BEOSIN
Blockchain Security

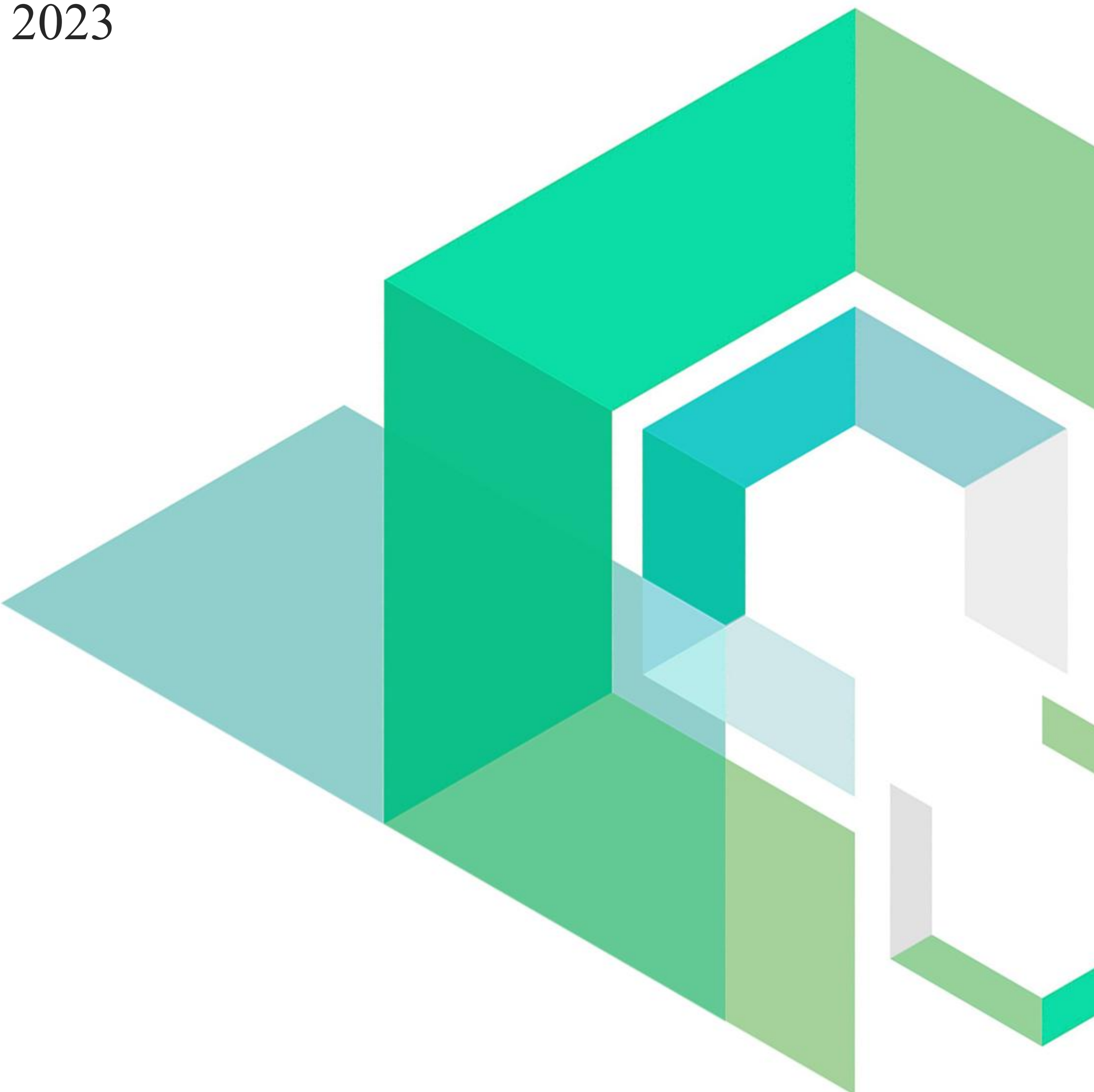
DIPX

Smart Contract Security Audit

V1.0

No. 202303221900

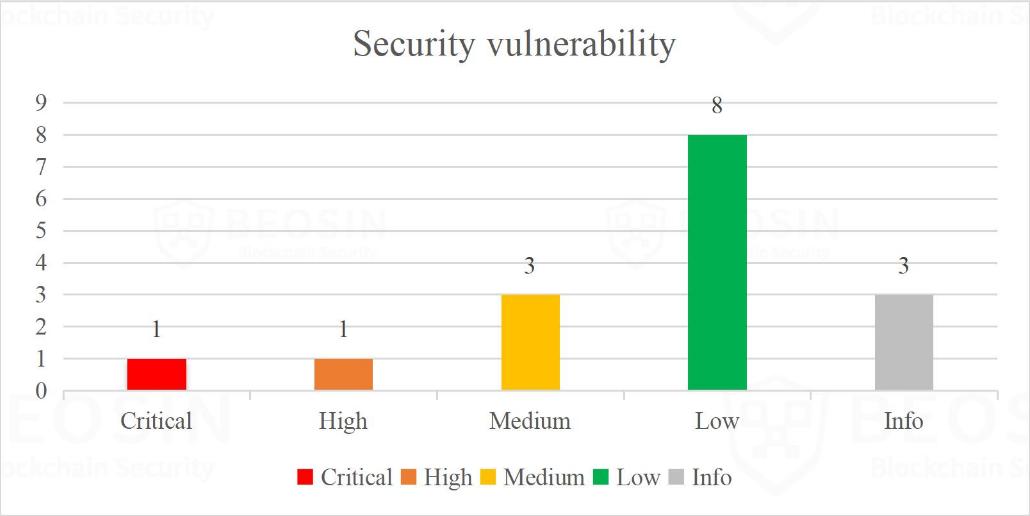
Mar 22th, 2023



Contents

Summary of Audit Results

After auditing, 1 Critical, 1 High, 3 Medium, 8 Low and 3 Info-risk items were identified in the DIPX project. Specific audit details will be presented in the **Findings** section. Users should pay attention to the following aspects when interacting with this project：



*Notes:

- **Risk Description:**

1. The owner can arbitrarily modify the oracle address and transfer the funds within the Vault contract. The project party stated that it will use multi-signature contracts and TimeLock contracts to manage owner permissions.

2. If the collateral token's price can only be queried from the AMM oracle of UniswapV2, the credibility of the oracle will be reduced.

3. Mixed pools do not support deflationary tokens.

- **Project Description:**

1. Business overview

DIPX is a decentralized trading platform built using oracle technology that supports on-chain derivatives index futures trading and perpetual contracts. DIPX operates on three different blockchain platforms: Arbitrum, Polygon, and Optimism. Users can obtain the corresponding LP tokens of the corresponding pool by staking in the corresponding pledge pool. Subsequently, users can choose to hold the LP tokens to receive transaction fees from subsequent liquidity providers or to receive reserve dividends from users whose LP tokens have been burned. Similarly, users can choose to trade index futures with their LP tokens (currently supporting leverage of up to 100 times), long or short based on the fluctuation of the target IndexToken price, and obtain benefits. When users are reducing positions or closing positions, the contract will mint or burn the corresponding LP tokens based on the user's current profit or loss.

1 Overview

1.1 Project Overview

Project Name	DIPX
Platform	Arbitrum, Polygon and Optimism

File Hash(SHA-256)	aea9dc6be09c0b61e25f5b46afa2294705092d379c9b9b57492f82c4475aef38(origin)
	b2d2ffa667c12bdd54f2cef7b158e5b6d40a8ace7c6b6cbe9e35cab90dc61ca5(final)

Audit scope	audit-contract\interfaces audit-contract\libraries audit-contract\oracle audit-contract\periphery audit-contract\referrals audit-contract\token\interfaces audit-contract\token\DIPX.sol audit-contract\token\DLP.sol audit-contract\token\MixedLP.sol audit-contract\token\SingleLP.sol audit-contract\DipxStorage.sol audit-contract\Handler.sol audit-contract\LpManager.sol audit-contract\OrderBook.sol audit-contract\PositionManager.sol audit-contract\Router.sol audit-contract\Vault.sol

1.2 Audit Overview

Audit work duration: Feb 27, 2023 – Mar 22, 2023

Audit methods: Formal Verification, Static Analysis, Typical Case Testing and Manual Review.

Audit team: Beosin Security Team.

2 Findings

Index	Risk description	Severity level	Status
DIPX-1	Incorrect precision conversion	Critical	Fixed
DIPX-2	Integer Overflow	High	Fixed
DIPX-3	Excessive permissions.	Medium	Acknowledged
DIPX-4	Default leverage too high	Medium	Acknowledged
DIPX-5	Too long array may cause gas exhaustion	Medium	Fixed
DIPX-6	Unlimited fee	Low	Fixed
DIPX-7	Unreasonable calculation in Pnlsupply	Low	Fixed
DIPX-8	Return parameter error	Low	Fixed
DIPX-9	Platform currency locked	Low	Fixed
DIPX-10	Insecure transfer function	Low	Fixed
DIPX-11	The fee parameter is incorrect	Low	Fixed
DIPX-12	Array lengths are misaligned in multiple places	Low	Fixed
DIPX-13	Inconsistent Deflationary Token Reserve	Low	Fixed
DIPX-14	Front-running risk	Info	Acknowledged
DIPX-15	Referral contract token transfer risk	Info	Acknowledged
DIPX-16	Visibility exception	Info	Fixed

Status Notes:

1. DIPX-3 not fixed, allowing the owner to arbitrarily modify the oracle address and transfer funds within the Vault contract.

2. DIPX-4 not fixed, which may cause delayed liquidation if the currency used for IndexToken is relatively obscure.

3. DIPX-14 not fixed and does not cause any impact.

4. DIPX-15 not fixed and does not cause any impact.

Finding Details:

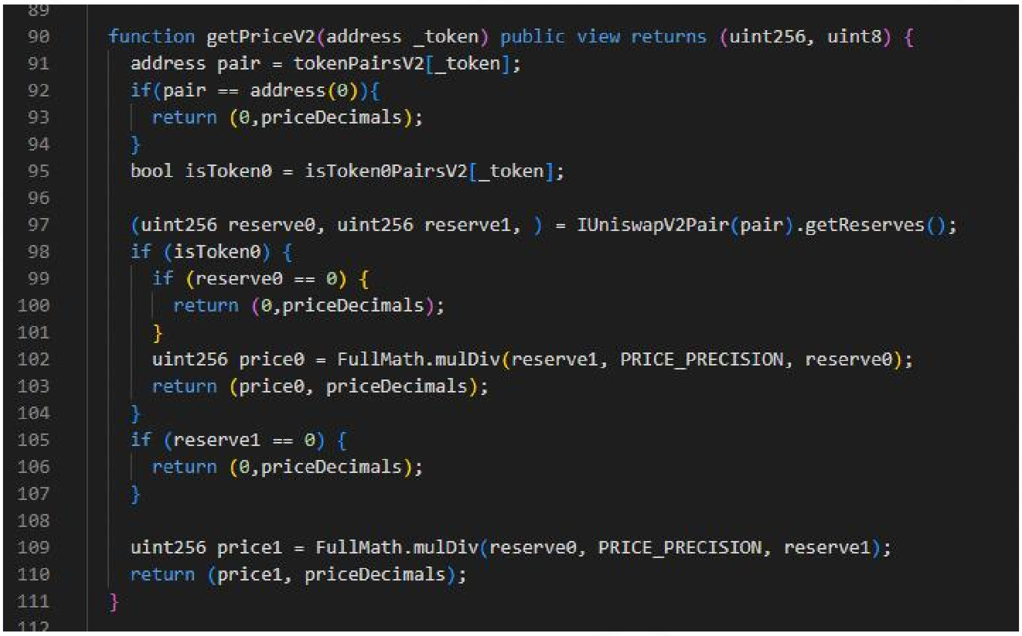
[DIPX-1] Incorrect precision conversion

Severity Level Critical

Type Business Security

Lines AmmPriceFeed.sol#L90-110

Description In the AmmPriceFeed.sol contract, the *getPriceV2* function may result in incorrect price calculation if the Decimals of token0 and token1 are different.

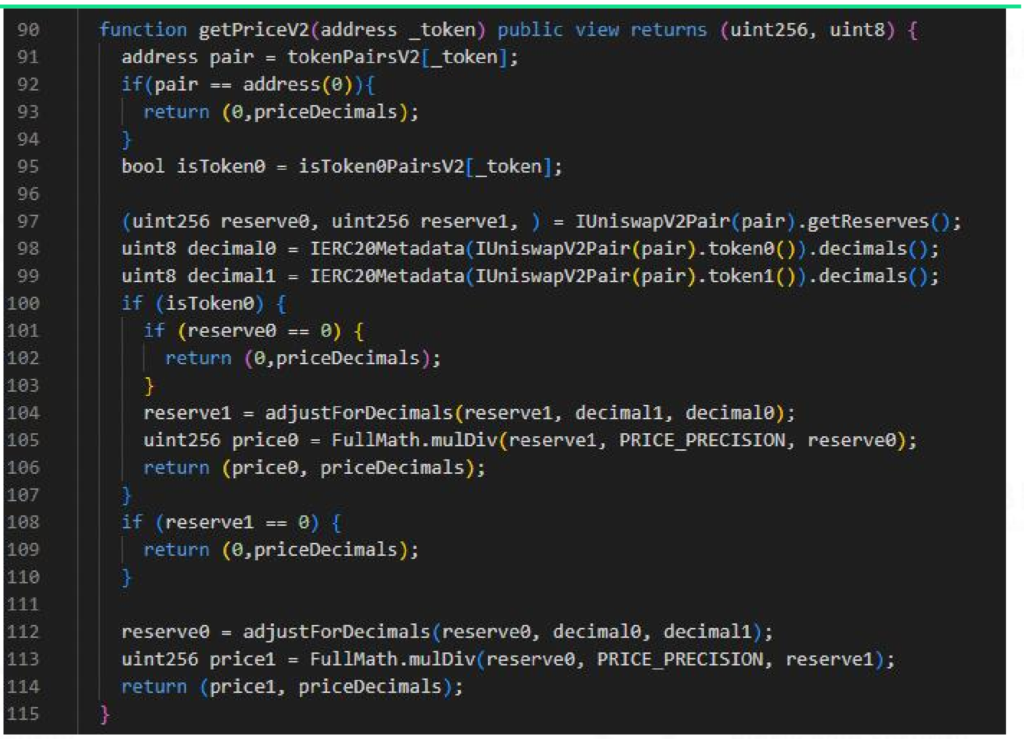


```
 89
 90  function getPriceV2(address _token) public view returns (uint256, uint8) {
 91    address pair = tokenPairsV2[_token];
 92    if(pair == address(0)){
 93      return (0,priceDecimals);
 94    }
 95    bool isToken0 = isToken0PairsV2[_token];
 96
 97    (uint256 reserve0, uint256 reserve1, ) = IUniswapV2Pair(pair).getReserves();
 98    if (isToken0) {
 99      if (reserve0 == 0) {
100        return (0,priceDecimals);
101      }
102      uint256 price0 = FullMath.mulDiv(reserve1, PRICE_PRECISION, reserve0);
103      return (price0, priceDecimals);
104    }
105    if (reserve1 == 0) {
106      return (0,priceDecimals);
107    }
108
109    uint256 price1 = FullMath.mulDiv(reserve0, PRICE_PRECISION, reserve1);
110    return (price1, priceDecimals);
111  }
112
```

Figure 1 Source code of *getPriceV2* function(Unfixed)

Recommendations It is recommended to perform precision conversion.

Status Fixed.



```
function getPriceV2(address _token) public view returns (uint256, uint8) {
  address pair = tokenPairsV2[_token];
  if(pair == address(0)){
    return (0,priceDecimals);
  }
  bool isToken0 = isToken0PairsV2[_token];

  (uint256 reserve0, uint256 reserve1, ) = IUniswapV2Pair(pair).getReserves();
  uint8 decimal0 = IERC20Metadata(IUniswapV2Pair(pair).token0()).decimals();
  uint8 decimal1 = IERC20Metadata(IUniswapV2Pair(pair).token1()).decimals();
  if (isToken0) {
    if (reserve0 == 0) {
      return (0,priceDecimals);
    }
    reserve1 = adjustForDecimals(reserve1, decimal1, decimal0);
    uint256 price0 = FullMath.mulDiv(reserve1, PRICE_PRECISION, reserve0);
    return (price0, priceDecimals);
  }
  if (reserve1 == 0) {
    return (0,priceDecimals);
  }

  reserve0 = adjustForDecimals(reserve0, decimal0, decimal1);
  uint256 price1 = FullMath.mulDiv(reserve0, PRICE_PRECISION, reserve1);
  return (price1, priceDecimals);
}
```

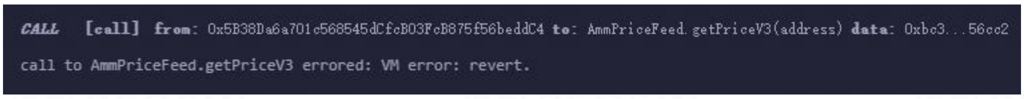
Figure 2 Source code of *getPriceV2* function(Fixed)

[DIPX-2] Integer Overflow

Severity Level	High
Type	General Vulnerability
Lines	AmmPriceFeed.sol#L133

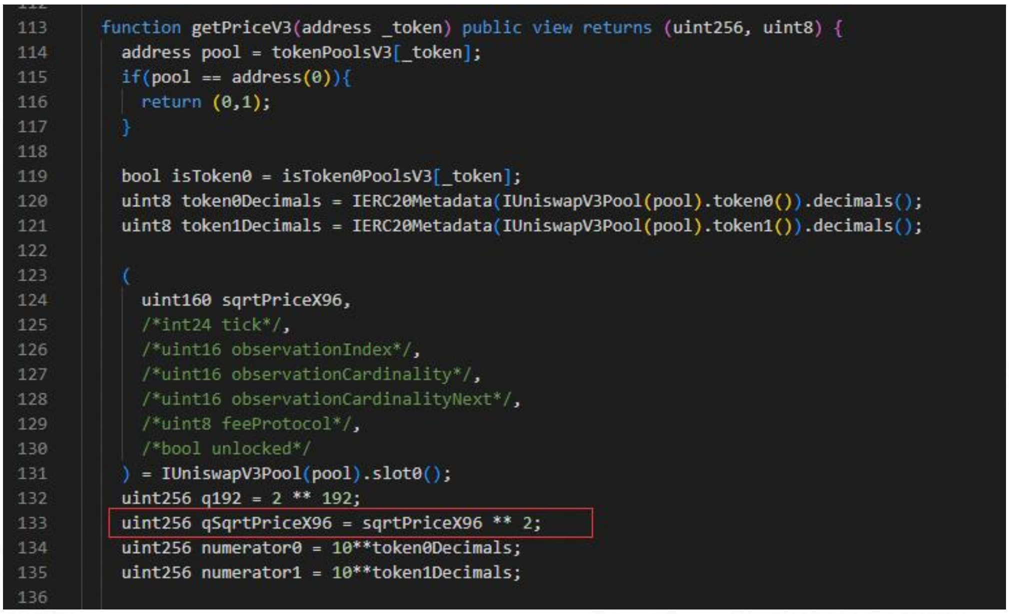
Description

In the AmmPriceFeed.sol contract, the *getPriceV3* function may cause overflow during price calculation, resulting in the entire transaction being reverted.



```
CALL   [call] from: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4 to: AmmPriceFeed.getPriceV3(address) data: 0xbc3...56cc2
call to AmmPriceFeed.getPriceV3 errored: VM error: revert.
```

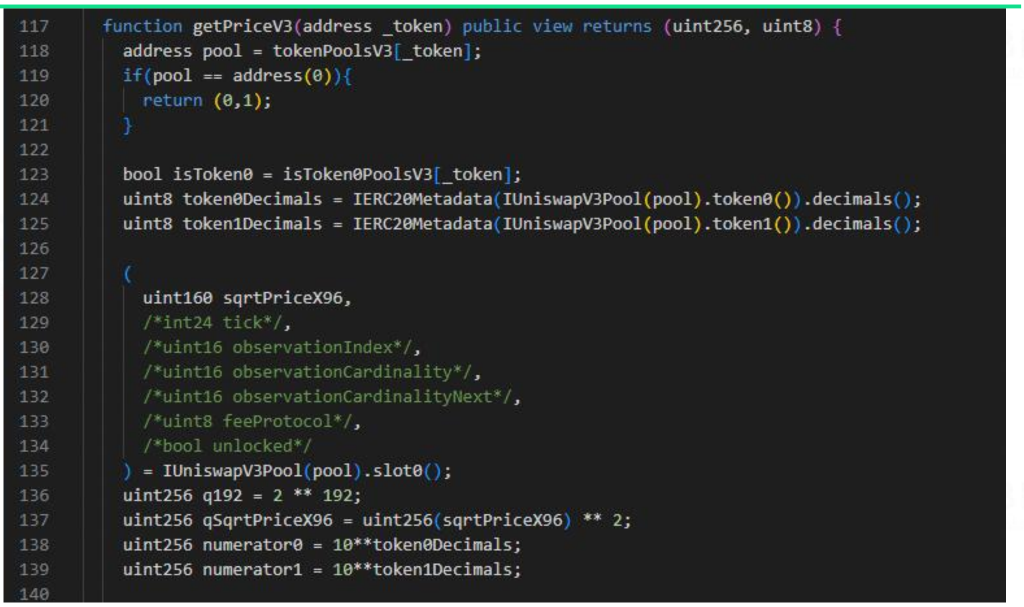
Figure 3 Error information



```
113   function getPriceV3(address _token) public view returns (uint256, uint8) {
114       address pool = tokenPoolsV3[_token];
115       if(pool == address(0)){
116           return (0,1);
117       }
118
119       bool isToken0 = isToken0PoolsV3[_token];
120       uint8 token0Decimals = IERC20Metadata(IUniswapV3Pool(pool).token0()).decimals();
121       uint8 token1Decimals = IERC20Metadata(IUniswapV3Pool(pool).token1()).decimals();
122
123       (
124           uint160 sqrtPriceX96,
125           /*int24 tick*/,
126           /*uint16 observationIndex*/,
127           /*uint16 observationCardinality*/,
128           /*uint16 observationCardinalityNext*/,
129           /*uint8 feeProtocol*/,
130           /*bool unlocked*/
131       ) = IUniswapV3Pool(pool).slot0();
132       uint256 q192 = 2 ** 192;
133       uint256 qSqrtPriceX96 = sqrtPriceX96 ** 2;
134       uint256 numerator0 = 10**token0Decimals;
135       uint256 numerator1 = 10**token1Decimals;
136
```

Figure 4 Source code of *getPriceV3* function(Unfixed)

Recommendations	It is recommended to modify it to uint256.
Status	Fixed.

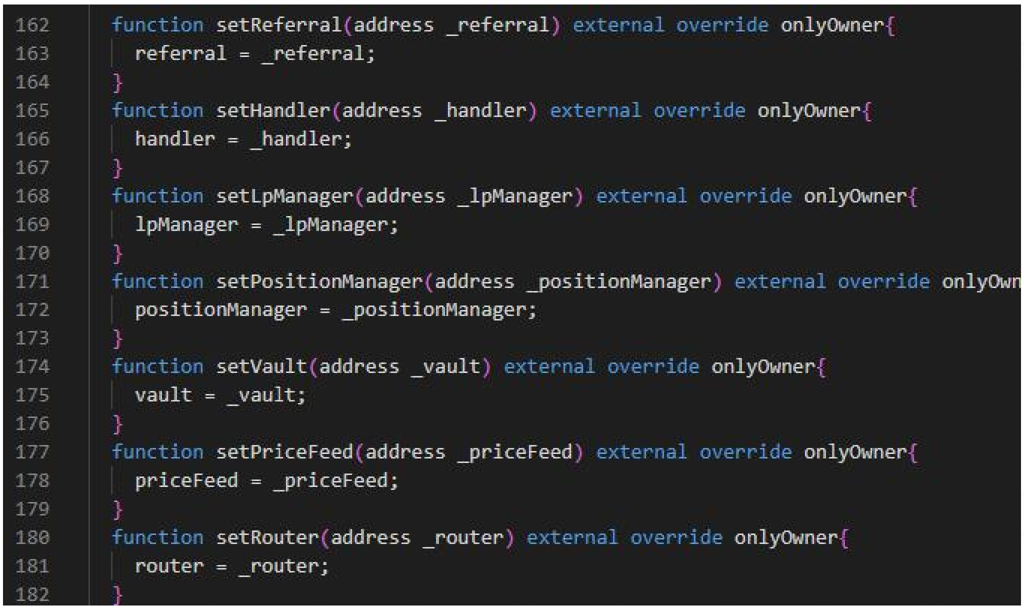


```
function getPriceV3(address _token) public view returns (uint256, uint8) {
  address pool = tokenPoolsV3[_token];
  if(pool == address(0)){
    return (0,1);
  }

  bool isToken0 = isToken0PoolsV3[_token];
  uint8 token0Decimals = IERC20Metadata(IUniswapV3Pool(pool).token0()).decimals();
  uint8 token1Decimals = IERC20Metadata(IUniswapV3Pool(pool).token1()).decimals();

  (
    uint160 sqrtPriceX96,
    /*int24 tick*/,
    /*uint16 observationIndex*/,
    /*uint16 observationCardinality*/,
    /*uint16 observationCardinalityNext*/,
    /*uint8 feeProtocol*/,
    /*bool unlocked*/
  ) = IUniswapV3Pool(pool).slot0();
  uint256 q192 = 2 ** 192;
  uint256 qSqrtPriceX96 = uint256(sqrtPriceX96) ** 2;
  uint256 numerator0 = 10**token0Decimals;
  uint256 numerator1 = 10**token1Decimals;

```

Figure 5 Source code of *getPriceV3* function(Fixed)

[DIPX-3] Excessive permissions

Severity Level	Medium
Type	Business Security
Lines	None
Description	The owner can modify multiple external contract addresses and oracle addresses. Additionally, the Minter of the Vault contract can transfer funds arbitrarily. If the private key is lost, it may pose a threat to the security of users' funds.



```
function setReferral(address _referral) external override onlyOwner{
  referral = _referral;
}
function setHandler(address _handler) external override onlyOwner{
  handler = _handler;
}
function setLpManager(address _lpManager) external override onlyOwner{
  lpManager = _lpManager;
}
function setPositionManager(address _positionManager) external override onlyOwn
  positionManager = _positionManager;
}
function setVault(address _vault) external override onlyOwner{
  vault = _vault;
}
function setPriceFeed(address _priceFeed) external override onlyOwner{
  priceFeed = _priceFeed;
}
function setRouter(address _router) external override onlyOwner{
  router = _router;
}
```

Figure 6 Source code of related function

Recommendations	It is recommended to manage the owner account using a multi-signature contract.
Status	Acknowledged. The project party will use multi-signature and time lock.

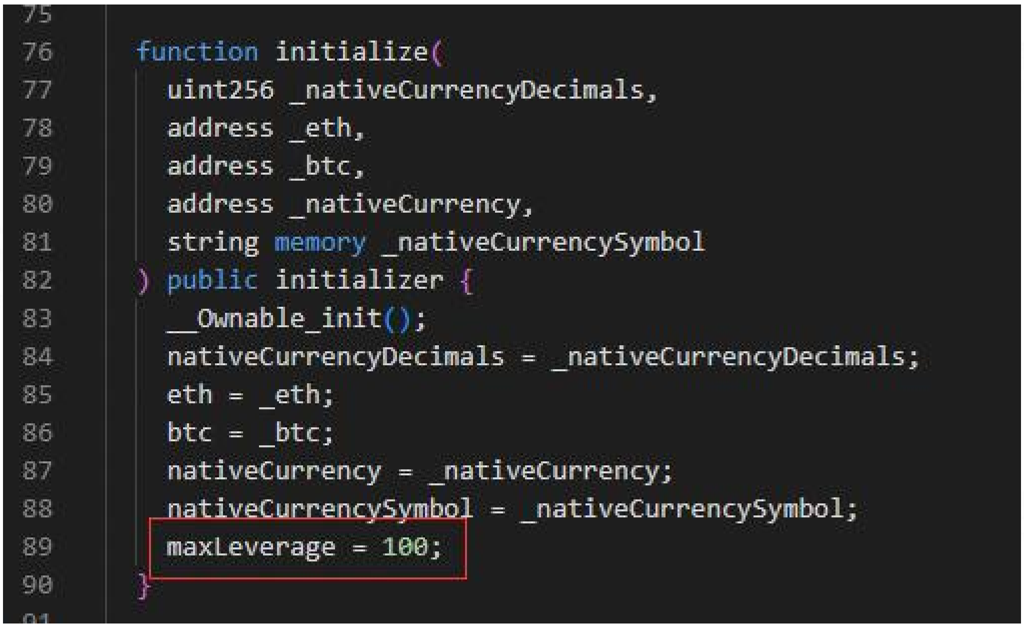
[DIPX-4] Default leverage too high

Severity Level Medium

Type Business Security

Lines DipxStorage.sol#L89

Description The maximum leverage for small cryptocurrencies is too high, which may result in excessive volatility and difficulty in timely liquidation. Attackers may also exploit high-leverage arbitrage opportunities by manipulating the oracle.



```
75
76      function initialize(
77        uint256 _nativeCurrencyDecimals,
78        address _eth,
79        address _btc,
80        address _nativeCurrency,
81        string memory _nativeCurrencySymbol
82      ) public initializer {
83        __Ownable_init();
84        nativeCurrencyDecimals = _nativeCurrencyDecimals;
85        eth = _eth;
86        btc = _btc;
87        nativeCurrency = _nativeCurrency;
88        nativeCurrencySymbol = _nativeCurrencySymbol;
89        maxLeverage = 100;
90      }
91
```

Figure 7 Source code of *initialize* function

Recommendations It is recommended to calculate the maximum leverage separately for small and large cryptocurrencies.

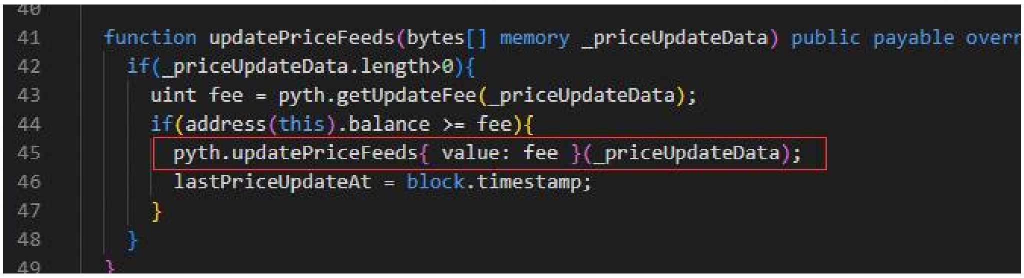
Status Acknowledged. The project party will use BTC/ETH as a price indicator, which will not cause too much fluctuation. Changes in this regard will be considered in the future.

[DIPX-5] Too long array may cause gas exhaustion

Severity Level	Medium
Type	Business Security
Lines	PythPriceFeed.sol#L45

Description

In the PythPriceFeed.sol contract, the *updatePriceFeeds* function needs to pay fees to Pyth, and the amount of the fees is determined by the length of the _priceUpdateData array. If an attacker uses the *updatePriceFeeds* function and maliciously passes in a longer _priceUpdateData, it will cause the contract's balance to be depleted and subsequent *updatePriceFeeds* calls will be invalid.



```
40
41    function updatePriceFeeds(bytes[] memory _priceUpdateData) public payable overr
42      if(_priceUpdateData.length>0){
43        uint fee = pyth.getUpdateFee(_priceUpdateData);
44        if(address(this).balance >= fee){
45          pyth.updatePriceFeeds{ value: fee }(_priceUpdateData);
46          lastPriceUpdateAt = block.timestamp;
47        }
48      }
49    }
```

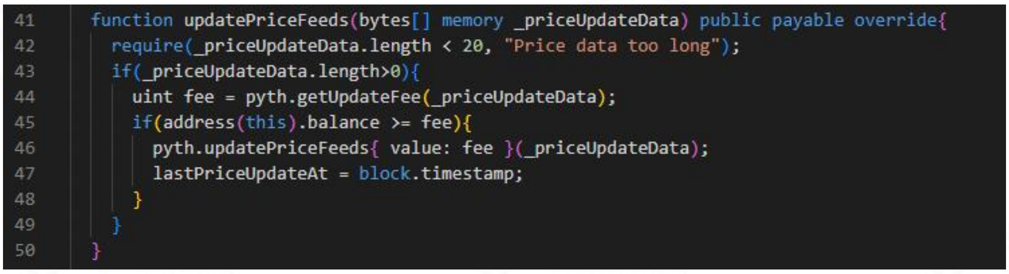
Figure 8 Source code of *updatePriceFeeds* function(Unfixed)

Recommendations

It is recommended to limit the length of _priceUpdateData and modify the visibility of the *updatePriceFeeds* function.

Status

Fixed.



```
41  function updatePriceFeeds(bytes[] memory _priceUpdateData) public payable override{
42    require(_priceUpdateData.length < 20, "Price data too long");
43    if(_priceUpdateData.length>0){
44      uint fee = pyth.getUpdateFee(_priceUpdateData);
45      if(address(this).balance >= fee){
46        pyth.updatePriceFeeds{ value: fee }(_priceUpdateData);
47        lastPriceUpdateAt = block.timestamp;
48      }
49    }
50  }
```

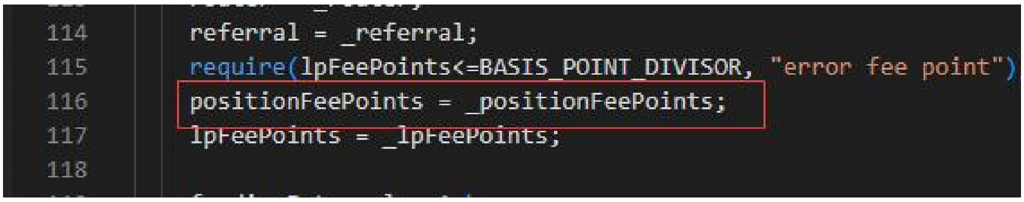
Figure 9 Source code of *updatePriceFeeds* function(Fixed)

[DIPX-6] Unlimited fee

Severity Level	Low
Type	Business Security
Lines	DipxStorage.sol#L115-116

Description

In the DipxStorage contract, the positionFeePoints and accountPositionFeePoints should also be limited to a maximum value. If the range of fees is not limited, users may be charged high fees.



```
114        referral = _referral;
115        require(lpFeePoints<=BASIS_POINT_DIVISOR, "error fee point")
116        positionFeePoints = _positionFeePoints;
117        lpFeePoints = _lpFeePoints;
118
```

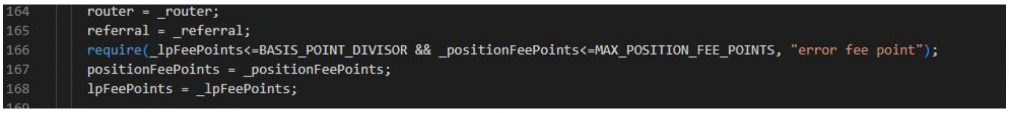
Figure 10 Source code of *initialize* function(Unfixed)

Recommendations

It is recommended to set the maximum limit.

Status

Fixed.



```
        router = _router;
        referral = _referral;
        require(_lpFeePoints<=BASIS_POINT_DIVISOR && _positionFeePoints<=MAX_POSITION_FEE_POINTS, "error fee point");
        positionFeePoints = _positionFeePoints;
        lpFeePoints = _lpFeePoints;
```

Figure 11 Source code of *initialize* function(Fixed)

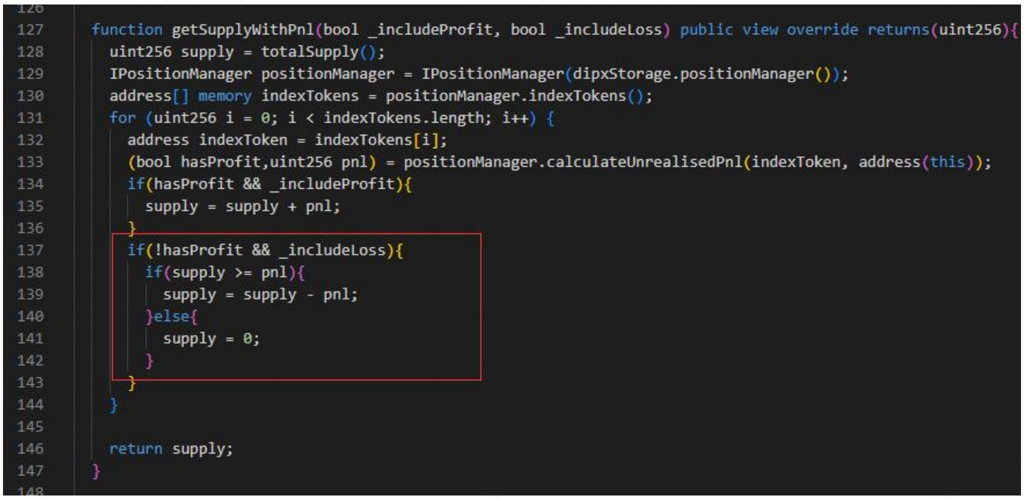
[DIPX-7] Unreasonable calculation in Pnlsupply

Severity Level Low

Type Business Security

Lines MixedLP.sol and SingleLP.sol#L137-143

Description In the MixedLP and SingleLP contracts, the *getSupplyWithPnl* function does not continue to deduct supply if the supply has been reduced to 0 in the previous rounds while iterating through the index tokens. However, if the later index tokens have profits, the total supply deducted will be less than expected.

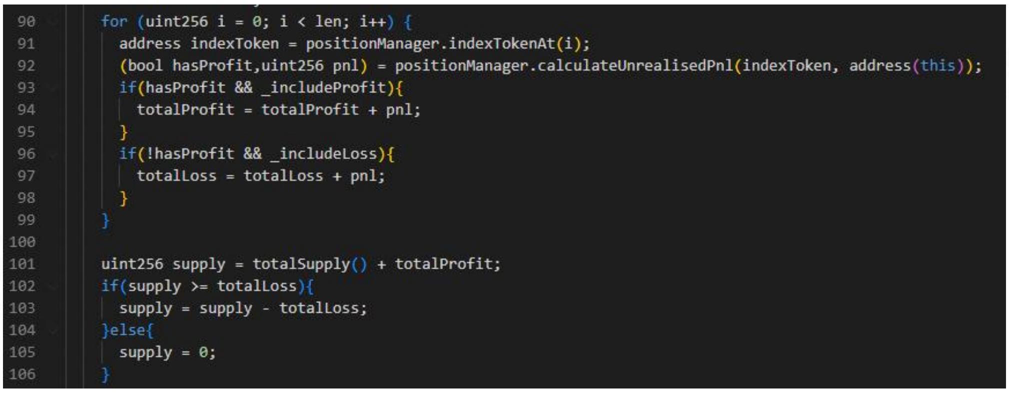


```
127  function getSupplyWithPnl(bool _includeProfit, bool _includeLoss) public view override returns(uint256){
128    uint256 supply = totalSupply();
129    IPositionManager positionManager = IPositionManager(dipxStorage.positionManager());
130    address[] memory indexTokens = positionManager.indexTokens();
131    for (uint256 i = 0; i < indexTokens.length; i++) {
132      address indexToken = indexTokens[i];
133      (bool hasProfit,uint256 pnl) = positionManager.calculateUnrealisedPnl(indexToken, address(this));
134      if(hasProfit && _includeProfit){
135        supply = supply + pnl;
136      }
137      if(!hasProfit && _includeLoss){
138        if(supply >= pnl){
139          supply = supply - pnl;
140        }else{
141          supply = 0;
142        }
143      }
144    }
145
146    return supply;
147  }
```

Figure 12 Source code of *getSupplyWithPnl* function(Unfixed)

Recommendations It is recommended to separate the profits and losses calculations, and subtract them from the final result after calculating them separately from the supply.

Status Fixed.



```
for (uint256 i = 0; i < len; i++) {
  address indexToken = positionManager.indexTokenAt(i);
  (bool hasProfit,uint256 pnl) = positionManager.calculateUnrealisedPnl(indexToken, address(this));
  if(hasProfit && _includeProfit){
    totalProfit = totalProfit + pnl;
  }
  if(!hasProfit && _includeLoss){
    totalLoss = totalLoss + pnl;
  }
}

uint256 supply = totalSupply() + totalProfit;
if(supply >= totalLoss){
  supply = supply - totalLoss;
}else{
  supply = 0;
}
```

Figure 13 Source code of *getSupplyWithPnl* function(Fixed)

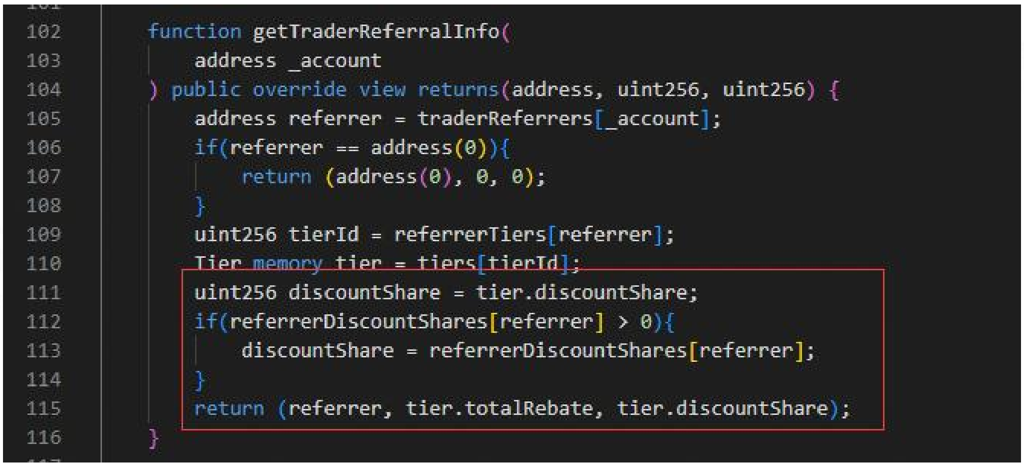
[DIPX-8] Return parameter error

Severity Level Low

Type Business Security

Lines Referral.sol#L110-115

Description In the Referral contract, the *getTraderReferralInfo* function still returns tier.discountShare instead of discountShare when retrieving the discountShare, will lead to return parameter error.

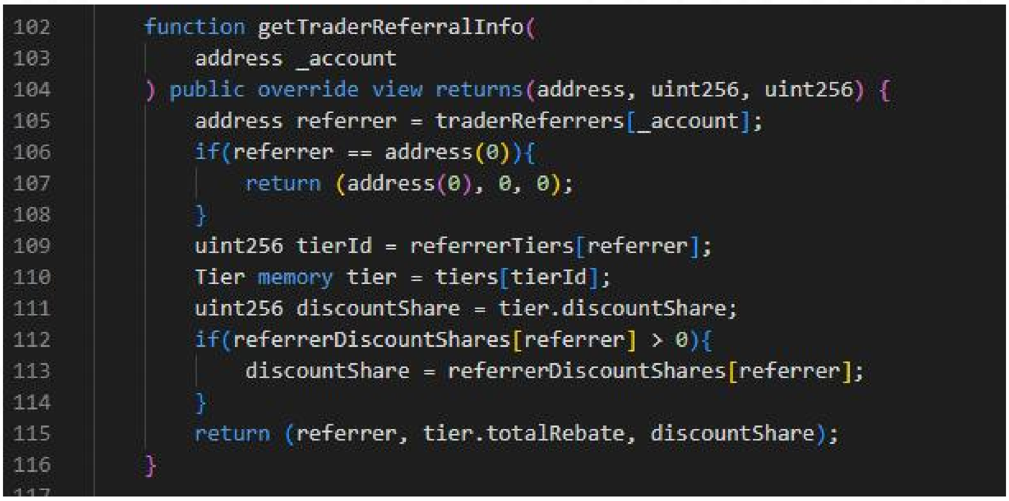


```
102     function getTraderReferralInfo(
103         address _account
104     ) public override view returns(address, uint256, uint256) {
105         address referrer = traderReferrers[_account];
106         if(referrer == address(0)){
107             return (address(0), 0, 0);
108         }
109         uint256 tierId = referrerTiers[referrer];
110         Tier memory tier = tiers[tierId];
111         uint256 discountShare = tier.discountShare;
112         if(referrerDiscountShares[referrer] > 0){
113             discountShare = referrerDiscountShares[referrer];
114         }
115         return (referrer, tier.totalRebate, tier.discountShare);
116     }
```

Figure 14 Source code of *getTraderReferralInfo* function(Unfixed)

Recommendations It is recommended to return discountShare.

Status Fixed.



```
102     function getTraderReferralInfo(
103         address _account
104     ) public override view returns(address, uint256, uint256) {
105         address referrer = traderReferrers[_account];
106         if(referrer == address(0)){
107             return (address(0), 0, 0);
108         }
109         uint256 tierId = referrerTiers[referrer];
110         Tier memory tier = tiers[tierId];
111         uint256 discountShare = tier.discountShare;
112         if(referrerDiscountShares[referrer] > 0){
113             discountShare = referrerDiscountShares[referrer];
114         }
115         return (referrer, tier.totalRebate, discountShare);
116     }
```

Figure 15 Source code of *getTraderReferralInfo* function(Fixed)

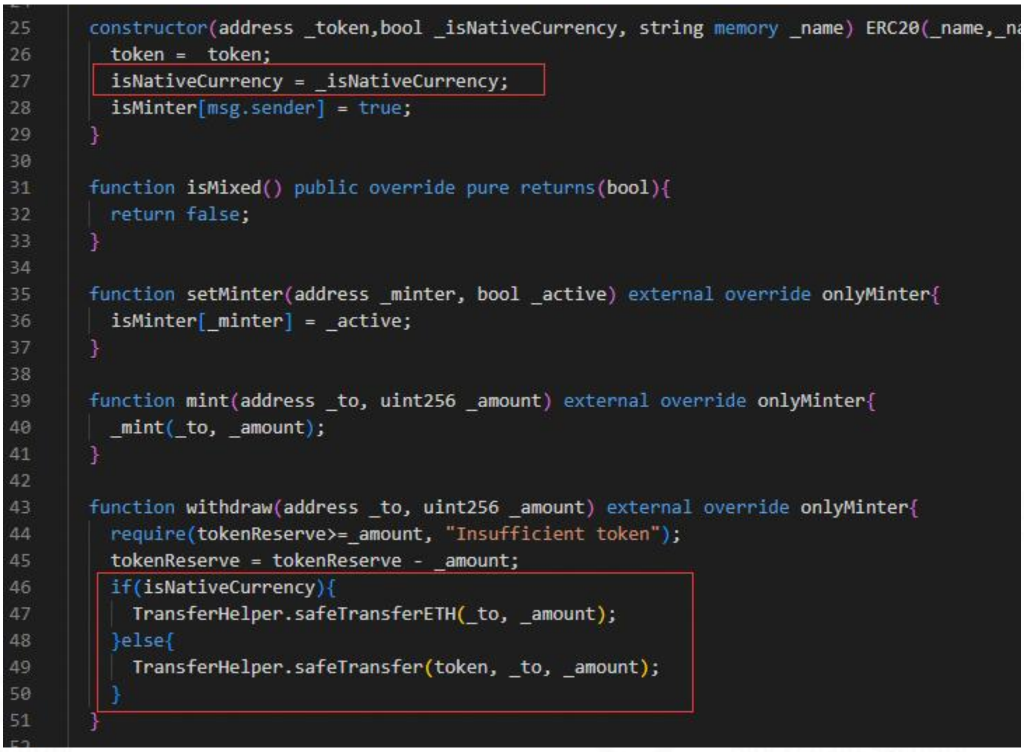
[DIPX-9] Platform currency locked

Severity Level Low

Type Business Security

Lines SingleLP.sol#L25-50

Description In the SingleLP contract, isNativeCurrency can only be initialized in the constructor. If isNativeCurrency is false and the contract receives platform tokens, the platform tokens will be locked in the contract.

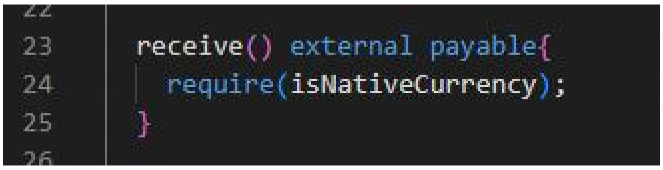


```
25    constructor(address _token,bool _isNativeCurrency, string memory _name) ERC20(_name,_na
26      token =  token;
27      isNativeCurrency = _isNativeCurrency;
28      isMinter[msg.sender] = true;
29    }
30
31    function isMixed() public override pure returns(bool){
32      return false;
33    }
34
35    function setMinter(address _minter, bool _active) external override onlyMinter{
36      isMinter[_minter] = _active;
37    }
38
39    function mint(address _to, uint256 _amount) external override onlyMinter{
40      _mint(_to, _amount);
41    }
42
43    function withdraw(address _to, uint256 _amount) external override onlyMinter{
44      require(tokenReserve>=_amount, "Insufficient token");
45      tokenReserve = tokenReserve - _amount;
46      if(isNativeCurrency){
47        TransferHelper.safeTransferETH(_to, _amount);
48      }else{
49        TransferHelper.safeTransfer(token, _to, _amount);
50      }
51    }
```

Figure 16 Source code of *withdraw* function(Unfixed)

Recommendations It is recommended to handle the case where isNativeCurrency is false in the *receive* function.

Status Fixed.



```
23    receive() external payable{
24      require(isNativeCurrency);
25    }
```

Figure 17 Source code of *receive* function(Fixed)

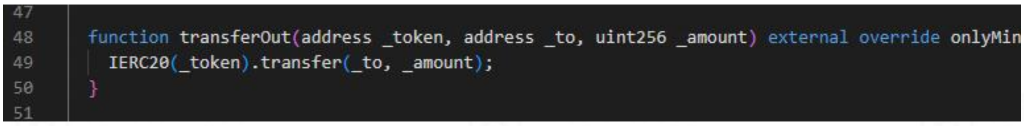
[DIPX-10] Insecure transfer function

Severity Level Low

Type Coding Conventions

Lines Vault.sol#L48-50

Description In the Vault contract, the *transferOut* function uses an unsafe *transfer* function, which may cause problems such as false deposits.

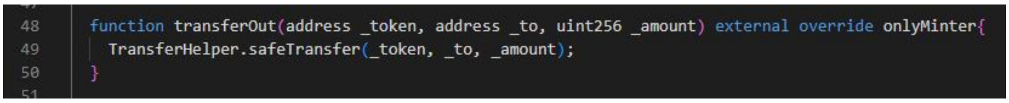


```
47
48    function transferOut(address _token, address _to, uint256 _amount) external override onlyMin
49      IERC20(_token).transfer(_to, _amount);
50    }
51
```

Figure 18 Source code of *transferOut* function(Unfixed)

Recommendations It is recommended to use *safeTransfer* or other similar functions to validate the transfer effectively.

Status Fixed.



```
48    function transferOut(address _token, address _to, uint256 _amount) external override onlyMinter{
49      TransferHelper.safeTransfer(_token, _to, _amount);
50    }
```

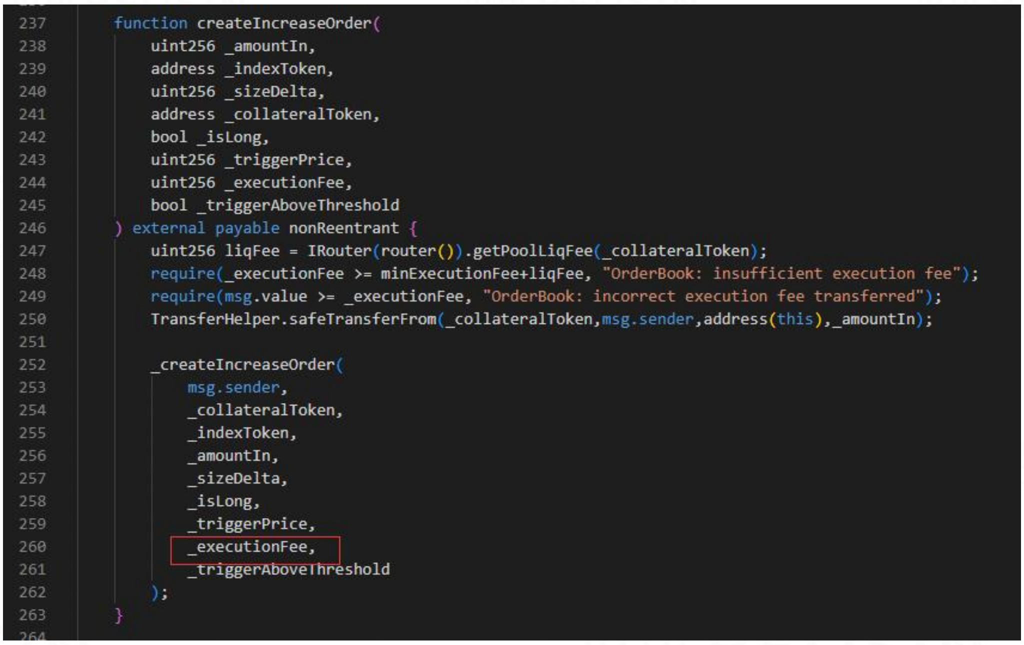
Figure 19 Source code of *transferOut* function(Fixed)

[DIPX-11] The fee parameter is incorrect

Severity Level	Low
Type	Business Security
Lines	OrderBook.sol#L260

Description

In the OrderBook contract, the *_createIncreaseOrder* should use msg.value instead of _executionFee. This is because when msg.value is greater than _executionFee, the cancellation order returns _executionFee instead of msg.value, resulting in less received funds.



```
237     function createIncreaseOrder(
238         uint256 _amountIn,
239         address _indexToken,
240         uint256 _sizeDelta,
241         address _collateralToken,
242         bool _isLong,
243         uint256 _triggerPrice,
244         uint256 _executionFee,
245         bool _triggerAboveThreshold
246     ) external payable nonReentrant {
247         uint256 liqFee = IRouter(router()).getPoolLiqFee(_collateralToken);
248         require(_executionFee >= minExecutionFee+liqFee, "OrderBook: insufficient execution fee");
249         require(msg.value >= _executionFee, "OrderBook: incorrect execution fee transferred");
250         TransferHelper.safeTransferFrom(_collateralToken,msg.sender,address(this),_amountIn);
251
252         _createIncreaseOrder(
253             msg.sender,
254             _collateralToken,
255             _indexToken,
256             _amountIn,
257             _sizeDelta,
258             _isLong,
259             _triggerPrice,
260             _executionFee,
261             _triggerAboveThreshold
262         );
263     }
```

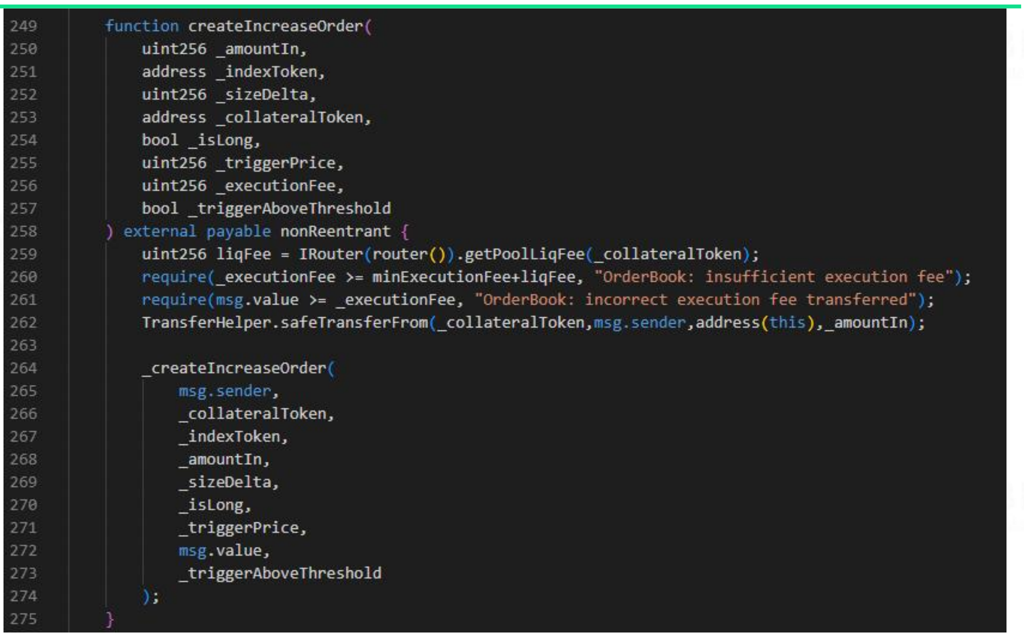
Figure 20 Source code of *createIncreaseOrder* function(Unfixed)

Recommendations

It is recommended to modify _executionFee to msg.value.

Status

Fixed.



```
    function createIncreaseOrder(
        uint256 _amountIn,
        address _indexToken,
        uint256 _sizeDelta,
        address _collateralToken,
        bool _isLong,
        uint256 _triggerPrice,
        uint256 _executionFee,
        bool _triggerAboveThreshold
    ) external payable nonReentrant {
        uint256 liqFee = IRouter(router()).getPoolLiqFee(_collateralToken);
        require(_executionFee >= minExecutionFee+liqFee, "OrderBook: insufficient execution fee");
        require(msg.value >= _executionFee, "OrderBook: incorrect execution fee transferred");
        TransferHelper.safeTransferFrom(_collateralToken,msg.sender,address(this),_amountIn);

        _createIncreaseOrder(
            msg.sender,
            _collateralToken,
            _indexToken,
            _amountIn,
            _sizeDelta,
            _isLong,
            _triggerPrice,
            msg.value,
            _triggerAboveThreshold
        );
    }
```

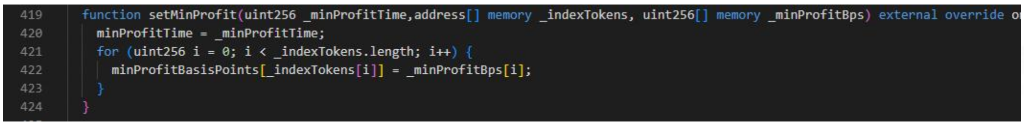
Figure 21 Source code of *createIncreaseOrder* function(Fixed)

[DIPX-12] Array lengths are misaligned in multiple places

Severity Level	Low
Type	Business Security
Lines	DipxStorage.sol#L419-424 OrderBook.sol#L352-362

Description

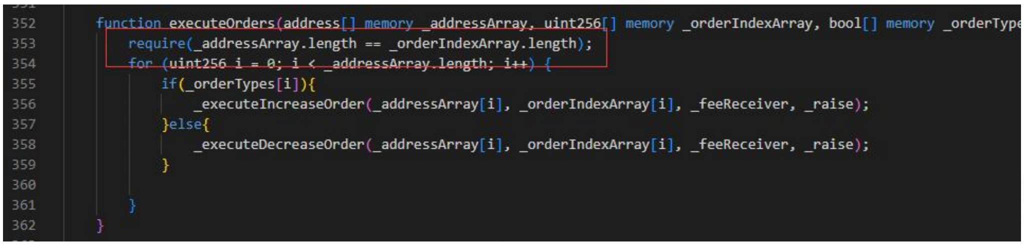
A. In the DipxStorage contract, array alignment verification is not performed for _indexTokens and _minProfitBps.



```
function setMinProfit(uint256 _minProfitTime,address[] memory _indexTokens, uint256[] memory _minProfitBps) external override o
  minProfitTime = _minProfitTime;
  for (uint256 i = 0; i < _indexTokens.length; i++) {
    minProfitBasisPoints[_indexTokens[i]] = _minProfitBps[i];
  }
}
```

Figure 22 Source code of *setMinProfit* function(Unfixed)

B. In the OrderBook contract, alignment verification is performed only for _addressArray and _orderIndexArray, but not for _orderTypes.



```
    function executeOrders(address[] memory _addressArray, uint256[] memory _orderIndexArray, bool[] memory _orderType
        require(_addressArray.length == _orderIndexArray.length);
        for (uint256 i = 0; i < _addressArray.length; i++) {
            if(_orderTypes[i]){
                _executeIncreaseOrder(_addressArray[i], _orderIndexArray[i], _feeReceiver, _raise);
            }else{
                _executeDecreaseOrder(_addressArray[i], _orderIndexArray[i], _feeReceiver, _raise);
            }

        }
    }
```

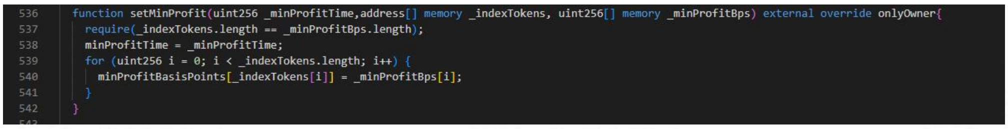
Figure 23 Source code of *executeOrders* function(Unfixed)

Recommendations

It is recommended to add alignment verification.

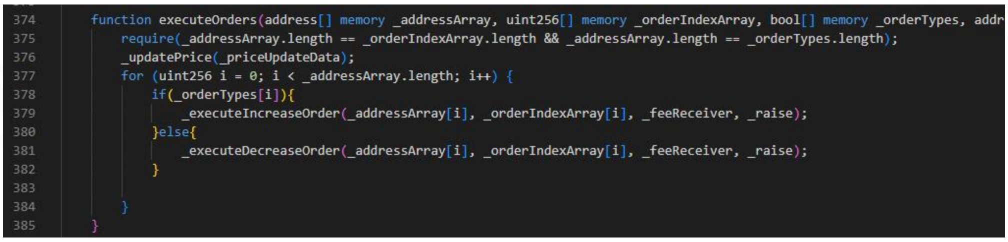
Status

Fixed.



```
function setMinProfit(uint256 _minProfitTime,address[] memory _indexTokens, uint256[] memory _minProfitBps) external override onlyOwner{
  require(_indexTokens.length == _minProfitBps.length);
  minProfitTime = _minProfitTime;
  for (uint256 i = 0; i < _indexTokens.length; i++) {
    minProfitBasisPoints[_indexTokens[i]] = _minProfitBps[i];
  }
}
```

Figure 24 Source code of *setMinProfit* function(Fixed)



```
    function executeOrders(address[] memory _addressArray, uint256[] memory _orderIndexArray, bool[] memory _orderTypes, addr
        require(_addressArray.length == _orderIndexArray.length && _addressArray.length == _orderTypes.length);
        _updatePrice(_priceUpdateData);
        for (uint256 i = 0; i < _addressArray.length; i++) {
            if(_orderTypes[i]){
                _executeIncreaseOrder(_addressArray[i], _orderIndexArray[i], _feeReceiver, _raise);
            }else{
                _executeDecreaseOrder(_addressArray[i], _orderIndexArray[i], _feeReceiver, _raise);
            }

        }
    }
```

Figure 25 Source code of *executeOrders* function(Fixed)

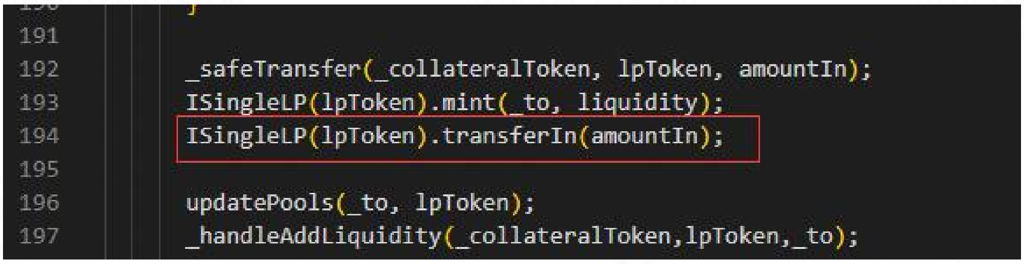
[DIPX-13] Inconsistent Deflationary Token Reserve

Severity Level Low

Type Business Security

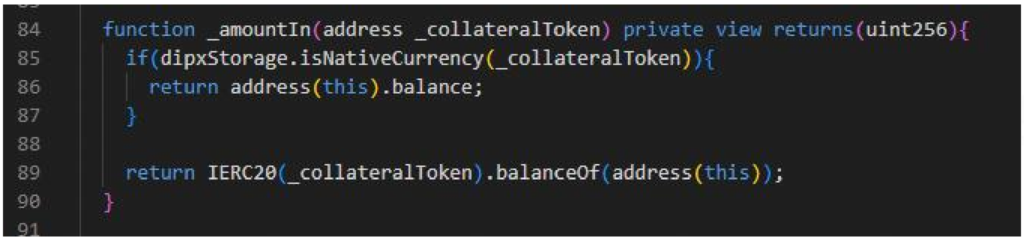
Lines Lpmanager.sol#L194,84-90

Description If the collateral pool allows the use of deflationary tokens, using the *_addLiquidity* function in the Lpmanager contract to add liquidity by directly using the amountIn as a parameter for *transferIn* may result in inconsistent tokenReserves with the actual reserves in the pool.



```
191
192         _safeTransfer(_collateralToken, lpToken, amountIn);
193         ISingleLP(lpToken).mint(_to, liquidity);
194         ISingleLP(lpToken).transferIn(amountIn);
195
196         updatePools(_to, lpToken);
197         _handleAddLiquidity(_collateralToken,lpToken,_to);
```

Figure 26 Source code of *_addLiquidity* function(Unfixed)

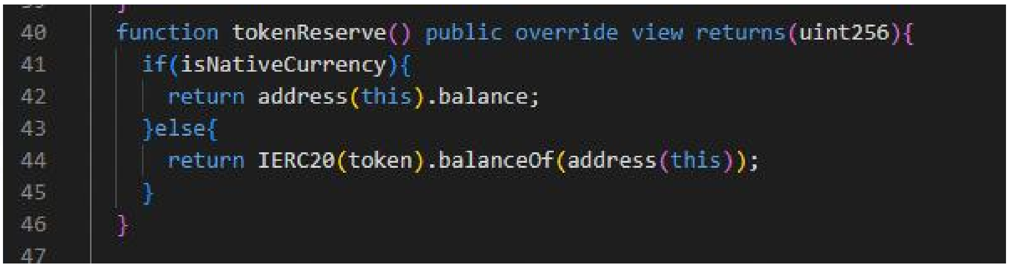


```
84    function _amountIn(address _collateralToken) private view returns(uint256){
85      if(dipxStorage.isNativeCurrency(_collateralToken)){
86        return address(this).balance;
87      }
88
89      return IERC20(_collateralToken).balanceOf(address(this));
90    }
```

Figure 27 Source code of *_amountIn* function(Unfixed)

Recommendations It is recommended to add a liquidity interface for deflationary tokens or to prohibit deflationary tokens from participating in the project.

Status Fixed. The Single pool supports deflationary tokens, and the Mixed pool does not support deflationary tokens.



```
function tokenReserve() public override view returns(uint256){
  if(isNativeCurrency){
    return address(this).balance;
  }else{
    return IERC20(token).balanceOf(address(this));
  }
}
```

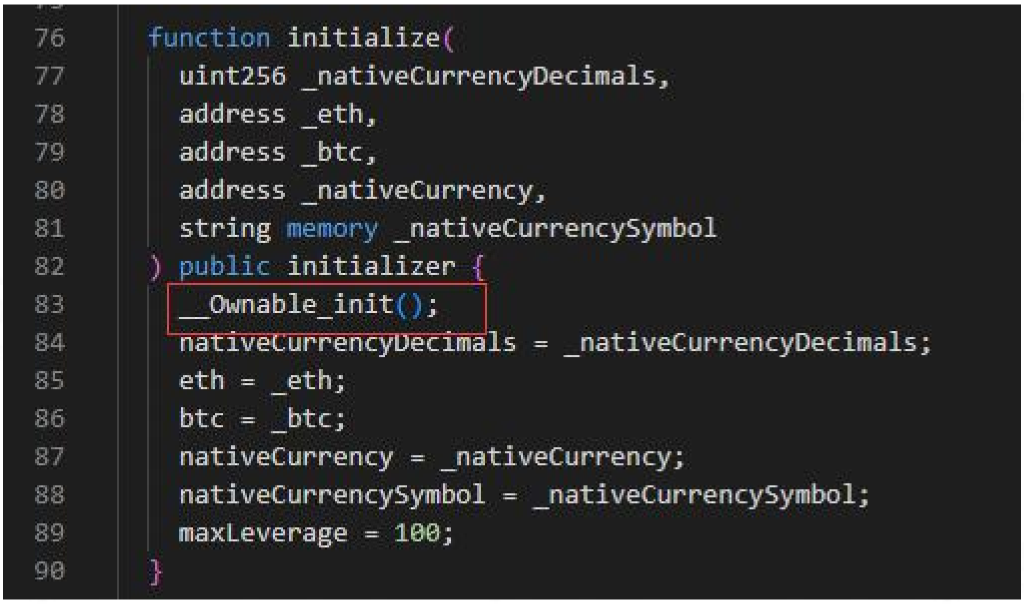
Figure 28 Source code of *tokenReserve* function in SingleLP(Fixed)

[DIPX-14] Front-running risk

Severity Level	Info
Type	Business Security
Lines	DipxStorage.sol#L83

Description

Multiple permission initializations are in the initialize function, which needs to be careful of MEV front-running.



```
76  function initialize(
77    uint256 _nativeCurrencyDecimals,
78    address _eth,
79    address _btc,
80    address _nativeCurrency,
81    string memory _nativeCurrencySymbol
82  ) public initializer {
83    __Ownable_init();
84    nativeCurrencyDecimals = _nativeCurrencyDecimals;
85    eth = _eth;
86    btc = _btc;
87    nativeCurrency = _nativeCurrency;
88    nativeCurrencySymbol = _nativeCurrencySymbol;
89    maxLeverage = 100;
90  }
```

Figure 29 Source code of *initialize* function

Recommendations

It is recommended to limit the initialization account or put the permission initialization into the constructor.

Status

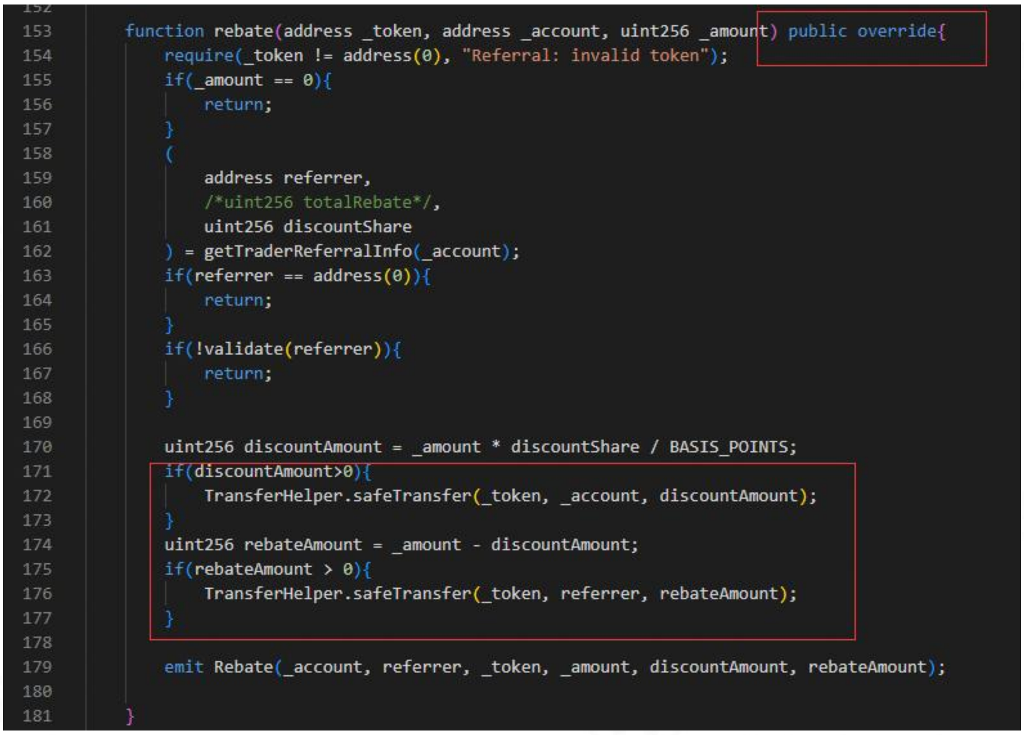
Acknowledged. The project party will initialize when deploying the contract.

[DIPX-15] Referral contract token transfer risk

Severity Level	Info
Type	Business Security
Lines	Referral.sol#L171-177

Description

In the Referral contract, the *rebate* function can be called by anyone. An attacker with the corresponding ReferralInfo can freely call the function and transfer the contract's tokens to any account and referrer address.



```
153     function rebate(address _token, address _account, uint256 _amount) public override{
154         require(_token != address(0), "Referral: invalid token");
155         if(_amount == 0){
156             return;
157         }
158         (
159             address referrer,
160             /*uint256 totalRebate*/,
161             uint256 discountShare
162         ) = getTraderReferralInfo(_account);
163         if(referrer == address(0)){
164             return;
165         }
166         if(!validate(referrer)){
167             return;
168         }
169
170         uint256 discountAmount = _amount * discountShare / BASIS_POINTS;
171         if(discountAmount>0){
172             TransferHelper.safeTransfer(_token, _account, discountAmount);
173         }
174         uint256 rebateAmount = _amount - discountAmount;
175         if(rebateAmount > 0){
176             TransferHelper.safeTransfer(_token, referrer, rebateAmount);
177         }
178
179         emit Rebate(_account, referrer, _token, _amount, discountAmount, rebateAmount);
180
181     }
```

Figure 30 Source code of *rebate* function

Recommendations	It is recommended to add function call restrictions.
Status	Acknowledged.

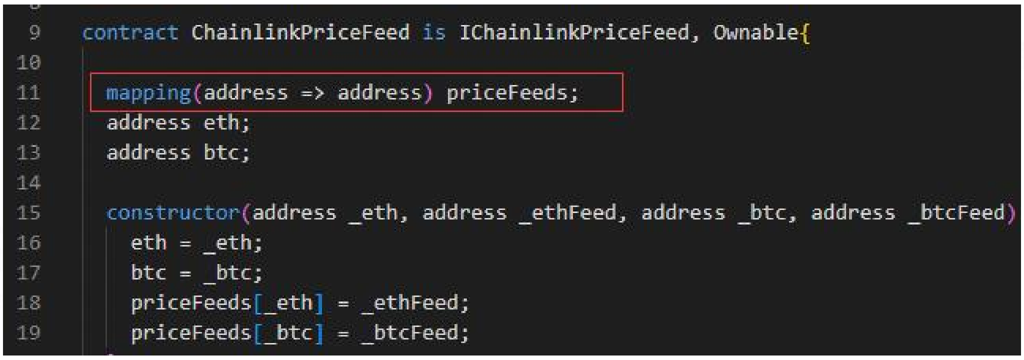
[DIPX-16] Visibility exception

Severity Level Info

Type Coding Conventions

Lines ChainlinkPriceFeed.sol#L11

Description In the ChainlinkPriceFeed contract, the visibility of the priceFeeds is private by default, which means users cannot query the oracle address corresponding to the token.

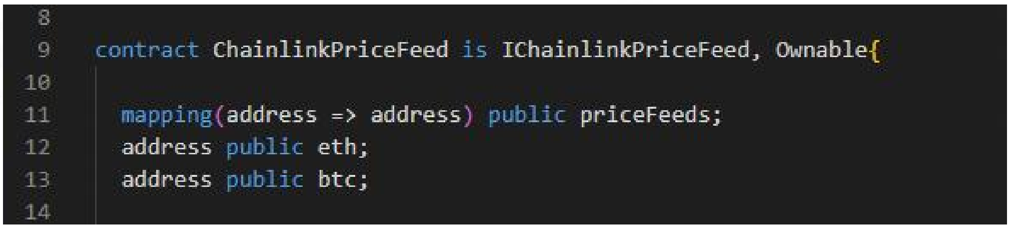


```
 9  contract ChainlinkPriceFeed is IChainlinkPriceFeed, Ownable{
10
11    mapping(address => address) priceFeeds;
12    address eth;
13    address btc;
14
15    constructor(address _eth, address _ethFeed, address _btc, address _btcFeed)
16      eth = _eth;
17      btc = _btc;
18      priceFeeds[_eth] = _ethFeed;
19      priceFeeds[_btc] = _btcFeed;
```

Figure 31 Source code of related code(Unfixed)

Recommendations It is recommended to modify the visibility of the priceFeeds or add a *getPriceFeeds* function for querying.

Status Fixed.



```
 8
 9  contract ChainlinkPriceFeed is IChainlinkPriceFeed, Ownable{
10
11    mapping(address => address) public priceFeeds;
12    address public eth;
13    address public btc;
14
```

Figure 32 Source code of related code(Fixed)

3 Appendix

3.1 Vulnerability Assessment Metrics and Status in Smart Contracts

3.1.1 Metrics

In order to objectively assess the severity level of vulnerabilities in blockchain systems, this report provides detailed assessment metrics for security vulnerabilities in smart contracts with reference to CVSS 3.1 (Common Vulnerability Scoring System Ver 3.1).

According to the severity level of vulnerability, the vulnerabilities are classified into four levels: "critical", "high", "medium" and "low". It mainly relies on the degree of impact and likelihood of exploitation of the vulnerability, supplemented by other comprehensive factors to determine of the severity level.

Impact Likelihood	Severe	High	Medium	Low
Probable	Critical	High	Medium	Low
Possible	High	High	Medium	Low
Unlikely	Medium	Medium	Low	Info
Rare	Low	Low	Info	Info

3.1.2 Degree of impact

- **Severe**

Severe impact generally refers to the vulnerability can have a serious impact on the confidentiality, integrity, availability of smart contracts or their economic model, which can cause substantial economic losses to the contract business system, large-scale data disruption, loss of authority management, failure of key functions, loss of credibility, or indirectly affect the operation of other smart contracts associated with it and cause substantial losses, as well as other severe and mostly irreversible harm.

- **High**

High impact generally refers to the vulnerability can have a relatively serious impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a greater economic loss, local functional unavailability, loss of credibility and other impact to the contract business system.

- **Medium**

Medium impact generally refers to the vulnerability can have a relatively minor impact on the confidentiality, integrity, availability of the smart contract or its economic model, which can cause a small amount of economic loss to the contract business system, individual business unavailability and other impact.

- **Low**

Low impact generally refers to the vulnerability can have a minor impact on the smart contract, which can pose certain security threat to the contract business system and needs to be improved.

3.1.4 Likelihood of Exploitation

- **Probable**

Probable likelihood generally means that the cost required to exploit the vulnerability is low, with no special exploitation threshold, and the vulnerability can be triggered consistently.

- **Possible**

Possible likelihood generally means that exploiting such vulnerability requires a certain cost, or there are certain conditions for exploitation, and the vulnerability is not easily and consistently triggered.

- **Unlikely**

Unlikely likelihood generally means that the vulnerability requires a high cost, or the exploitation conditions are very demanding and the vulnerability is highly difficult to trigger.

- **Rare**

Rare likelihood generally means that the vulnerability requires an extremely high cost or the conditions for exploitation are extremely difficult to achieve.

3.1.5 Fix Results Status

Status	Description
Fixed	The project party fully fixes a vulnerability.
Partially Fixed	The project party did not fully fix the issue, but only mitigated the issue.
Acknowledged	The project party confirms and chooses to ignore the issue.

3.2 Audit Categories

No.	Categories	Subitems
1	Coding Conventions	Compiler Version Security
		Deprecated Items
		Redundant Code
		require/assert Usage
		Gas Consumption
2	General Vulnerability	Integer Overflow/Underflow
		Reentrancy
		Pseudo-random Number Generator (PRNG)
		Transaction-Ordering Dependence
		DoS (Denial of Service)
		Function Call Permissions
		call/delegatecall Security
		Returned Value Security
		tx.origin Usage
		Replay Attack
		Overriding Variables
		Third-party Protocol Interface Consistency
3	Business Security	Business Logics
		Business Implementations
		Manipulable Token Price
		Centralized Asset Control
		Asset Tradability
		Arbitrage Attack

Beosin classified the security issues of smart contracts into three categories: Coding Conventions, General Vulnerability, Business Security. Their specific definitions are as follows:

- **Coding Conventions**

Audit whether smart contracts follow recommended language security coding practices. For example, smart contracts developed in Solidity language should fix the compiler version and do not use deprecated keywords.

- **General Vulnerability**

General Vulnerability include some common vulnerabilities that may appear in smart contract projects. These vulnerabilities are mainly related to the characteristics of the smart contract itself, such as integer overflow/underflow and denial of service attacks.

- **Business Security**

Business security is mainly related to some issues related to the business realized by each project, and has a relatively strong pertinence. For example, whether the lock-up plan in the code match the white paper, or the flash loan attack caused by the incorrect setting of the price acquisition oracle.

*Note that the project may suffer stake losses due to the integrated third-party protocol. This is not something Beosin can control. Business security requires the participation of the project party. The project party and users need to stay vigilant at all times.

3.3 Disclaimer

The Audit Report issued by Beosin is related to the services agreed in the relevant service agreement. The Project Party or the Served Party (hereinafter referred to as the "Served Party") can only be used within the conditions and scope agreed in the service agreement. Other third parties shall not transmit, disclose, quote, rely on or tamper with the Audit Report issued for any purpose.

The Audit Report issued by Beosin is made solely for the code, and any description, expression or wording contained therein shall not be interpreted as affirmation or confirmation of the project, nor shall any warranty or guarantee be given as to the absolute flawlessness of the code analyzed, the code team, the business model or legal compliance.

The Audit Report issued by Beosin is only based on the code provided by the Served Party and the technology currently available to Beosin. However, due to the technical limitations of any organization, and in the event that the code provided by the Served Party is missing information, tampered with, deleted, hidden or subsequently altered, the audit report may still fail to fully enumerate all the risks.

The Audit Report issued by Beosin in no way provides investment advice on any project, nor should it be utilized as investment suggestions of any type. This report represents an extensive evaluation process designed to help our customers improve code quality while mitigating the high risks in blockchain.

3.4 About Beosin

Beosin is the first institution in the world specializing in the construction of blockchain security ecosystem. The core team members are all professors, postdocs, PhDs, and Internet elites from world-renowned academic institutions. Beosin has more than 20 years of research in formal verification technology, trusted computing, mobile security and kernel security, with overseas experience in studying and collaborating in project research at well-known universities. Through the security audit and defense deployment of more than 2,000 smart contracts, over 50 public blockchains and wallets, and nearly 100 exchanges worldwide, Beosin has accumulated rich experience in security attack and defense of the blockchain field, and has developed several security products specifically for blockchain.

BEOSIN
Blockchain Security

Official Website

https://www.beosin.com

Telegram

https://t.me/+dD8Bnqd133RmNWNl

Twitter

https://twitter.com/Beosin_com

Email

Contact@beosin.com