged
#previewTrade()
  ✔ should return 102 (339ms)
  ✔ should return not 101 when protocolfee is not 0 (329ms)
  ✔ should be reverted by TL_OPD: enough mm (243ms)
  - should return 102, true, true
  ✔ should return 102, true, false (333ms)
  - should return 102, false, true
#maxTrade()
  ✔ should return not 0 (330ms)
  ✔ should return 0
  ✔ should return not 0, isBaseToQuete is true (415ms)
  ✔ should return not 0, is ExactInput is true, isBaseToQuete is true (416ms)
  ✔ should return not 0, is ExactInput is true, isBaseToQuete is false (398ms)
  ✔ should return 0, isSelf and isLiquidation are false (377ms)
  ✔ should return not 0, when protocolfeeratio is 0 (419ms)
#getTotalAccountValue()
  ✔ should not return 0 (380ms)
#getPositionShare()
  ✔ should not return 0 (347ms)
#getPositionNotional()
  ✔ should not return 0 (368ms)
#getTotalPositionNotional()
  ✔ should not return 0 (350ms)
#getOpenPositionShare()
  ✔ should not return 0 (362ms)
```

```
    #getOpenPositionNotional()
      ✔ should not return 0 (366ms)
    #getTotalOpenPositionNotional()
      ✔ should not return 0 (367ms)
    #hasEnoughMaintenanceMargin()
      ✔ should not return 0 (375ms)
    #hasEnoughInitialMargin()
      ✔ should not return 0 (399ms)

  PerpdexMarket
    #constructor()
      ✔ should be reverted by PM_C: base price feed invalid
      ✔ should be reverted by PM_C: quote price feed invalid

    #swap()
      ✔ should be reverted by PM_S: too large amount
      ✔ should be emitted by Swapped (49ms)
      ✔ should be emitted by Swapped, true, false (47ms)
      ✔ should be emitted by Swapped, true, true (48ms)
      ✔ should be emitted by Swapped, false,true (48ms)
    #addLiquidity()
      ✔ should be emitted by LiquidityAdded
    #removeLiquidity()
      ✔ should be emitted by removeLiquidity (47ms)
    #setPoolFeeRatio()
      ✔ should be reverted by PM_SPFR: too large
      ✔ should return 1e4
    #setFundingMaxPremiumRatio()
      ✔ should be reverted by PM_SFMPR: too large
      ✔ should return 10
    #setFundingMaxElapsedSec()
      ✔ should be reverted by PM_SFMES: too large
      ✔ should return 1
    #setFundingRolloverSec()
      ✔ should be reverted by PM_SFRS: too large
      ✔ should be reverted by PM_SFRS: too small
      ✔ should be emitted by FundingRolloverSecChanged
    #setPriceLimitConfig()
      ✔ should be reverted by PE_SPLC: too large liquidation
      ✔ should be reverted by PE_SPLC: invalid
      ✔ should be reverted by PE_SPLC: ema too large liq
      ✔ should be reverted by PE_SPLC: ema invalid
      ✔ should set 0
    #previewSwap()
      ✔ should be reverted by PM_PS: too large amount
      ✔ should return not 0 (70ms)
    #maxSwap()
      ✔ should return not 0 (69ms)
      ✔ should return not 0, true, false (66ms)
      ✔ should return not 0, true, true (66ms)
      ✔ should return not 0, false, true (65ms)
    #getLiquidityValue()
      ✔ should return 10000 (53ms)
    #getLiquidityDeleveraged()
      ✔ should return -10000 (56ms)
    #getCumDeleveragedPerLiquidityX96()
      ✔ should return 0 (51ms)
    #baseBalancePerShareX96()
      ✔ should return 79228162514264337593543950336
    #getMarkPriceX96()
      ✔ should return 79228162514264337593543950336 (53ms)

  PerpMath
    #formatSqrtPriceX96ToPriceX96
      ✔ should return 0
      ✔ should return 12
    #formatX10_18ToX96()
      ✔ should return 79228162514264337593543950336
    #formatX96ToX10_18()
      ✔ should return 1^18

BandPriceFeed
    #contructor()
      ✔ should return stdref address
      ✔ should be reverted by BPF_C: ref address not contract
    #getPrice()
      ✔ should return 10 (39ms)
    #decimals()
      ✔ should return 18
    #_getReferenceData()
      ✔ should be reverted by BPF_GRD: quote time is zero
      ✔ should be reverted by BPF_GRD: base time is zero
      ✔ should be reverted by BPF_GRD: invalid price

  ChainlinkPriceFeed
    #contructor()
      ✔ should return aggregator address
      ✔ should be reverted by CPF_C: agg address not contract
    #decimals()
      ✔ should return decimals, 18
    #getPrice()
      ✔ should return lastestPrice, 1
    #_requireEnoughHistory()
      ✔ should be reverted by CPF_REH: no enough history
    #_getRoundData()
      ✔ should not be reverted even latestPrice is 0

  DiaPriceFeed
    #contructor()
      ✔ should return stdref address
      ✔ should be reverted by BPF_C: ref address not contract
    #getPrice()
      ✔ should return 100
    #decimals()
      ✔ should return 8
    #_getOracleData()
      ✔ should be reverted by DPF_GOD: time is zero
      ✔ should be reverted by DPF_GOD: invalid price

  UniswapV3PriceFeed
    #contructor()
      ✔ should return pool address
      ✔ should be reverted by UVPF_C: pool is not contract
    #getPrice()
      ✔ should return 0
    #decimals()
      ✔ should return 18

PerpdexLongToken
    #constructor()
      ✔ should return symbol, plUSDtest
    #depositETH()
      ✔ should be emitted by Deposit
    #mintETH()
      ✔ should be emitted by Deposit
    #withdrawETH()
      ✔ should be emitted by Withdraw (195ms)
    #redeemETH()
      ✔ should be emitted by Withdraw (188ms)
    #previewDeposit()
      ✔ should not be equal (220ms)
    #previewMint()
      ✔ should be equal (151ms)
    #previewWithdraw()
      ✔ should not equal (331ms)
    #previewRedeem()
      ✔ should not equal (337ms)
    #maxDeposit()
      ✔ should not be reverted
    #maxMint()
      ✔ should not be reverted
    #maxWithdraw()
      ✔ should not be reverted
    #maxRedeem()
      ✔ should not be reverted

  PerpdexLongToken
```

```
  #constructor()
    ✔ should be reverted by PTB_C: weth can not be used
  #deposit()
    ✔ should be emitted by Deposit
  #mint()
    ✔ should be emitted by Deposit (95ms)
  #redeem()
    ✔ should be emitted by Withdraw (212ms)

PerpdexTokenBase
  #constructor()
    ✔ should return symbol, plUSDtest
    ✔ should be reverted by PTB_C: weth is required
  #transfer()
    ✔ should be equal to zero (110ms)
  #approve()
    ✔ should set allowance maxUint256
  #transferFrom()
    - should trasnfer Longtoken
  #increaseAllowance()
    ✔ should return 100
  #decreaseAllowance()
    ✔ should return 50
  #totalAssets()
    ✔ should return totalAsset (94ms)
  #convertToShares()
    ✔ should return 10000
  #convertToAssets()
    ✔ should return 10000
```

End of Document