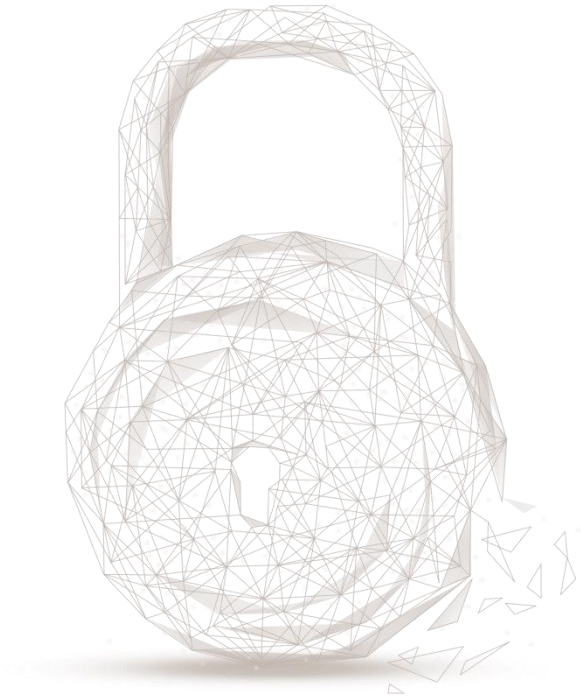
BEOSIN
Blockchain Security

Smart contract security audit report



Audit Number：202103151813

Smart Contract Info：

https://github.com/DODOEX/contractV2/tree/main/contracts/CrowdPooling
https://github.com/DODOEX/contractV2/tree/main/contracts/DODOVendingMachine

Start Commit Hash：

01c544780291a5acc3e2be4980493e63065fb200

Finish Commit Hash：

abe9919a01d17f8acc29fd0a2a95ad3c1bd49264

Start Date：2021.03.10

Completion Date：2021.03.15

Overall Result：Pass

Audit Team: Beosin (Chengdu LianAn) Technology Co. Ltd.

Audit Categories and Results:

No.	Categories	Subitems	Results
1	Coding Conventions	Compiler Version Security	Pass
		Deprecated Items	Pass
		Redundant Code	Pass
		SafeMath Features	Pass
		require/assert Usage	Pass
		Gas Consumption	Pass
		Visibility Specifiers	Pass
		Fallback Usage	Pass
2	General Vulnerability	Integer Overflow/Underflow	Pass
		Reentrancy	Pass
		Pseudo-random Number Generator (PRNG)	Pass
		Transaction-Ordering Dependence	Pass
		DoS (Denial of Service)	Pass

		Access Control of Owner	Pass
		Low-level Function (call/delegatecall) Security	Pass
		Returned Value Security	Pass
		tx.origin Usage	Pass
		Replay Attack	Pass
		Overriding Variables	Pass
3	Business Security	Business Logics	Pass
		Business Implementations	Pass

Disclaimer: This audit is only applied to the type of auditing specified in this report and the scope of given in the results table. Other unknown security vulnerabilities are beyond auditing responsibility. Beosin (Chengdu LianAn) Technology only issues this report based on the attacks or vulnerabilities that already existed or occurred before the issuance of this report. For the emergence of new attacks or vulnerabilities that exist or occur in the future, Beosin (Chengdu LianAn) Technology lacks the capability to judge its possible impact on the security status of smart contracts, thus taking no responsibility for them. The security audit analysis and other contents of this report are based solely on the documents and materials that the contract provider has provided to Beosin (Chengdu LianAn) Technology before the issuance of this report, and the contract provider warrants that there are no missing, tampered, deleted; if the documents and materials provided by the contract provider are missing, tampered, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, or if the documents and materials provided are changed after the issuance of this report, Beosin (Chengdu LianAn) Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin (Chengdu LianAn) Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin (Chengdu LianAn). Due to the technical limitations of any organization, this report conducted by Beosin (Chengdu LianAn) still has the possibility that the entire risk cannot be completely detected. Beosin (Chengdu LianAn) disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin (Chengdu LianAn).

Audit Results Explained:

Beosin (Chengdu LianAn) Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit two folder of project DODODEX, including Coding Standards, Security, and Business Logic. **The DODODEX project passed all audit items. The overall result is Pass. The smart contract is able to function properly.**

1. Coding Conventions

Check the code style that does not conform to Solidity code style.

1.1 Compiler Version Security

- Description: Check whether the code implementation of current contract contains the exposed solidity compiler bug.
- Result: Pass

1.2 Deprecated Items

- Description: Check whether the current contract has the deprecated items.
- Result: Pass

1.3 Redundant Code

- Description: Check whether the contract code has redundant codes.
- Result: Pass

1.4 SafeMath Features

- Description: Check whether the SafeMath has been used. Or prevents the integer overflow/underflow in mathematical operation.
- Result: Pass

1.5 require/assert Usage

- Description: Check the use reasonability of 'require' and 'assert' in the contract.
- Result: Pass

1.6 Gas Consumption

- Description: Check whether the gas consumption exceeds the block gas limitation.
- Result: Pass

1.7 Visibility Specifiers

- Description: Check whether the visibility conforms to design requirement.
- Result: Pass

1.8 Fallback Usage

- Description: Check whether the Fallback function has been used correctly in the current contract.
- Result: Pass

2. General Vulnerability

Check whether the general vulnerabilities exist in the contract.

2.1 Integer Overflow/Underflow

- Description: Check whether there is an integer overflow/underflow in the contract and the calculation result is abnormal.
- Result: Pass

2.2 Reentrancy

- Description: An issue when code can call back into your contract and change state, such as withdrawing ETH.
- Result: Pass

2.3 Pseudo-random Number Generator (PRNG)

- Description: Whether the results of random numbers can be predicted.
- Result: Pass

2.4 Transaction-Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Pass

2.5 DoS (Denial of Service)

- Description: Whether exist DoS attack in the contract which is vulnerable because of unexpected reason.
- Result: Pass

2.6 Access Control of Owner

- Description: Whether the owner has excessive permissions, such as malicious issue, modifying the balance of others.
- Result: Pass

2.7 Low-level Function (call/delegatecall) Security

- Description: Check whether the usage of low-level functions like call/delegatecall have vulnerabilities.
- Result: Pass

2.8 Returned Value Security

- Description: Check whether the function checks the return value and responds to it accordingly.
- Result: Pass

2.9 tx.origin Usage

- Description: Check the use secure risk of 'tx.origin' in the contract. In this project, the contract
- Result: Pass

2.10 Replay Attack

- Description: Check whether the implement possibility of Replay Attack exists in the contract.
- Result: Pass

2.11 Overriding Variables

- Description: Check whether the variables have been overridden and lead to wrong code execution.
- Result: Pass

3. Business Security

In the two contracts given, the CP contract implements the crowdfunding function. Users can create a CP contract to list their own BASE tokens for crowdfunding, and participants can subscribe for BASE tokens through the CP contract. After the subscription is over, anyone can initiate a transaction to create the corresponding DVM contract. The DVM contract realizes the transaction of BASE token and QUOTE token, as well as the function of adding liquidity.

3.1 Business analysis of Contract CP

(1) CP

The CP contract implements the initialization function of the contract, and the creator can call the *init* function to initialize the crowdfunding contract.

```
function init(
    address[] calldata addressList,
    uint256[] calldata timeLine,
    uint256[] calldata valueList,
    bool isOpenTWAP
) external {
    /*
    Address List
    0. owner
    1. maintainer
    2. baseToken
    3. quoteToken
    4. permissionManager
    5. feeRateModel
    6. poolFactory
     */

    require(addressList.length == 7, "LIST_LENGTH_WRONG");

    initOwner(addressList[0]);
    _MAINTAINER_ = addressList[1];
    _BASE_TOKEN_ = IERC20(addressList[2]);
    _QUOTE_TOKEN_ = IERC20(addressList[3]);
    _BIDDER_PERMISSION_ = IPermissionManager(addressList[4]);
    _MT_FEE_RATE_MODEL_ = IFeeRateModel(addressList[5]);
    _POOL_FACTORY_ = addressList[6];

    /*
    Time Line
    0. phase bid starttime
    1. phase bid duration
    2. phase calm duration
    3. freeze duration
    4. vesting duration
    */

    require(timeLine.length == 5, "LIST_LENGTH_WRONG");

    _PHASE_BID_STARTTIME_ = timeLine[0];
    _PHASE_BID_ENDTIME_ = _PHASE_BID_STARTTIME_.add(timeLine[1]);
    _PHASE_CALM_ENDTIME_ = _PHASE_BID_ENDTIME_.add(timeLine[2]);

    _FREEZE_DURATION_ = timeLine[3];
    _VESTING_DURATION_ = timeLine[4];

    require(block.timestamp <= _PHASE_BID_STARTTIME_, "TIMELINE_WRONG");

    /*
    Value List
    0. pool quote cap
    1. k
    2. i
    3. cliff rate
    */

    require(valueList.length == 4, "LIST_LENGTH_WRONG");

    _POOL_QUOTE_CAP_ = valueList[0];
    _K_ = valueList[1];
    _I_ = valueList[2];
    _CLIFF_RATE_ = valueList[3];

    require(_I_ > 0 && _I_ <= 1e36, "I_VALUE_WRONG");
    require(_K_ <= 1e18, "K_VALUE_WRONG");
    require(_CLIFF_RATE_ <= 1e18, "CLIFF_RATE_WRONG");

    _TOTAL_BASE_ = _BASE_TOKEN_.balanceOf(address(this));

    _IS_OPEN_TWAP_ = isOpenTWAP;

    require(address(this).balance == _SETTEL_FUND_, "SETTLE_FUND_NOT_MATCH");
}
```

Figure 1 Source code of *init*

- Related functions: *init*
- Result: Pass

(2) CPStorage

The CPStorage contract mainly stores contract-related variables and implements modifier.

```
80      modifier phaseBid() {
81          require(
82              block.timestamp >= _PHASE_BID_STARTTIME_ && block.timestamp < _PHASE_BID_ENDTIME_,
83              "NOT_PHASE_BID"
84          );
85          _;
86      }
87
88      modifier phaseCalm() {
89          require(
90              block.timestamp >= _PHASE_BID_ENDTIME_ && block.timestamp < _PHASE_CALM_ENDTIME_,
91              "NOT_PHASE_CALM"
92          );
93          _;
94      }
95
96      modifier phaseBidOrCalm() {
97          require(
98              block.timestamp >= _PHASE_BID_STARTTIME_ && block.timestamp < _PHASE_CALM_ENDTIME_,
99              "NOT_PHASE_BID_OR_CALM"
100         );
101         _;
102     }
103
104     modifier phaseSettlement() {
105         require(block.timestamp >= _PHASE_CALM_ENDTIME_, "NOT_PHASE_EXE");
106         _;
107     }
108
109     modifier phaseVesting() {
110         require(_SETTLED_, "NOT_VESTING");
111         _;
112     }
```

Figure 2 Source code of *modifier*

- Related functions: *phaseBid*, *phaseCalm*, *phaseBidOrCalm*, *phaseSettlement*, *phaseVesting*
- Safe suggestion: phaseCalm, phaseVesting is not used, it is recommended to delete
- Result: Ignore
- Result: Pass

(3) CPFunding

The CPFunding contract implements the functions of *bid*, *cancel*, *settle* and *emergencySettle* Users can subscribe for BASE tokens with QUOTE tokens by calling the *bid* function, and the handling fee needs to be deducted. Within the specified time, users can also call the *cancel* function at any time to redeem their own

QUTOE tokens. After the subscription period is over, anyone can call the *settle* function to deploy the DVM contract. In addition, the contract provides *emergencySettle* function to ensure the security of QUOTE tokens.

```
37      function bid(address to) external phaseBid preventReentrant isBidderAllow(to) {
38          uint256 input = _getQuoteInput();
39          uint256 mtFee = DecimalMath.mulFloor(input, _MT_FEE_RATE_MODEL_.getFeeRate(to));
40          _transferQuoteOut(_MAINTAINER_, mtFee);
41          _mintShares(to, input.sub(mtFee));
42          _sync();
43          emit Bid(to, input, mtFee);
44      }
```

Figure 3 Source code of *bid*

```
46  function cancel(address to, uint256 amount, bytes calldata data) external phaseBidOrCalm preventReentrant {
47      require(_SHARES_[msg.sender] >= amount, "SHARES_NOT_ENOUGH");
48      _burnShares(msg.sender, amount);
49      _transferQuoteOut(to, amount);
50      _sync();
51
52      if(data.length > 0){
53          IDODOCallee(to).CPCancelCall(msg.sender,amount,data);
54      }
55
56      emit Cancel(msg.sender,amount);
57  }
```

Figure 4 Source code of *cancel*

```
71      function settle() external phaseSettlement preventReentrant {
72          _settle();
73
74          (uint256 poolBase, uint256 poolQuote, uint256 poolI, uint256 unUsedBase, uint256 unUsedQuote) = getSettleResult();
75          _UNUSED_BASE_ = unUsedBase;
76          _UNUSED_QUOTE_ = unUsedQuote;
77
78          address _poolBaseToken;
79          address _poolQuoteToken;
80
81          if (_UNUSED_BASE_ > poolBase) {
82              _poolBaseToken = address(_QUOTE_TOKEN_);
83              _poolQuoteToken = address(_BASE_TOKEN_);
84          } else {
85              _poolBaseToken = address(_BASE_TOKEN_);
86              _poolQuoteToken = address(_QUOTE_TOKEN_);
87          }
88
89          _POOL_ = IDVMFactory(_POOL_FACTORY_).createDODOVendingMachine(
90              _poolBaseToken,
91              _poolQuoteToken,
92              3e15, // 0.3% lp feeRate
93              poolI,
94              DecimalMath.ONE,
95              _IS_OPEN_TWAP_
96          );
97
98          uint256 avgPrice = unUsedBase == 0 ? _I_ : DecimalMath.divCeil(poolQuote, unUsedBase);
99          _AVG_SETTLED_PRICE_ = avgPrice;
100
101         _transferBaseOut(_POOL_, poolBase);
102         _transferQuoteOut(_POOL_, poolQuote);
103
104         _TOTAL_LP_AMOUNT_ = IDVM(_POOL_).buyShares(address(this));
105
106         msg.sender.transfer(_SETTEL_FUND_);
107
108         emit Settle();
109     }
```

Figure 5 Source code of *settle*

```
111     // in case something wrong with base token contract
112     function emergencySettle() external phaseSettlement preventReentrant {
113         require(block.timestamp >= _PHASE_CALM_ENDTIME_.add(_SETTLEMENT_EXPIRE_), "NOT_EMERGENCY");
114         _settle();
115         _UNUSED_QUOTE_ = _QUOTE_TOKEN_.balanceOf(address(this));
116     }
```

Figure 6 Source code of *emergencySettle*

- Related functions: *bid*, *cancel*, *settle*, *emergencySettle*
- Result: Pass

(4) CPVesting

The CPVesting contract implements the related functions after DVM deployment. Users participating in the subscription can obtain the subscribed BASE tokens and recover the excess QUOTE tokens by calling the *bidderClaim* function. The initiator can obtain liquidity tokens by calling the *claimLPToken* function.

```
50      function bidderClaim(address to,bytes calldata data) external afterSettlement {
51          require(!_CLAIMED_[msg.sender], "ALREADY_CLAIMED");
52          _CLAIMED_[msg.sender] = true;
53
54          uint256 baseAmount = _UNUSED_BASE_.mul(_SHARES_[msg.sender]).div(_TOTAL_SHARES_);
55          uint256 quoteAmount = _UNUSED_QUOTE_.mul(_SHARES_[msg.sender]).div(_TOTAL_SHARES_);
56
57          _transferBaseOut(to, baseAmount);
58          _transferQuoteOut(to, quoteAmount);
59
60          if(data.length>0){
61              IDODOCallee(to).CPClaimBidCall(msg.sender,baseAmount,quoteAmount,data);
62          }
63
64          emit Claim(msg.sender, baseAmount, quoteAmount);
65      }
```

Figure 7 Source code of *bidderClaim*

```
69      function claimLPToken() external onlyOwner afterFreeze {
70          uint256 lpAmount = getClaimableLPToken();
71          IERC20(_POOL_).safeTransfer(_OWNER_, lpAmount);
72          emit ClaimLP(lpAmount);
73      }
```

Figure 8 Source code of *claimLPToken*

- Related functions: *bidderClaim*, *claimLPToken*, *getClaimableLPToken*, *getRemainingLPRatio*
- Result: Pass

3.2 Business analysis of Contract DVM

(1) DVM

The DVM contract mainly implements the initialization function of the contract, and the creator can call the init function to initialize the contract.

```
577     function init(
578         address maintainer,
579         address baseTokenAddress,
580         address quoteTokenAddress,
581         uint256 lpFeeRate,
582         address mtFeeRateModel,
583         uint256 i,
584         uint256 k,
585         bool isOpenTWAP
586     ) external {
587         require(!_DVM_INITIALIZED_, "DVM_INITIALIZED");
588         _DVM_INITIALIZED_ = true;
589 
590         require(baseTokenAddress != quoteTokenAddress, "BASE_QUOTE_CAN_NOT_BE_SAME");
591         _BASE_TOKEN_ = IERC20(baseTokenAddress);
592         _QUOTE_TOKEN_ = IERC20(quoteTokenAddress);
593 
594         require(i > 0 && i <= 10**36);
595         _I_ = i;
596 
597         require(k <= 10**18);
598         _K_ = k;
599 
600         _LP_FEE_RATE_ = lpFeeRate;
601         _MT_FEE_RATE_MODEL_ = IFeeRateModel(mtFeeRateModel);
602         _MAINTAINER_ = maintainer;
603 
604         _IS_OPEN_TWAP_ = isOpenTWAP;
605         if(isOpenTWAP) _BLOCK_TIMESTAMP_LAST_ = uint32(block.timestamp % 2**32);
606 
607         string memory connect = "_";
608         string memory suffix = "DLP";
609 
610         name = string(abi.encodePacked(suffix, connect, addressToShortString(address(this))));
611         symbol = "DLP";
612         decimals = _BASE_TOKEN_.decimals();
613 
614         // ============================== Permit ====================================
615         uint256 chainId;
616         assembly {
617             chainId := chainid()
618         }
619         DOMAIN_SEPARATOR = keccak256(
620             abi.encode(
621                 // keccak256('EIP712Domain(string name,string version,uint256 chainId,address v
622                 0x8b73c3c69bb8fe3d512ecc4cf759cc79239f7b179b0ffacaa9a75d522b39400f,
623                 keccak256(bytes(name)),
624                 keccak256(bytes("1")),
625                 chainId,
626                 address(this)
627             )
628         );
629         // ==========================================================================
630     }
```

Figure 9 Source code of *init*

- Related functions: *init*, *version*, *addressToString*
- Result: Pass

(2) DVMStorage

The DVMStorage contract mainly stores contract-related variables.

(3) DVMTrader

The DVMTrader contract implements the buy and sell of BASE tokens, and also implements the flashLoan function.

```
41      function sellBase(address to)
42          external
43          preventReentrant
44          returns (uint256 receiveQuoteAmount)
45      {
46          uint256 baseBalance = _BASE_TOKEN_.balanceOf(address(this));
47          uint256 baseInput = baseBalance.sub(uint256(_BASE_RESERVE_));
48          uint256 mtFee;
49          (receiveQuoteAmount, mtFee) = querySellBase(tx.origin, baseInput);
50
51          _transferQuoteOut(to, receiveQuoteAmount);
52          _transferQuoteOut(_MAINTAINER_, mtFee);
53          _setReserve(baseBalance, _QUOTE_TOKEN_.balanceOf(address(this)));
54
55          emit DODOSwap(
56              address(_BASE_TOKEN_),
57              address(_QUOTE_TOKEN_),
58              baseInput,
59              receiveQuoteAmount,
60              msg.sender,
61              to
62          );
63      }
```

Figure 10 Source code of *sellBase*

```
65      function sellQuote(address to)
66          external
67          preventReentrant
68          returns (uint256 receiveBaseAmount)
69      {
70          uint256 quoteBalance = _QUOTE_TOKEN_.balanceOf(address(this));
71          uint256 quoteInput = quoteBalance.sub(uint256(_QUOTE_RESERVE_));
72          uint256 mtFee;
73          (receiveBaseAmount, mtFee) = querySellQuote(tx.origin, quoteInput);
74
75          _transferBaseOut(to, receiveBaseAmount);
76          _transferBaseOut(_MAINTAINER_, mtFee);
77          _setReserve(_BASE_TOKEN_.balanceOf(address(this)), quoteBalance);
78
79          emit DODOSwap(
80              address(_QUOTE_TOKEN_),
81              address(_BASE_TOKEN_),
82              quoteInput,
83              receiveBaseAmount,
84              msg.sender,
85              to
86          );
87      }
```

Figure 11 Source code of *sellQuote*

```
89  function flashLoan(
90      uint256 baseAmount,
91      uint256 quoteAmount,
92      address assetTo,
93      bytes calldata data
94  ) external preventReentrant {
95      _transferBaseOut(assetTo, baseAmount);
96      _transferQuoteOut(assetTo, quoteAmount);
97
98      if (data.length > 0)
99          IDODOCallee(assetTo).DVMFlashLoanCall(msg.sender, baseAmount, quoteAmount, data);
100
101     uint256 baseBalance = _BASE_TOKEN_.balanceOf(address(this));
102     uint256 quoteBalance = _QUOTE_TOKEN_.balanceOf(address(this));
103
104     // no input -> pure loss
105     require(
106         baseBalance >= _BASE_RESERVE_ || quoteBalance >= _QUOTE_RESERVE_,
107         "FLASH_LOAN_FAILED"
108     );
109
110     // sell quote
111     if (baseBalance < _BASE_RESERVE_) {
112         uint256 quoteInput = quoteBalance.sub(uint256(_QUOTE_RESERVE_));
113         (uint256 receiveBaseAmount, uint256 mtFee) = querySellQuote(tx.origin, quoteInput);
114         require(uint256(_BASE_RESERVE_).sub(baseBalance) <= receiveBaseAmount, "FLASH_LOAN_FAILED");
115
116         _transferBaseOut(_MAINTAINER_, mtFee);
117         emit DODOSwap(
118             address(_QUOTE_TOKEN_),
119             address(_BASE_TOKEN_),
120             quoteInput,
121             receiveBaseAmount,
122             msg.sender,
123             assetTo
124         );
125     }
126
127     // sell base
128     if (quoteBalance < _QUOTE_RESERVE_) {
129         uint256 baseInput = baseBalance.sub(uint256(_BASE_RESERVE_));
130         (uint256 receiveQuoteAmount, uint256 mtFee) = querySellBase(tx.origin, baseInput);
131         require(uint256(_QUOTE_RESERVE_).sub(quoteBalance) <= receiveQuoteAmount, "FLASH_LOAN_FAILED"
132
133         _transferQuoteOut(_MAINTAINER_, mtFee);
134         emit DODOSwap(
135             address(_BASE_TOKEN_),
136             address(_QUOTE_TOKEN_),
137             baseInput,
138             receiveQuoteAmount,
139             msg.sender,
140             assetTo
141         );
142     }
143
144     _sync();
145
146     emit DODOFlashLoan(msg.sender, assetTo, baseAmount, quoteAmount);
147 }
```

Figure 12 Source code of *flashLoan*

- Related functions: *flashLoan*, *querySellBase*, *querySellQuote*, *sellBase*, *sellQuote*

- Result: Pass

(4) DVMFunding

DVMFunding contract implements adding and removing liquidity.

```
25  function buyShares(address to)
26      external
27      preventReentrant
28      returns (
29          uint256 shares,
30          uint256 baseInput,
31          uint256 quoteInput
32      )
33  {
34      uint256 baseBalance = _BASE_TOKEN_.balanceOf(address(this));
35      uint256 quoteBalance = _QUOTE_TOKEN_.balanceOf(address(this));
36      uint256 baseReserve = _BASE_RESERVE_;
37      uint256 quoteReserve = _QUOTE_RESERVE_;
38
39      baseInput = baseBalance.sub(baseReserve);
40      quoteInput = quoteBalance.sub(quoteReserve);
41      require(baseInput > 0, "NO_BASE_INPUT");
42
43      // Round down when withdrawing. Therefore, never be a situation occuring balance is 0 but totals
44      // But May Happen, reserve >0 But totalSupply = 0
45      if (totalSupply == 0) {
46          // case 1. initial supply
47          require(baseBalance >= 10**3, "INSUFFICIENT_LIQUIDITY_MINED");
48          shares = baseBalance;
49      } else if (baseReserve > 0 && quoteReserve == 0) {
50          // case 2. supply when quote reserve is 0
51          shares = baseInput.mul(totalSupply).div(baseReserve);
52      } else if (baseReserve > 0 && quoteReserve > 0) {
53          // case 3. normal case
54          uint256 baseInputRatio = DecimalMath.divFloor(baseInput, baseReserve);
55          uint256 quoteInputRatio = DecimalMath.divFloor(quoteInput, quoteReserve);
56          uint256 mintRatio = quoteInputRatio < baseInputRatio ? quoteInputRatio : baseInputRatio;
57          shares = DecimalMath.mulFloor(totalSupply, mintRatio);
58      }
59      _mint(to, shares);
60      _setReserve(baseBalance, quoteBalance);
61      emit BuyShares(to, shares, _SHARES_[to]);
62  }
```

Figure 13 Source code of *buyShares*

```
65  function sellShares(
66      uint256 shareAmount,
67      address to,
68      uint256 baseMinAmount,
69      uint256 quoteMinAmount,
70      bytes calldata data,
71      uint256 deadline
72  ) external preventReentrant returns (uint256 baseAmount, uint256 quoteAmount) {
73      require(deadline >= block.timestamp, "TIME_EXPIRED");
74      require(shareAmount <= _SHARES_[msg.sender], "DLP_NOT_ENOUGH");
75      uint256 baseBalance = _BASE_TOKEN_.balanceOf(address(this));
76      uint256 quoteBalance = _QUOTE_TOKEN_.balanceOf(address(this));
77      uint256 totalShares = totalSupply;
78
79      baseAmount = baseBalance.mul(shareAmount).div(totalShares);
80      quoteAmount = quoteBalance.mul(shareAmount).div(totalShares);
81
82      require(
83          baseAmount >= baseMinAmount && quoteAmount >= quoteMinAmount,
84          "WITHDRAW_NOT_ENOUGH"
85      );
86
87      _burn(msg.sender, shareAmount);
88      _transferBaseOut(to, baseAmount);
89      _transferQuoteOut(to, quoteAmount);
90      _sync();
91
92      if (data.length > 0) {
93          IDODOCallee(to).DVMSellShareCall(
94              msg.sender,
95              shareAmount,
96              baseAmount,
97              quoteAmount,
98              data
99          );
100     }
101
102     emit SellShares(msg.sender, to, shareAmount, _SHARES_[msg.sender]);
103 }
```

Figure 14 Source code of *sellShares*

- Related functions: *buyShares*, *sellShares*
- Result: Pass

(5) DVMVault

The DVMVault contract implements ERC20 tokens, which support offline approve.

```
198     function permit(
199         address owner,
200         address spender,
201         uint256 value,
202         uint256 deadline,
203         uint8 v,
204         bytes32 r,
205         bytes32 s
206     ) external {
207         require(deadline >= block.timestamp, "DODO_DVM_LP: EXPIRED");
208         bytes32 digest = keccak256(
209             abi.encodePacked(
210                 "\x19\x01",
211                 DOMAIN_SEPARATOR,
212                 keccak256(
213                     abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++, deadline)
214                 )
215             )
216         );
217         address recoveredAddress = ecrecover(digest, v, r, s);
218         require(
219             recoveredAddress != address(0) && recoveredAddress == owner,
220             "DODO_DVM_LP: INVALID_SIGNATURE"
221         );
222         _approve(owner, spender, value);
223     }
```

- Related functions: *buyShares*, *sellShares*
- Result: Pass

Figure 15 Source code of *permit*

4. Conclusion

Beosin(ChengduLianAn) conducted a detailed audit on the design and code implementation of the smart contracts DVM and CP. The DVM, CP contracts of project DODODEX passed all audit items, The overall audit result is **Pass.**

Official Website

https://lianantech.com

E-mail

vaas@lianantech.com

Twitter

https://twitter.com/Beosin_com