BEOSIN
Blockchain Security

Smart contract audit report

for

DOTC

Audit Number: 202202281850

Project Name: DOTC

Deployment Platform: Ethereum, BNB Chain, Huobi Eco Chain, Polygon Chain, TRON

Audit Project Link:

https://github.com/DOTCPro/Contracts

Initial Commit ID: b1785dcd51678fc34456bf35f6bb62000207410e

Final Commit ID: 6887e1312b39ca7be7e06bf9da6de8df69aa99e0

Audit Start Date: 2021.12.01

Audit Completion Date: 2022.02.28

Audit Team: Beosin Technology Co. Ltd.

Audit Results Explained

Beosin Technology has used several methods including Formal Verification, Static Analysis, Typical Case Testing and Manual Review to audit three major aspects of DOTC project (Incremental audit of commits from b1785dcd51678fc34456bf35f6bb62000207410e to 6887e1312b39ca7be7e06bf9da6de8df69aa99e0), including Coding Conventions, General Vulnerability and Business Security. **After auditing, the DOTC project was found to have 2 Critical-risk items, 3 High-risk items, 2 Medium-risk items, 8 Low-risk items and 8 Info items.** The following is the detailed audit information for this project.

Index	Risk items	Risk level	Status
DOTC-1	Db.lendTable is not instantiated, causing some contracts to not working normally	Critical	Fixed
DOTC-2	When the pledged user unlocks the pledged assets, the logic is chaotic	Critical	Fixed
DOTC-3	Unreasonable data recording methods may cause data confusion	High	Partially Fixed
DOTC-4	Adding global variables in each facet contract may cause data to be overwritten	High	Fixed
DOTC-5	The second appeal will record the number of arbitrations repeatedly, which may cause some functions to be unusable	High	Fixed
DOTC-6	In the *backLendAssets* function, the data returned by the event is wrong	Medium	Fixed
DOTC-7	The detail.lockedAmount of the corresponding Ad order was not updated correctly when the Ex order was completed	Medium	Fixed
DOTC-8	When the balance of the referrer of the losing party is insufficient to pay the penalty, it will not be punished	Low	Fixed
DOTC-9	If the nPeriodCoun in the contract is too large, it will cause calculation errors	Low	Fixed
DOTC-10	Avoid empty characters in the order id, which will affect certain judgment logic of the contract	Low	Fixed
DOTC-11	In the *_backToken* function of the DOTCLendBase contract, it is required that the number of tokens available to the user must be greater than the number of tokens that should be returned	Low	Fixed
DOTC-12	When the user withdraws the pledged assets, the corresponding pledge data will not be cleared, which will affect the query data	Low	Acknowledged
DOTC-13	The *transfer* function failed due to the irregular implementation of USDT on the corresponding platform	Low	Fixed
DOTC-14	The remaining deposit counted by the *_checkAdOrderLeftDeposit* function is inaccurate	Low	Acknowledged
DOTC-15	"else if" in queryUserLendRate function should be "if"	Low	Fixed
DOTC-16	When creating an arbitration in the DOTCArbitFacet contract, the id entered by the user is used as the index record, and there may be data coverage problems.	Info	Acknowledged
DOTC-17	In the DOTCArbitFacet contract, when checking the input parameter for creating an arbitration, the orderArbitTimes should be less than 2	Info	Fixed
DOTC-18	*_checkAdOrder* function comment error in DOTCAdOrderFacet contract	Info	Fixed

DOTC-19	There is some redundant code in the contract	Info	Fixed
DOTC-20	When the arbitration data is deleted in the *_removeArbiterFromDB* function, only delete is used, and the corresponding array length is not reduced	Info	Fixed
DOTC-21	The *tokenDeposit* function in the DOTCUserFacet contract declares the payable keyword, but the contract does not need to receive platform token	Info	Fixed
DOTC-22	The losing party's assets will send half of the traded assets to risk pools, there may be precision issues	Info	Acknowledged
DOTC-23	Compiler warning	Info	Acknowledged

Risk descriptions:

Item **DOTC-3** is partial fixed, in special cases, it may still lead to confusion in the arbitration data.

Item **DOTC-12** is not fixed. This affects the query data of pledged users.

Item **DOTC-14** is not fixed. This results in inaccurate results from the *_checkAdOrderLeftDeposit* function.

Item **DOTC-16** is not fixed. If there is a duplicate arbitration id in the project, the arbitration queried based on this ID will be new, and the old arbitration cannot be queried.

Item **DOTC-22** is not fixed. This will have a small impact on token holdings of pools A and B in the contract.

Item **DOTC-23** is not fixed. This does not affect contract operation.

Findings

[DOTC-1 Critical] Db.lendTable is not instantiated, causing some contracts to not working normally

- **Description:** As shown in the figure below, the lend module is added in the new version, but it is not declared in the corresponding "db", causing some contracts not working normally.

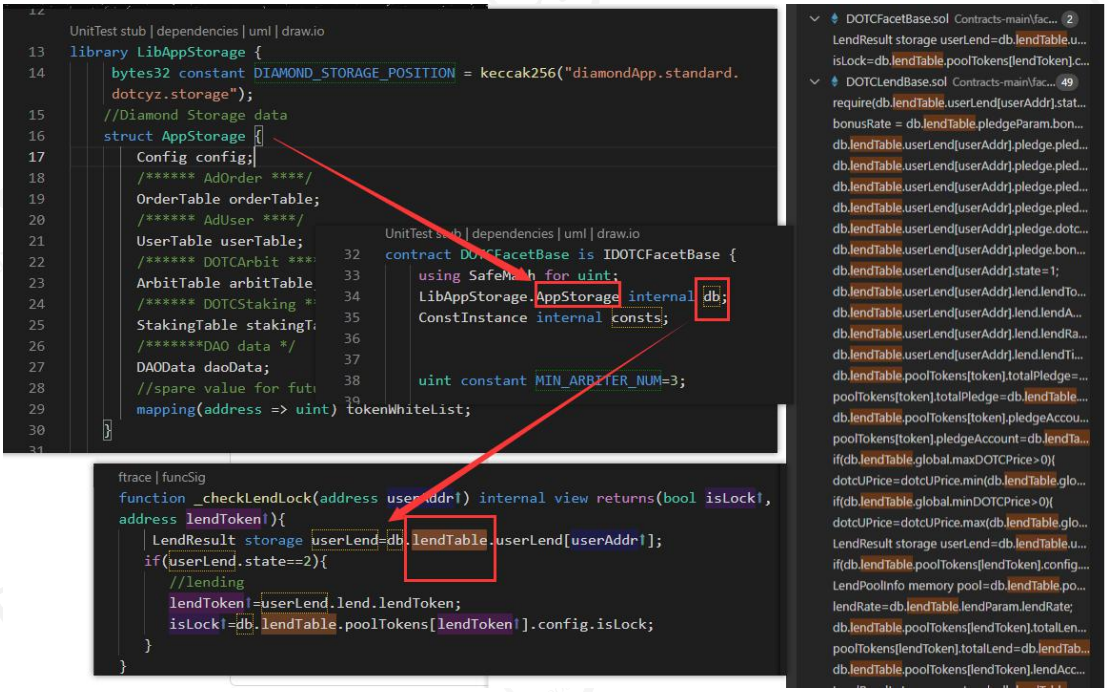

Figure 1 "db" declaration and use

- **Fix recommendations:** Declare the lend module in the AppStorage structure.
- **Status: Fixed.**

[DOTC-2 Critical] When the pledged user unlocks the pledged assets, the logic is chaotic

- **Description:** As shown in the figure below, the code logic of the *_unlockPledgeAmount* function in the DOTCLendBase contract is rather confusing:

  (1) According to the code logic, the asset amount currentSupply in the risk pool has not been updated, while the user's available asset amount has increased; (Line 188)

  (2) At line 189, the pledgeAmount is greater than myBalance, so the subtraction here will fail;

  (3) Line 199 is refunded again, and the refund amount is backDotcAmount, without subtracting the refunded part of the risk pool above.

```
180             //normal
181             db.userTable.userAssets[userAddr][pledgeToken].available=db.userTable.userAssets[userAddr][pledgeToken].available.add(pledgeAmount);
182             db.lendTable.poolTokens[pledgeToken].totalPledge=myBalance.sub(pledgeAmount);
183           }else{
184             //not enough
185             //back from risk pool
186             uint riskBalance=db.daoData.riskPool.poolTokens[pledgeToken].currentSupply;
187             uint backAmount=pledgeAmount.min(riskBalance);
188             db.userTable.userAssets[userAddr][pledgeToken].available=db.userTable.userAssets[userAddr][pledgeToken].available.add(backAmount);
189             db.lendTable.poolTokens[pledgeToken].totalPledge=myBalance.sub(pledgeAmount);
190             if(pledgeAmount>backAmount){
191               if(pledgeToken==db.config.usdtContract){
192                 //USDT,back dotc
193                 uint riskPrice=db.lendTable.global.riskPrice;
194                 require(riskPrice>0,'riskPrice zero');
195                 require(block.timestamp<db.lendTable.global.priceTime.add(PERIOD_RISK),'Risk price overTime');
196                 uint riskDotcBalance=db.daoData.riskPool.poolTokens[db.config.dotcContract].currentSupply;
197                 uint dotcAmount= pledgeAmount.mul(riskPrice).div(consts.priceParam.nUsdtDecimals);
198                 uint backDotcAmount=dotcAmount.min(riskDotcBalance);
199                 db.userTable.userAssets[userAddr][db.config.dotcContract].available=db.userTable.userAssets[userAddr][db.config.dotcContract].available.add(backDotcAmount)
200               }else if(pledgeToken==db.config.dotcContract){
201                 //DOTC
202
```

Figure 2 Source code of function *_unlockPledgeAmount*

- **Fix recommendations:** According to business needs, rewrite the design of the function.
- **Status: Fixed.**

```
{
    //start unlock
    pledgeToken=userLend.pledge.pledgeToken;
    pledgeAmount=userLend.pledge.pledgeAmount;
    uint poolBalance =db.lendTable.poolTokens[pledgeToken].totalPledge.sub(db.lendTable.poolTokens[pledgeToken].totalLend);
    require(poolBalance >= pledgeAmount,'Insufficient available balance in the loan pool');

   /*  if(poolBalance >=pledgeAmount){*/
      //normal
    db.userTable.userAssets[userAddr][pledgeToken].available=db.userTable.userAssets[userAddr][pledgeToken].available.add(pledgeAmount);
    db.lendTable.poolTokens[pledgeToken].totalPledge= db.lendTable.poolTokens[pledgeToken].totalPledge.sub(pledgeAmount);
   /* }else{
      //not enough
      //back from risk pool
      uint riskBalance=db.daoData.riskPool.poolTokens[pledgeToken].currentSupply;
      uint backAmount=pledgeAmount.min(riskBalance);
      db.userTable.userAssets[userAddr][pledgeToken].available=db.userTable.userAssets[userAddr][pledgeToken].available.add(backAmount)
      db.lendTable.poolTokens[pledgeToken].totalPledge= poolBalance.sub(pledgeAmount);
      if(pledgeAmount>backAmount){
        if(pledgeToken==db.config.usdtContract){
          //USDT,back dotc
          uint riskPrice=db.lendTable.global.riskPrice;
          require(riskPrice>0,'riskPrice zero');
          require(block.timestamp<db.lendTable.global.priceTime.add(PERIOD_RISK),'Risk price overTime');
          uint riskDotcBalance=db.daoData.riskPool.poolTokens[db.config.dotcContract].currentSupply;
          uint dotcAmount= pledgeAmount.mul(riskPrice).div(consts.priceParam.nUsdtDecimals);
          uint backDotcAmount=dotcAmount.min(riskDotcBalance);
          db.userTable.userAssets[userAddr][db.config.dotcContract].available=db.userTable.userAssets[userAddr][db.config.dotcContrac
        }else if(pledgeToken==db.config.dotcContract){
          //DOTC
        }
      }
    }*/
}
```

Figure 3 Source code of function *_unlockPledgeAmount* (fixed)

[DOTC-3 High] Unreasonable data recording methods may cause data confusion

- **Description:** As shown in the figure below, in the DOTCArbitFacet contract, the arbitration record method corresponding to the order is indexed by the id of the corresponding Ex order. However, in fact, there may be cases where the same Ex order id is used in multiple Ad orders during arbitration, which will cause confusion in data records and even DoS attacks.

```
function createOrderArbit(string calldata adOrderId,string calldata exOrderId,string calldata arbitId) external returns (bool result,uint period) {
  require(!db.config.isPause,'system paused');
  _checkExArbitApply(adOrderId,exOrderId,msg.sender);
  uint ncheckResult=_checkExArbitAccess(exOrderId,msg.sender);
  require(ncheckResult!=0,'you can not apply arbit now.');
  if(ncheckResult==1){
    //settle first cost
    if(db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.orderArbitTimes>0){
      if(db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.lockedDotcAmount>0){

        address payUser=db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.lockedUser;
        uint amount=db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.lockedDotcAmount;
        db.userTable.userAssets[payUser][db.config.dotcContract].locked=db.userTable.userAssets[payUser][db.config.dotcContract].locked.sub(amount);
        db.arbitTable.extend.arbitGivedToken[db.config.dotcContract]=db.arbitTable.extend.arbitGivedToken[db.config.dotcContract].add(amount);
        db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.lockedDotcAmount=0;
      }
    }
    if(consts.arbitParam.nOrderArbitCost>0){
      uint dotcCost=_getDOTCNumFromUSDT(consts.arbitParam.nOrderArbitCost);
      _lockToken(msg.sender,db.config.dotcContract,dotcCost);
      db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.lockedDotcAmount=dotcCost;
      db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.lockedUser=msg.sender;
    }
  }
  //create arbit
```

Figure 4 Source code of function *createOrderArbit*

- **Fix recommendations:** Select a unique parameter as the index of the data.
- **Status: Partially Fixed.** As shown in the figure below, the contract restricts that the Ex order id must contain the Ad order id, but it is still necessary to pay attention to the situation that one Ad order id contains another Ad order id.

```
function _checkExOrder(string memory adOrderId,string memory exOrderId,uint amount) internal view {
  require(db.orderTable.otcAdOrders[adOrderId].makerAddress !=address(0),'AdOrder not exists');
  require(db.orderTable.otcAdOrders[adOrderId].state == OrderState.ONTRADE,'AdOrder has been closed');
  require(db.orderTable.otcAdOrders[adOrderId].makerAddress != msg.sender,'you can not trade with yourself');

  require(StringUtils.indexOf(exOrderId,adOrderId) == 0, 'Invalid exOrderId prefix');

  if(db.userTable.userList[msg.sender].isVIP){
    require(db.orderTable.userOrderDb[msg.sender].noneExOrder<consts.orderLimit.vipOrderNum,'ExOrder VIP Limit');
  }else{
    require(db.orderTable.userOrderDb[msg.sender].noneExOrder<consts.orderLimit.orderNum,'ExOrder Limit');
  }

  if (uint(db.orderTable.otcAdOrders[adOrderId].side) == 0 && db.orderTable.otcAdOrders[adOrderId].tokenA == db.config.dotcContract) {
    (bool isLock,address lendToken) = _checkLendLock(msg.sender);
    require(lendToken != db.config.dotcContract, "Lending DOTC can not be sold");
  }
```

Figure 5 Part source code of function *_checkExOrder*

[DOTC-4 High] Adding global variables in each facet contract may cause data to be overwritten

- **Description:** This project uses the same storage contract to store data, and its basic storage template is the DOTCFeeFacet contract. Data coverage may exist for the newly added variables of some logical contracts. For example, the isInited variable is added to the DOTCFactoryDiamond contract to store whether the contract is initialized. After the contract is initialized, the value of the isInited is true. At this time, when the DOTCManageFacet logical contract is used, its isManageInited will directly become true. (there is also a type issue with the new variables in the DOTCOracleFacet contract)

```
contract DOTCFactoryDiamond is IDOTCFactoryDiamond{

    LibAppStorage.AppStorage internal db;
    ConstInstance internal consts;
    bool internal isInited;

    constructor() {

    }
    function InitContract(
        IDiamondCut.FacetCut[] memory _diamondCut,
        address _owner,
        address _dotcContract,
        address _wethContract,
        address _usdtContract) external{
        if(!isInited){
          _checkParam(_diamondCut,_owner,_dotcContract,_wethContract,_usdtContract);
          _initInterface();
          isInited=true;
        }
```

Figure 6 Part source code of contract DOTCFactoryDiamond

```
contract DOTCManageFacet is DOTCFacetBase,IDOTCManageFacet {
    using SafeMath for uint;

    bool internal isManageInited;

    function setContractManager(address _newManager) external returns(bool result) {
        LibDiamond.enforceIsContractOwner();
        LibDiamond.DiamondStorage storage ds = LibDiamond.diamondStorage();
        address previousManager = ds.contractManager;
        ds.contractManager = _newManager;
        result=true;
        emit ManagerTransferred(previousManager, _newManager);
    }
    function getContractManager() external view returns (address contractManager_) {
        contractManager_ = LibDiamond.diamondStorage().contractManager;
    }
    function setEmergentPause(bool isPause,bool isPauseAsset) external{
        LibDiamond.enforceIsContractOwner();
```

Figure 7 Part source code of contract DOTCManageFacet

- **Fix recommendations:** Declare all variables required by the project in the DOTCFacetBase contract to determine their storage location.
- **Status: Fixed.**

[DOTC-5 High] The second appeal will record the number of arbitrations repeatedly, which may cause some functions to be unusable

- **Description:** As shown in the figure below, during the second appeal, the arbitration count of the corresponding account will be updated again (it will increase during the first appeal), but even if the arbitration is completed, the arbitration count of the corresponding account will only decrease by 1. The count of arbitration orders recorded by the account that caused the second appeal after the arbitration is not 0 (the totalOrderArbitCount also has a similar issue), which will cause the *queryUserInvitor* function to fail.

```
59
60          db.arbitTable.orderArbitList[exOrderId].state=ArbitState.Dealing;
61          db.arbitTable.orderArbitList[exOrderId].lastApplyTime=block.timestamp;
62          db.arbitTable.orderArbitList[exOrderId].arbitResult=ArbitResult.None;
63          db.arbitTable.orderArbitList[exOrderId].currentArbitId = arbitId;
64          db.arbitTable.extend.arbitIdList[arbitId]=exOrderId;
65          db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.lastCompleteTime=0;
66          db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.isSettled=false;
67          db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.orderArbitTimes++;
68          db.arbitTable.totalOrderArbitCount++;
69          db.userTable.userList[db.arbitTable.orderArbitList[exOrderId].applyUser].arbitExOrderCount++;
70          db.userTable.userList[db.arbitTable.orderArbitList[exOrderId].appelle].arbitExOrderCount++;
71          //update arbit period
72          db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.period=_calculateArbitPeriod(exOrderId);
73          period=uint(db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.period);
74      }
```

Figure 8 Part source code of function *createOrderArbit*

```
32  function queryUserInvitor() external view returns(address invitor){
33    invitor=db.userTable.userInviteList[msg.sender];
34  }
35  function updateSponsorAmount(address userAddr,uint amount,uint8 v, bytes32 r,bytes32 s) external returns (bool result) {
36    //check balance
37    {
38      require(userAddr!=address(0),'user address invalid.');
39      // require(amount >= consts.priceParam.nDOTCDecimals,'amount invalid.');
40      require(db.userTable.userInviteList[userAddr]==address(0) || db.userTable.userInviteList[userAddr]==msg.sender,'user has been invited');
41      require(db.userTable.userList[userAddr].arbitExOrderCount<=0,'user has an unclosed arbit');
42    }
43    {
44      if(db.userTable.userInviteList[userAddr]==address(0)){
45        string memory originData='InviterAddress:';
46        originData=LibStrings.strConcat(originData,LibStrings.addressToString(msg.sender));
47        Sign memory sig=Sign(v,r,s);
48        require( SignHelper.verifyString(originData,sig)==userAddr,'signature invalid');
```

Figure 9 Source code of function *queryUserInvitor*

- **Fix recommendations:** The second appeal does not increase the number of corresponding arbitrations.
- **Status: Fixed.** As shown in the figure below, only the first appeal will increase the arbitration count.

```
if(db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.orderArbitTimes==0){
  //first time apply
  db.arbitTable.orderArbitList[exOrderId].applyUser=msg.sender;
  if(db.orderTable.otcTradeOrders[adOrderId][exOrderId].takerAddress==msg.sender){
   db.arbitTable.orderArbitList[exOrderId].appelle=db.orderTable.otcTradeOrders[adOrderId][exOrderId].makerAddress;
  }else{
    db.arbitTable.orderArbitList[exOrderId].appelle=db.orderTable.otcTradeOrders[adOrderId][exOrderId].takerAddress;
  }

  // first create arbit count
  db.arbitTable.totalOrderArbitCount++;
  db.userTable.userList[db.arbitTable.orderArbitList[exOrderId].applyUser].arbitExOrderCount++;
  db.userTable.userList[db.arbitTable.orderArbitList[exOrderId].appelle].arbitExOrderCount++;
}

db.arbitTable.orderArbitList[exOrderId].state=ArbitState.Dealing;
db.arbitTable.orderArbitList[exOrderId].lastApplyTime=block.timestamp;
db.arbitTable.orderArbitList[exOrderId].arbitResult=ArbitResult.None;
db.arbitTable.orderArbitList[exOrderId].currentArbitId = arbitId;
db.arbitTable.extend.arbitIdList[arbitId]=exOrderId;
db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.lastCompleteTime=0;
db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.isSettled=false;
db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.orderArbitTimes++;
// db.arbitTable.totalOrderArbitCount++;
// db.userTable.userList[db.arbitTable.orderArbitList[exOrderId].applyUser].arbitExOrderCount++;
// db.userTable.userList[db.arbitTable.orderArbitList[exOrderId].appelle].arbitExOrderCount++;
//update arbit period
db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.period=_calculateArbitPeriod(exOrderId);
period=uint(db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.period);
}
```

Figure 10 Part source code of function *createOrderArbit*(fixed)

[DOTC-6 Medium] In the *backLendAssets* function, the data returned by the event is wrong

- **Description:** As shown in the figure below, in the *backLendAssets* function of the DOTCLendFacet contract, the *_backToken* function will be called to give back the borrowed assets. Therefore, obtaining the lend amount will return 0, causing the event to return the wrong lendAmount.

```
35      }
36      function backLendAssets() external returns(uint interest,address lendToken,uint actualBack){
37          {
38              require(!db.lendTable.isForbidden,'lend banned');
39          }
40          (interest,lendToken,actualBack) = _backToken(msg.sender);
41          uint lendAmount=db.lendTable.userLend[msg.sender].lend.lendAmount;
42
43          emit _userLendBacked(msg.sender,lendToken,lendAmount,interest,actualBack);
44      }
```

Figure 11 Source code of function *backLendAssets*

- **Fix recommendations:** Get the lendAmount before backing the borrowed assets.
- **Status: Fixed.**

```
function backLendAssets() external returns(uint interest,address lendToken,uint actualBack){
    {
        require(!db.lendTable.isForbidden,'lend banned');
    }
    uint lendAmount=db.lendTable.userLend[msg.sender].lend.lendAmount;
    (interest,lendToken,actualBack) = _backToken(msg.sender);

    emit _userLendBacked(msg.sender,lendToken,lendAmount,interest,actualBack);
}
```

Figure 12 Source code of function *backLendAssets*(fixed)

[DOTC-7 Medium] The detail.lockedAmount of the corresponding Ad order was not updated correctly when the Ex order was completed

- **Description:** The detail.lockedAmount variable of the Ad order is the sum of the locked quantity of the corresponding uncompleted Ex orders, but according to the current code logic, even when the corresponding Ex order is successfully completed, it will not be reduced accordingly, which will cause the corresponding Ad order to not be closed normally.
- **Fix recommendations:** When the Ex order is completed, the lockedAmount quantity of the corresponding Ad order will be updated synchronously.
- **Status: Fixed**

```
function _clearExOrderAssets(string memory adOrderId,string memory exOrderId,address buyerAddr,address sellerAddr,FeeInfo memory buyFee,FeeInfo memory sellFee) internal{
  uint deposit=db.orderTable.otcTradeOrders[adOrderId][exOrderId].depositInfo.deposit;
  uint tradeAmount=db.orderTable.otcTradeOrders[adOrderId][exOrderId].detail.tradeAmount;
  address tokenA=db.orderTable.otcTradeOrders[adOrderId][exOrderId].detail.tokenA;
  uint nHistoryTradeTimes=db.orderTable.otcTradeStatistics[db.orderTable.otcTradeOrders[adOrderId][exOrderId].takerAddress][db.orderTable.otcTradeOrders[adOrderId][exOrderId
  if(tokenA == db.config.usdtContract){

    _unLockToken(buyerAddr,db.config.dotcContract,deposit);
    _unLockToken(sellerAddr,db.config.dotcContract,deposit);

    db.userTable.userAssets[sellerAddr][tokenA].locked=db.userTable.userAssets[sellerAddr][tokenA].locked.sub(tradeAmount);
    db.userTable.userAssets[sellerAddr][db.config.usdtContract].locked=db.userTable.userAssets[sellerAddr][db.config.usdtContract].locked.sub(sellFee.feeValue);

    _addUnlockedAmount(buyerAddr,tokenA,tradeAmount.sub(buyFee.feeValue),nHistoryTradeTimes>0?consts.periodParam.otherTradeLockTime:consts.periodParam.firstTradeLockTime);
    //db.userTable.userAssets[buyerAddr][tokenA].available=db.userTable.userAssets[buyerAddr][tokenA].available.add(tradeAmount.sub(buyFee.feeValue));
    db.orderTable.otcAdOrders[adOrderId].detail.lockedAmount=db.orderTable.otcAdOrders[adOrderId].detail.lockedAmount.sub(tradeAmount);

    _RewardTradeMining(adOrderId,exOrderId,buyerAddr,buyFee);
    _RewardTradeMining(adOrderId,exOrderId,sellerAddr,sellFee);
    //usdt fee to risk pool
    _transferFeeToStakingPool(buyFee);
    _transferFeeToStakingPool(sellFee);
  }else{
```

Figure 13 Part source code of function *_clearExOrderAssets*(fixed)

[DOTC-8 Low] When the balance of the referrer of the losing party is insufficient to pay the penalty, it will not be punished

- **Description:** In the DOTCArbitSettleBase contract, the invitor of the losing party will be punished, but as shown in the figure below, when the invitor's balance of the losing party is insufficient to pay the penalty, they will not be punished.

```
146         db.userTable.userSponsorData[invitor].totalSupply=db.userTable.userSponsorData[invitor].totalSupply.sub(nClearAmount);
147         if(db.userTable.userAssets[invitor][db.config.dotcContract].locked >= nClearAmount){
148             db.userTable.userAssets[invitor][db.config.dotcContract].locked=db.userTable.userAssets[invitor][db.config.dotcContract].locked.sub(nClearAmount);
149             db.arbitTable.orderArbitSettle[exOrderId].invitorSponsor.token=db.config.dotcContract;
150             db.arbitTable.orderArbitSettle[exOrderId].invitorSponsor.userAddr=invitor;
151             db.arbitTable.orderArbitSettle[exOrderId].invitorSponsor.isAddOrSub=false;
152             db.arbitTable.orderArbitSettle[exOrderId].invitorSponsor.amount=nClearAmount;
153 
154             db.arbitTable.extend.arbitGivedToken[db.config.dotcContract]=db.arbitTable.extend.arbitGivedToken[db.config.dotcContract].add(nClearAmount);
155         }
156     }
```

Figure 14 Part source code of function *_clearInvitorSponsor*

- **Fix recommendations:** If the invitor's balance cannot be maintained, all current locked amount of the invitor will be deducted.
- **Status: Fixed.**

```
function _clearInvitorSponsor(string memory exOrderId,address userAddr,uint dotcAmount) internal {
    address invitor=db.userTable.userInviteList[msg.sender];
    if(invitor==address(0)) return;
    uint sponsorAmount=db.userTable.userSponsorData[invitor].sponsorBalances[userAddr];
    uint nClearAmount=sponsorAmount.min(dotcAmount.mul(10).div(100));
    nClearAmount=nClearAmount.min(db.userTable.userAssets[invitor][db.config.dotcContract].locked);
    if(nClearAmount<=0) return;

    db.userTable.userSponsorData[invitor].sponsorBalances[userAddr]= db.userTable.userSponsorData[invitor].sponsorBalances[userAddr].sub(nClearAmount);
    db.userTable.userSponsorData[invitor].totalSupply=db.userTable.userSponsorData[invitor].totalSupply.sub(nClearAmount);
//      if(db.userTable.userAssets[invitor][db.config.dotcContract].locked >= nClearAmount){
    {
        db.userTable.userAssets[invitor][db.config.dotcContract].locked=db.userTable.userAssets[invitor][db.config.dotcContract].locked.sub(nClearAmount);
        db.arbitTable.orderArbitSettle[exOrderId].invitorSponsor.token=db.config.dotcContract;
        db.arbitTable.orderArbitSettle[exOrderId].invitorSponsor.userAddr=invitor;
        db.arbitTable.orderArbitSettle[exOrderId].invitorSponsor.isAddOrSub=false;
        db.arbitTable.orderArbitSettle[exOrderId].invitorSponsor.amount=nClearAmount;

        db.arbitTable.extend.arbitGivedToken[db.config.dotcContract]=db.arbitTable.extend.arbitGivedToken[db.config.dotcContract].add(nClearAmount);
    }
}
function _rewardArbiter(string memory adOrderId,string memory exOrderId,uint winnerCount) internal {
```

Figure 15 Source code of function *_clearInvitorSponsor*(fixed)

[DOTC-9 Low] If the nPeriodCoun in the contract is too large, it will cause calculation errors

- **Description:** As shown in the figure below, in the DOTCFacetBase contract, when calculating the rebate ratio, the index of nPeriodCount is used for calculation. If nPeriodCoun is too large, the calculation here will be wrong. (Although nPeriodCount will not be too large under normal circumstances)

```
    }
    function _getBackRate() internal view returns(uint backRate){
        if(db.daoData.miningPool.poolTokens[db.config.dotcContract].initSupply<=0 || db.daoData.miningPool.poolTokens[db.config.dotcContract].currentSupply<=0) backRate=0;
        uint nPeriodCount=db.daoData.miningPool.poolTokens[db.config.dotcContract].periodCount;
        if(nPeriodCount>10**12) nPeriodCount=10**12;
        backRate=db.daoData.miningPool.poolTokens[db.config.dotcContract].initBackRate.mul(700 ** nPeriodCount).div(1000 ** nPeriodCount);
        if(backRate>1000) backRate=1000;
    }
```

Figure 16 Source code of function *_getBackRate*

- **Fix recommendations:** Limit the size of the nPeriodCount variable.
- **Status: Fixed.**

```
function _getBackRate() internal view returns(uint backRate){
    if(db.daoData.miningPool.poolTokens[db.config.dotcContract].initSupply<=0 || db.daoData.miningPool.poolTokens[db.config.dotcContract].currentSupply<=0) backRate=0;
    uint nPeriodCount=db.daoData.miningPool.poolTokens[db.config.dotcContract].periodCount;
//    if(nPeriodCount>10**12) nPeriodCount=10**12;
    if(nPeriodCount>=10) nPeriodCount=10;
    backRate=db.daoData.miningPool.poolTokens[db.config.dotcContract].initBackRate.mul(700 ** nPeriodCount).div(1000 ** nPeriodCount);
    if(backRate>1000) backRate=1000;
}
```

Figure 17 Source code of function *_getBackRate*(fixed)

[DOTC-10 Low] Avoid empty characters in the order id, which will affect certain judgment logic of the contract

- **Description:** The current project's order id is a variable of type string, and the default is an empty string to indicate an empty order. If the corresponding order id is set to an empty string, it will cause an error in the internal logic of the contract.
- **Fix recommendations:** Restrict the order id created by the user cannot be an empty string.
- **Status: Fixed.**

```
function createExOrder(string calldata adOrderId,string calldata exOrderId,uint amount) external returns (bool result) {
    require(!db.config.isPause,'system paused');
    require(bytes(adOrderId).length > 0,'adOrderId not null');
    require(bytes(exOrderId).length > 0,'exOrderId not null');
    _checkExOrder(adOrderId,exOrderId,amount);
    {
      db.orderTable.otcAdOrders[adOrderId].detail.leftAmount=db.orderTable.otcAdOrders[adOrderId].detail.leftAmount.sub(amount);
      db.orderTable.otcAdOrders[adOrderId].detail.lockedAmount=db.orderTable.otcAdOrders[adOrderId].detail.lockedAmount.add(amount);
    }
    (uint256 nOrderValue,uint256 dotcAmount,uint256 deposit)=_queryAdDeposit(adOrderId,amount);
    require(nOrderValue >= consts.priceParam.minOrderValue,'ExOrder value too little');
```

Figure 18 Part source code of function *createExOrder*

[DOTC-11 Low] In the *_backToken* function of the DOTCLendBase contract, it is required that the number of tokens available to the user must be greater than the number of tokens that should be returned

- **Description:** As shown in the figure below, in the *_backToken* function of the DOTCLendBase contract, the user's available tokens are required to be greater than the number of tokens that should be returned, which should be greater than or equal to here.

```
ftrace | funcSig
function _backToken(address userAddr↑) internal returns(uint interest↑,address lendToken↑,uint bac
    LendResult storage userLend=db.lendTable.userLend[userAddr↑];

    {
        require(userLend.state==2,'state not lending');
        require(userLend.lend.lendAmount>0,'lendAmount zero');
       // require(block.timestamp<userLend.pledge.pledgeTime.add(userLend.pledge.pledgePeriod),'pe
    }
    lendToken↑=userLend.lend.lendToken;
    interest↑=_getInterest(userLend.lend.lendToken,userLend.lend.lendAmount,userLend.lend.lendTime
    interest↑=interest↑.min(userLend.lend.lendAmount);
    backAmount↑=interest↑.add(userLend.lend.lendAmount);

    AssetInfo storage userAssets = db.userTable.userAssets[userAddr↑][lendToken↑];
    require(userAssets.available > backAmount↑, 'Insufficient available');
    {
        //start backToken
        //update state
        userLend.state=1;
        //update assets
```

Figure 19 Part source code of function *_backToken*

- **Fix recommendations:** Change ">" here to ">=".
- **Status: Fixed.**

```
function _backToken(address userAddr) internal returns(uint interest,address lendToken,uint backAmount){
    LendResult storage userLend=db.lendTable.userLend[userAddr];

    {
        require(userLend.state==2,'state not lending');
        require(userLend.lend.lendAmount>0,'lendAmount zero');
       // require(block.timestamp<userLend.pledge.pledgeTime.add(userLend.pledge.pledgePeriod),'period exceed');
    }
    lendToken=userLend.lend.lendToken;
    interest= getInterest(userLend.lend.lendToken,userLend.lend.lendAmount,userLend.lend.lendTime,userLend.lend.lendRate);
    interest=interest.min(userLend.lend.lendAmount);
    backAmount=interest.add(userLend.lend.lendAmount);

    AssetInfo storage userAssets = db.userTable.userAssets[userAddr][lendToken];
    require(userAssets.available >= backAmount, 'Insufficient available');
    {
        //start backToken
        //update state
        userLend.state=1;
        //update assets
        userAssets.available=userAssets.available.sub(backAmount);
        //update total
        db.lendTable.poolTokens[lendToken].totalLend=db.lendTable.poolTokens[lendToken].totalLend.sub(userLend.lend.lendAmount);
        if(db.lendTable.poolTokens[lendToken].lendAccount>0)
        {
            db.lendTable.poolTokens[lendToken].lendAccount--;
```

Figure 20 Part source code of function *_backToken*(fixed)

[DOTC-12 Low] When the user withdraws the pledged assets, the corresponding pledge data will not be cleared, which will affect the query data

- **Description:** When the user retrieves the pledged assets, the corresponding pledge data will not be cleared, which will affect the query data.

```
    }
    function _unlockPledgeAmount(address userAddr) internal returns(address pledgeToken,uint pledgeAmount,uint bonusAmount){
        //check unlock period
        LendResult storage userLend=db.lendTable.userLend[userAddr];
        {
            require(userLend.state==1,'state error');
            require(userLend.pledge.pledgeAmount>0,'lendAmount zero');
            //check period
            require(block.timestamp>=userLend.pledge.pledgeTime.add(userLend.pledge.pledgePeriod),'period limit');
        }
        {
            //start unlock
            pledgeToken=userLend.pledge.pledgeToken;
            pledgeAmount=userLend.pledge.pledgeAmount;
            uint poolBalance =db.lendTable.poolTokens[pledgeToken].totalPledge.sub(db.lendTable.poolTokens[pledgeToken].totalLend);
            require(poolBalance >= pledgeAmount,'Insufficient available balance in the loan pool');

          /*  if(poolBalance >=pledgeAmount){*/
              //normal
            db.userTable.userAssets[userAddr][pledgeToken].available=db.userTable.userAssets[userAddr][pledgeToken].available.add(pledgeAm
            db.lendTable.poolTokens[pledgeToken].totalPledge= db.lendTable.poolTokens[pledgeToken].totalPledge.sub(pledgeAmount);
           /* }else{
                //not enough
                //back from risk pool
                uint riskBalance=db.daoData.riskPool.poolTokens[pledgeToken].currentSupply;
                uint backAmount=pledgeAmount.min(riskBalance);
                db.userTable.userAssets[userAddr][pledgeToken].available=db.userTable.userAssets[userAddr][pledgeToken].available.add(back
```

Figure 21 Source code of function *_unlockPledgeAmount*

- **Fix recommendations:** When withdrawing the pledged assets, clear the corresponding data.
- **Status: Acknowledged.**

[DOTC-13 Low] The *transfer* function failed due to the irregular implementation of USDT on the corresponding platform

- **Description:** As shown in the figure below, when the contract transfers tokens, it will call the *transfer* function of the corresponding token contract and check its return value. If it is a TRON platform, its USDT implementation is not standardized, and the *transfer* function always returns false, which will cause the project contract call to fail.

```
30
31      function transfer(
32          address _token,
33          address _to,
34          uint256 _value
35      ) internal {
36          uint256 size;
37          assembly {
38              size := extcodesize(_token)
39          }
40          require(size > 0, "LibERC20: Address has no code");
41          (bool success, bytes memory result) = _token.call(abi.encodeWithSelector(IERC20.transfer.selector, _to, _value));
42          handleReturn(success, result);
43      }
```

Figure 22 Source code of function *transfer*

- **Fix recommendations:** Special handling for USDT transfer.
- **Status: Fixed.** Added DOTCTronUserFacet contract to handle USDT transfers on TRON.

Figure 23 Modified USDT transfer processing

[DOTC-14 Low] The remaining deposit counted by the *_checkAdOrderLeftDeposit* function is inaccurate

- **Description:** When the *_checkAdOrderLeftDeposit* function performs deposit check, it is required that the sum of this deposit and the deposit of the completed order cannot exceed the deposit of the corresponding Ad order, but when the corresponding Ex order is completed, the corresponding data will be cleared, so this function only counts placed an uncompleted Ex order.

```
51      }
52      function _checkAdOrderLeftDeposit(string memory adOrderId,uint newDeposit) internal view{
53          uint nTotalUsed=0;
54          for(uint i=0;i<db.orderTable.otcAdOrderCounter[adOrderId].length;i++){
55              string memory exOrderId=db.orderTable.otcAdOrderCounter[adOrderId][i];
56              nTotalUsed=nTotalUsed.add(db.orderTable.otcTradeOrders[adOrderId][exOrderId].depositInfo.deposit);
57          }
58          nTotalUsed=nTotalUsed.add(newDeposit);
59          require(nTotalUsed<=db.orderTable.otcAdOrders[adOrderId].depositInfo.deposit,'adOrder left deposit not enough');
60      }
```

Figure 24 Source code of function *_checkAdOrderLeftDeposit*

- **Fix recommendations:** Determine whether there are enough transaction tokens by the leftAmount corresponding to the Ad order.
- **Status: Acknowledged.**

[DOTC-15 Low] "else if" in *queryUserLendRate* function should be "if"

- **Description:** As shown in the figure below, relevant data should be returned in each state.

```
120     }
121     function queryUserLendRate(address userAddr) external view returns(uint interest,address intToken,uint pledgeRate,uint bonusDOTC){
122         //check unlock period
123         LendResult storage userLend=db.lendTable.userLend[userAddr];
124         if(userLend.state==2){
125             //lending
126             intToken=userLend.lend.lendToken;
127             interest = _getInterest(intToken,userLend.lend.lendAmount,userLend.lend.lendTime,userLend.lend.lendRate);
128         }else if(userLend.state>=1){
129             //pledged
130             pledgeRate=10000;
131             //bonus
132             bonusDOTC = _getPledgeBonus(userLend);
133         }
134     }
```

Figure 25 Source code of function *queryUserLendRate*

- **Fix recommendations:** Determine the business requirements, if not, modify the code.
- **Status: Fixed.** This function has been deprecated.

[DOTC-16 Info] When creating an arbitration in the DOTCArbitFacet contract, the id entered by the user is used as the index record, and there may be data coverage problems.

- **Description:** As shown in the figure below, when creating an arbitration in the DOTCArbitFacet contract, the id entered by the user is used as the index record, and there may be a data coverage problem (only affects the query).

```
59
60          db.arbitTable.orderArbitList[exOrderId].state=ArbitState.Dealing;
61          db.arbitTable.orderArbitList[exOrderId].lastApplyTime=block.timestamp;
62          db.arbitTable.orderArbitList[exOrderId].arbitResult=ArbitResult.None;
63          db.arbitTable.orderArbitList[exOrderId].currentArbitId = arbitId;
64          db.arbitTable.extend.arbitIdList[arbitId]=exOrderId;
65          db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.lastCompleteTime=0;
66          db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.isSettled=false;
67          db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.orderArbitTimes++;
68          db.arbitTable.totalOrderArbitCount++;
69          db.userTable.userList[db.arbitTable.orderArbitList[exOrderId].applyUser].arbitExOrderCount++;
70          db.userTable.userList[db.arbitTable.orderArbitList[exOrderId].appelle].arbitExOrderCount++;
71          //update arbit period
```

Figure 26 Source code of function *createOrderArbit*

- **Fix recommendations:** Use unique variable as order index.
- **Status: Acknowledged.**

[DOTC-17 Info] In the DOTCArbitFacet contract, when checking the input parameter for creating an arbitration, the orderArbitTimes should be less than 2

- **Description:** As shown in the figure below, when the DOTCArbitFacet contract creates an arbitration, the number of appeals orderArbitTimes required to correspond to the arbitration must be less than 3, but in fact, according to the requirements, the number of appeals orderArbitTimes should be less than 2.

```
177     }
178     function _checkExArbitApply(string calldata adOrderId,string calldata exOrderId,address userAddr) internal view{
179       require(_getArbiterLength()>=consts.arbitParam.nArbitNum,'arbiter count is less than minimum');
180       require(db.orderTable.otcAdOrders[adOrderId].makerAddress !=address(0),'AdOrder not exists');
181       require(db.orderTable.otcTradeOrders[adOrderId][exOrderId].makerAddress !=address(0),'Trade Order not exists');
182       require(db.orderTable.otcTradeOrders[adOrderId][exOrderId].makerAddress==userAddr || db.orderTable.otcTradeOrders[adOrderId][exOrd
183       //require(db.arbitTable.orderArbitList[exOrderId].state!=ArbitState.Dealing,'you have an uncompleted arbit now.');
184       require(db.orderTable.otcAdOrders[adOrderId].state==OrderState.ONTRADE,'the ad order has been closed.');
185       require(db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.orderArbitTimes<3,'The maximum number of applications has been reach
186       require(db.orderTable.otcTradeOrders[adOrderId][exOrderId].detail.state==TradeState.Filled || db.orderTable.otcTradeOrders[adOrder
187     }
```

Figure 27 Source code of function *_checkExArbitApply*

- **Fix recommendations:** Modify the orderArbitTimes here to be less than 2.
- **Status: Fixed.**

```
    }
    function _checkExArbitApply(string calldata adOrderId,string calldata exOrderId,address userAddr) internal view{
      require(_getArbiterLength()>=consts.arbitParam.nArbitNum,'arbiter count is less than minimum');
      require(db.orderTable.otcAdOrders[adOrderId].makerAddress !=address(0),'AdOrder not exists');
      require(db.orderTable.otcTradeOrders[adOrderId][exOrderId].makerAddress !=address(0),'Trade Order not exists');
      require(db.orderTable.otcTradeOrders[adOrderId][exOrderId].makerAddress==userAddr || db.orderTable.otcTradeOrders[adOrderId][exOrderId].takerAddress==userAddr ,'no access');
      //require(db.arbitTable.orderArbitList[exOrderId].state!=ArbitState.Dealing,'you have an uncompleted arbit now.');
      require(db.orderTable.otcAdOrders[adOrderId].state==OrderState.ONTRADE,'the ad order has been closed.');
      require(db.arbitTable.orderArbitList[exOrderId].arbitBackInfo.orderArbitTimes<2,'The maximum number of applications has been reached');
      require(db.orderTable.otcTradeOrders[adOrderId][exOrderId].detail.state==TradeState.Filled || db.orderTable.otcTradeOrders[adOrderId][exOrderId].detail.state==TradeState.Mo
    }
```

Figure 28 Source code of function *_checkExArbitApply*(fixed)

[DOTC-18 Info] *_checkAdOrder* function comment error in DOTCAdOrderFacet contract

- **Description:** In the DOTCAdOrderFacet contract, when checking the input parameters for creating an Ad order, the maxAmount is judged, and the requirement cannot be greater than the total amount of the order totalAmount, but the following error message expresses the opposite meaning.

```
50      }
51      function _checkAdOrder(AdInput memory  adInput) internal view {
52        require(msg.sender!=address(0),"sender invalid");
53        require(db.userTable.userList[msg.sender].isVIP,"only vip user can create adorder.");
54        require(db.orderTable.userOrderDb[msg.sender].noneAdOrder<consts.orderLimit.vipAdorder,'AdOrder Limit');
55        require(adInput.tokenA!=address(0),"tokenA address invalid");
56        require(adInput.tokenB!=address(0),"tokenB address invalid");
57        require(adInput.tokenB==db.config.usdtContract,"tokenB can only be USDT");
58
59        if (adInput.side == 1 && adInput.tokenA == db.config.dotcContract) {
60          (bool isLock,address lendToken) = __checkLendLock(msg.sender);
61          require(lendToken != db.config.dotcContract,"Lending DOTC can not be sold");
62        }
63
64        require(adInput.price> 0,'price must be greater than 0');
65        require(adInput.totalAmount>0,"amount invalid");
66        require(adInput.minAmount>0,"minAmount invalid");
67        require(adInput.maxAmount>0,"maxAmount invalid");
68        require(adInput.totalAmount>=adInput.maxAmount,"totalAmount greater than maxAmount");
69        require(adInput.minAmount<=adInput.maxAmount, "maxAmount less than minAmount");
70        require(db.orderTable.otcAdOrders[adInput.orderId].makerAddress ==address(0),'AdOrder exists');
71
72      }
73      function _queryAdDeposit(AdInput memory  adInput) internal view returns(uint nOrderValue,uint dotcAmount,uint deposit){
74          uint tokenADecimals= 10 ** LibERC20.queryDecimals(adInput.tokenA);
75          if(adInput.tokenA==db.config.usdtContract){
```

Figure 29 Source code of function *_checkAdOrder*

- **Fix recommendations:** Modify the error message.
- **Status: Fixed.**

```
function _checkAdOrder(AdInput memory  adInput) internal view {
  require(msg.sender!=address(0),"sender invalid");
  require(db.userTable.userList[msg.sender].isVIP,"only vip user can create adorder.");
  require(db.orderTable.userOrderDb[msg.sender].noneAdOrder<consts.orderLimit.vipAdorder,'AdOrder Limit');
  require(adInput.tokenA!=address(0),"tokenA address invalid");
  require(adInput.tokenB!=address(0),"tokenB address invalid");
  require(adInput.tokenB==db.config.usdtContract,"tokenB can only be USDT");

  if (adInput.side == 1 && adInput.tokenA == db.config.dotcContract) {
    (bool isLock,address lendToken) =  _checkLendLock(msg.sender);
      if (lendToken == db.config.dotcContract) {
          require(!isLock,"Lending DOTC can not be sold");
      }
   // require(lendToken != db.config.dotcContract,"Lending DOTC can not be sold");
  }

  require(adInput.price> 0,'price must be greater than 0');
  require(adInput.totalAmount>0,"amount invalid");
  require(adInput.minAmount>0,"minAmount invalid");
  require(adInput.maxAmount>0,"maxAmount invalid");
  require(adInput.totalAmount>=adInput.maxAmount,"maxAmount greater than totalAmount");
  require(adInput.minAmount<=adInput.maxAmount,"maxAmount less than minAmount");
  require(db.orderTable.otcAdOrders[adInput.orderId].makerAddress ==address(0),'AdOrder exists');
```

Figure 30 Source code of function *_checkAdOrder*(fixed)

[DOTC-19 Info] There is some redundant code in the contract

- **Description:** There is some redundant code in the contract. As shown in the figure below, the caller here is not a VIP, so the "if" statement is redundant.

```
99    function applyVIP() external returns (bool result) {
100     UserInfo storage info=db.userTable.userList[msg.sender];
101     require(info.isVIP==false,'user has been vip');
102     require(db.stakingTable.poolA[db.config.dotcContract].accountStakings[msg.sender].balance.add(db.stakir
103
104     if(!info.isVIP){
105         //update vip state
106         db.userTable.userList[msg.sender].isVIP=true;
107         result=true;
108     }
109     emit _VIPApplied(msg.sender,result);
110   }
```

Figure 31 Source code of function *applyVIP*

In addition, the *diamondCut* function code in DiamondCutFacet is exactly the same as that in LibDiamond. Considering the gas consumption of the contract, the code can be optimized.

```
function diamondCut(IDiamondCut.FacetCut[] memory _diamondCut,address _init,bytes memory _calldata) internal {
    DiamondStorage storage ds = diamondStorage();
    uint256 originalSelectorCount = ds.selectorCount;
    uint256 selectorCount = originalSelectorCount;
    bytes32 selectorSlot;
    // Check if last selector slot is not full
    if (selectorCount % 8 > 0) {
        // get last selectorSlot
        selectorSlot = ds.selectorSlots[selectorCount / 8];
    }
    // loop through diamond cut
    for (uint256 facetIndex; facetIndex < _diamondCut.length; facetIndex++) {
        FacetInfo memory facetInfo=FacetInfo(
          selectorCount,
          selectorSlot,
          _diamondCut[facetIndex].facetAddress,
          _diamondCut[facetIndex].action,
          _diamondCut[facetIndex].functionSelectors
        );
        (selectorCount, selectorSlot) = addReplaceRemoveFacetSelectors(facetInfo);
    }
    if (selectorCount != originalSelectorCount) {
        ds.selectorCount = uint16(selectorCount);
    }
    // If last selector slot is not full
    if (selectorCount % 8 > 0) {
        ds.selectorSlots[selectorCount / 8] = selectorSlot;
    }
```

```
contract DiamondCutFacet is IDiamondCut {
    // Standard diamondCut external function
    /// @notice Add/replace/remove any number of functions and optionally exec
    ///         a function with delegatecall
    /// @param _diamondCut Contains the facet addresses and function selectors
    /// @param _init The address of the contract or facet to execute _calldata
    /// @param _calldata A function call, including function selector and argu
    ///                  _calldata is executed with delegatecall on _init
    function diamondCut(
        FacetCut[] calldata _diamondCut,
        address _init,
        bytes calldata _calldata
    ) external override {
        LibDiamond.enforceIsContractOwner();
        LibDiamond.DiamondStorage storage ds = LibDiamond.diamondStorage();
        uint256 originalSelectorCount = ds.selectorCount;
        uint256 selectorCount = originalSelectorCount;
        bytes32 selectorSlot;
        // Check if last selector slot is not full
        if (selectorCount % 8 > 0) {
            // get last selectorSlot
            selectorSlot = ds.selectorSlots[selectorCount / 8];
        }
        // loop through diamond cut
        for (uint256 facetIndex; facetIndex < _diamondCut.length; facetIndex+
            // selectorCount,
            // selectorSlot,
            // _diamondCut[facetIndex].facetAddress,
            // _diamondCut[facetIndex].action,
            // _diamondCut[facetIndex].functionSelectors
```

Figure 32 Source code of function *diamondCut*

- **Fix recommendations:** Optimize redundant code.
- **Status: Partially Fixed.** Redundant code for function *applyVIP* has been removed.

```
function applyVIP() external returns (bool result) {
  UserInfo storage info=db.userTable.userList[msg.sender];
  require(info.isVIP == false,'user has been vip');
  require(db.stakingTable.poolA[db.config.dotcContract].accountStakings[msg.sender].balance.add(db.stakingTable.

    if(!info.isVIP)
  {
      //update vip state
      db.userTable.userList[msg.sender].isVIP=true;
      result=true;
  }
  emit _VIPApplied(msg.sender,result);
}
```

Figure 33 Source code of function *applyVIP*(fixed)

[DOTC-20 Info] When the arbitration data is deleted in the *_removeArbiterFromDB* function, only delete is used, and the corresponding array length is not reduced

- **Description:** As shown in the figure below, when deleting the arbitration data in the *_removeArbiterFromDB* function, the "delete" is used, but the corresponding array length is not reduced.

```
ftrace | funcSig
249  function _removeArbiterFromDB(address arbiter1) internal{
250      uint aValue=uint(uint160(arbiter1));
251      uint index=DOTCLib._findArrayIndexByValue(db.arbitTable.arbiterList,aValue);
252      if(index>0){
253          index--;
254          if(db.arbitTable.arbiterList.length==1){
255              delete db.arbitTable.arbiterList;
256          }else if(index==db.arbitTable.arbiterList.length-1){
257              delete db.arbitTable.arbiterList[db.arbitTable.arbiterList.length-1];
258          }
259          else{
260              db.arbitTable.arbiterList[index]=db.arbitTable.arbiterList[db.arbitTable.arbiterList.length-1];
261
262              delete db.arbitTable.arbiterList[db.arbitTable.arbiterList.length-1];
263          }
264
```

Figure 34 Source code of function *_removeArbiterFromDB*

- **Fix recommendations:** Use "pop" to delete data in an array.
- **Status: Fixed**

```
/******arbit method */
function _removeArbiterFromDB(address arbiter) internal{
    uint aValue=uint(uint160(arbiter));
    uint index=DOTCLib._findArrayIndexByValue(db.arbitTable.arbiterList,aValue);
    if(index>0){
        index--;
        if(db.arbitTable.arbiterList.length==1 || index==db.arbitTable.arbiterList.length-1){ //
            db.arbitTable.arbiterList.pop();
        } else{
            db.arbitTable.arbiterList[index]=db.arbitTable.arbiterList[db.arbitTable.arbiterList.length-1];
            db.arbitTable.arbiterList.pop();
        }
    }
```

Figure 35 Source code of function *_removeArbiterFromDB*(fixed)

[DOTC-21 Info] The *tokenDeposit* function in the DOTCUserFacet contract declares the payable keyword, but the contract does not need to receive platform token

- **Description:** According to the current business logic, the contract does not need to receive platform currency, but as shown in the figure below, the "payable" keyword is declared in the *tokenDeposit* function in the project.

```
function tokenDeposit(address token,uint amount) external payable returns (bool result) {
  require(!db.config.isPauseAsset,'system paused');
  require(token!=address(0),'token invalid');
  require(amount>0,'amount must be greater than 0');
  db.userTable.userAssets[msg.sender][token].available=db.userTable.userAssets[msg.sender][token].available.add(amount);
  LibERC20.transferFrom(token, msg.sender, address(this), amount);

  emit _tokenDeposited(msg.sender,token,amount);

  return true;
}
```

Figure 36 Source code of function *tokenDeposit*

- **Fix recommendations:** Remove the "payable" keyword.
- **Status: Fixed.**

```
function tokenDeposit(address token,uint amount) external returns (bool result) {
  require(!db.config.isPauseAsset,'system paused');
  require(token!=address(0),'token invalid');
  require(amount>0,'amount must be greater than 0');
  db.userTable.userAssets[msg.sender][token].available=db.userTable.userAssets[msg.sender][token].available.add(amount);
  LibERC20.transferFrom(token, msg.sender, address(this), amount);

  emit _tokenDeposited(msg.sender,token,amount);

  return true;
}
```

Figure 37 Source code of function *tokenDeposit*(fixed)

[DOTC-22 Info] The losing party's assets will send half of the traded assets to risk pools, there may be precision issues

- **Description:** As shown in the figure below, for the party that fails the appeal, its corresponding transaction assets will be divided equally and sent to poolA and poolB respectively. The code uses "/2" directly, which may lead to a small amount of assets left in the contract.

```
118
119     }else if(tokenA==db.config.usdtContract){
120       _updatePoolPeriod();
121       db.stakingTable.poolA[db.config.dotcContract].v2Info.totalUSDTBonus=db.stakingTable.poolA[db.config.dotcContract].v2Info.totalUSDTBonus.add(tradeAmount/2);
122       db.stakingTable.poolA[db.config.dotcContract].totalUSDTBonus=db.stakingTable.poolA[db.config.dotcContract].totalUSDTBonus.add(tradeAmount/2);
123       db.arbitTable.orderArbitSettle[exOrderId].stakingPoolA.token=db.config.usdtContract;
124       db.arbitTable.orderArbitSettle[exOrderId].stakingPoolA.amount=tradeAmount/2;
125
126       db.stakingTable.poolB[db.config.dotcContract].v2Info.totalUSDTBonus=db.stakingTable.poolB[db.config.dotcContract].v2Info.totalUSDTBonus.add(tradeAmount/2);
127       db.stakingTable.poolB[db.config.dotcContract].totalUSDTBonus=db.stakingTable.poolB[db.config.dotcContract].totalUSDTBonus.add(tradeAmount/2);
128       db.arbitTable.orderArbitSettle[exOrderId].stakingPoolB.token=db.config.usdtContract;
129       db.arbitTable.orderArbitSettle[exOrderId].stakingPoolB.amount=tradeAmount/2;
130     }else{
```

Figure 38 Source code of function *_clearLoserDeposit*

- **Fix recommendations:** Send the remaining assets after sending to poolA directly to poolB.
- **Status: Acknowledged.**

[DOTC-23 Info] Compiler warning

- **Description:** The project contract specifies that the compiler version is 0.7.0 or higher.

In addition, when some versions of the contract are compiled, the compiler warning shown in the figure below will appear.

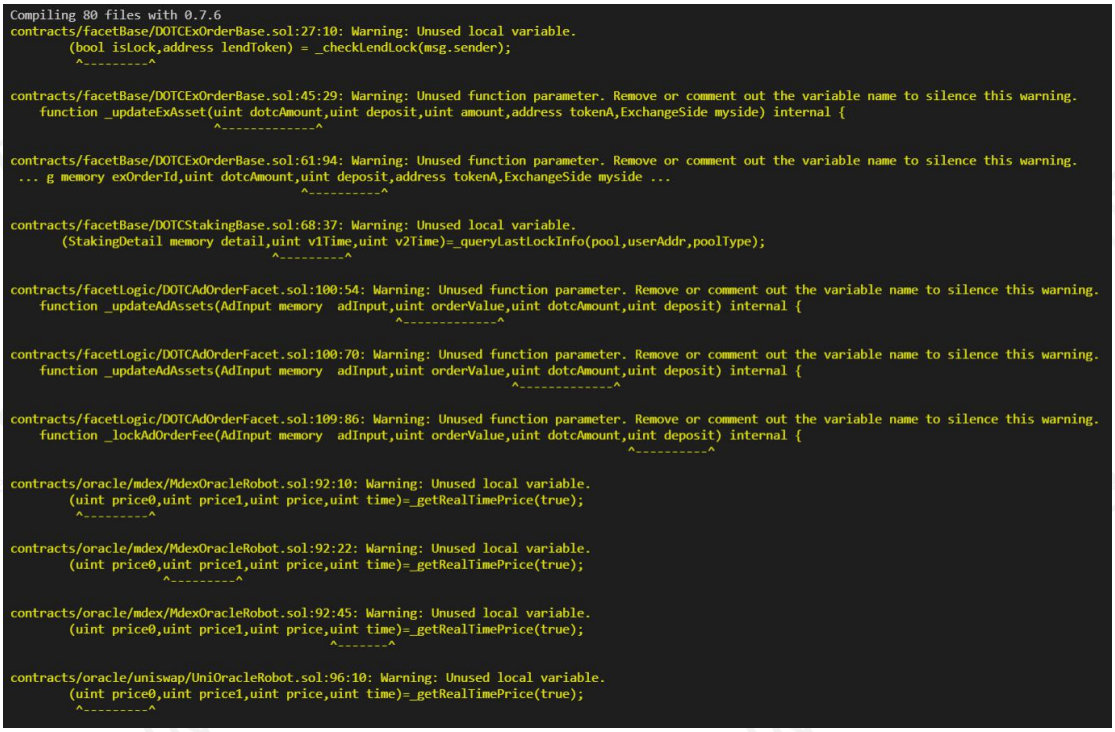

Figure 39 compiler warning

- **Fix recommendations:** It is recommended to use the 0.7.0 version of the compiler to avoid compiler warnings for some contracts in the project.
- **Status: Acknowledged.**

Other audit items explained

1、 Contract mode with separation of storage and logic

The contract of this project adopts the mode of separation of storage and logic, that is, multiple logic contracts share the same storage contract, so there are two issues to pay attention to:

(1) All logical contracts can be updated (modified contract functions) through the corresponding interface, then this audit is only for the relevant contracts of the current version

(2) The existence of special type variables (such as dynamic arrays, mapping) in the contract itself can theoretically expand infinitely, and it is possible to cover the original data. Therefore, it is recommended to limit the number of special data types in the contract.

2、 DOTC tokens cannot be sold or pledged

As shown in the figure below, the contract in the project restricts users from selling the borrowed DOTC tokens. The following three points should be noted:

(1) The project is limited to DOTC, but not to USDT;

(2) It is not limited in quantity, but the state;

(3) Users can use the borrowed DOTC to pay the handling fee.

```
22        }else{
23          require(db.orderTable.userOrderDb[msg.sender].noneExOrder<consts.orderLimit.orderNum,'ExOrder Limit');
24        }
25
26          if (uint(db.orderTable.otcAdOrders[adOrderId].side) == 0
27              && db.orderTable.otcAdOrders[adOrderId].tokenA == db.config.dotcContract) {
28  //            require(false, 'dotc sold');
29              (bool isLock,address lendToken) = _checkLendLock(msg.sender);
30              require(lendToken != db.config.dotcContract, "Lending DOTC can not be sold");
31          }
```

Figure 40 Part source code of function *_checkExOrder*

3、 According to the current code logic, for an Ex order (token transaction), the seller can confirm and complete the transaction before the buyer has paid but not confirmed yet.

4、 In the pledge module of this project, when the user withdraws the pledge, it will be deducted from v1 first, and if the amount of assets in v1 is insufficient, it will be deducted from v2; in addition, after the user withdraws the pledged assets, the corresponding assets also need to be locked unLockWaitTime before it can be unlocked.

5、 In the pledge module of this project, two types of pledges, A and B, can be carried out respectively, of which type A is permanently locked and cannot be withdrawn. In addition, if the pledge data of v1 has not been updated for a long time (the update interval is greater than POOLA_MAX_DAYS), the corresponding pledge data will be cleared.

6、 This project has the highest authority owner and administrator manger, which has the highest authority, which can control the entire project. Although according to the project side's feedback: the project will be managed by the timeLock contract in the next version, but it is still recommended that participants pay attention to the operation of owner and manager.

7、 When arbitration is carried out in the project, the arbitrator will be randomly selected from the arbitration list. The following figure shows the random number generation algorithm used in this project. It can be found that the random number factor can still be predicted, but considering the actual business, the likelihood of an exploit from this point is relatively low.

```
function rand(uint _length,uint nonce) internal view  returns(uint) {
    require(_length!=0,"max num is zero");
    uint random = uint(keccak256(abi.encodePacked(block.difficulty, block.timestamp,msg.sender,nonce)));
    return  random%_length;
}
```

Figure 41 Source code of function *rand*

8、 According to the current code logic, staking users in this cycle can receive rewards from the previous cycle in this cycle, not strictly every 30 days.

9、 In the DOTCTronUserFacet contract in the project, when sponsoring a user, the user's permission is not required.

```
function updateSponsorAmount(address userAddr,uint amount,uint8 v, bytes32 r,bytes32 s) external returns (bool result) {
  //check balance
  {
    require(userAddr!=address(0),'user address invalid.');
    // require(amount >= consts.priceParam.nDOTCDecimals,'amount invalid.');
    require(db.userTable.userInviteList[userAddr]==address(0) || db.userTable.userInviteList[userAddr]==msg.sender,'user has been invited');
    require(db.userTable.userList[userAddr].arbitExOrderCount<=0,'user has an unclosed arbit');
  }
  {
    if(db.userTable.userInviteList[userAddr]==address(0)){
      string memory originData='InviterAddress:';
      originData=LibStrings.strConcat(originData,LibStrings.addressToString(msg.sender));
  //    Sign memory sig=Sign(v,r,s);
   //   require(SignHelper.verifyString(bytes(originData),sig) == userAddr,'signature invalid');
     }
```

Figure 42 Part source code of function *updateSponsorAmount*

Appendix 1 Vulnerability Severity Level and Status Description

- Vulnerability Severity Level

Vulnerability Level	Description	Example
Critical	Vulnerabilities that lead to the complete destruction of the project and cannot be recovered. It is strongly recommended to fix.	Malicious tampering of core contract privileges and theft of contract assets.
High	Vulnerabilities that lead to major abnormalities in the operation of the contract due to contract operation errors. It is strongly recommended to fix.	Unstandardized docking of the USDT interface, causing the user"s assets to be unable to withdraw.
Medium	Vulnerabilities that cause the contract operation result to be inconsistent with the design but will not harm the core business. It is recommended to fix.	The rewards that users received do not match expectations.
Low	Vulnerabilities that have no impact on the operation of the contract, but there are potential security risks, which may affect other functions. The project party needs to confirm and determine whether the fix is needed according to the business scenario as appropriate.	Inaccurate annual interest rate data queries.
Info	There is no impact on the normal operation of the contract, but improvements are still recommended to comply with widely accepted common project specifications.	It is needed to trigger corresponding events after modifying the core configuration.

- Fix Result 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.

Appendix 2 Description of 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
3	Business Security	Business Logics
		Business Implementations

1. Coding Conventions

1.1. Compiler Version Security

The old version of the compiler may cause various known security issues. Developers are advised to specify the contract code to use the latest compiler version and eliminate the compiler alerts.

1.2. Deprecated Items

The Solidity smart contract development language is in rapid iteration. Some keywords have been deprecated by newer versions of the compiler, such as throw, years, etc. To eliminate the potential pitfalls they may cause, contract developers should not use the keywords that have been deprecated by the current compiler version.

1.3. Redundant Code

Redundant code in smart contracts can reduce code readability and may require more gas consumption for contract deployment. It is recommended to eliminate redundant code.

1.4. SafeMath Features

Check whether the functions within the SafeMath library are correctly used in the contract to perform mathematical operations, or perform other overflow prevention checks.

1.5. require/assert Usage

Solidity uses state recovery exceptions to handle errors. This mechanism will undo all changes made to the state in the current call (and all its subcalls) and flag the errors to the caller. The functions assert and require can be used to check conditions and throw exceptions when the conditions are not met. The assert function can only be used to test for internal errors and check non-variables. The require function is used to confirm the validity of conditions, such as whether the input variables or contract state variables meet the conditions, or to verify the return value of external contract calls.

1.6. Gas Consumption

The smart contract virtual machine needs gas to execute the contract code. When the gas is insufficient, the code execution will throw an out of gas exception and cancel all state changes. Contract developers are required to control the gas consumption of the code to avoid function execution failures due to insufficient gas.

1.7. Visibility Specifiers

Check whether the visibility conforms to design requirement.

1.8. Fallback Usage

Check whether the Fallback function has been used correctly in the current contract.

2. General Vulnerability

2.1. Integer overflow

Integer overflow is a security problem in many languages, and they are especially dangerous in smart contracts. Solidity can handle up to 256-bit numbers (2**256-1). If the maximum number is increased by 1, it will overflow to 0. Similarly, when the number is a uint type, 0 minus 1 will underflow to get the maximum number value. Overflow conditions can lead to incorrect results, especially if its possible results are not

expected, which may affect the reliability and safety of the program. For the compiler version after Solidity 0.8.0, smart contracts will perform overflow checking on mathematical operations by default. In the previous compiler versions, developers need to add their own overflow checking code, and SafeMath library is recommended to use.

2.2. Reentrancy

The reentrancy vulnerability is the most typical Ethereum smart contract vulnerability, which has caused the DAO to be attacked. The risk of reentry attack exists when there is an error in the logical order of calling the call.value() function to send assets.

2.3 Pseudo-random Number Generator (PRNG)

Random numbers may be used in smart contracts. In solidity, it is common to use block information as a random factor to generate, but such use is insecure. Block information can be controlled by miners or obtained by attackers during transactions, and such random numbers are to some extent predictable or collidable.

2.4. Transaction-Ordering Dependence

In the process of transaction packing and execution, when faced with transactions of the same difficulty, miners tend to choose the one with higher gas cost to be packed first, so users can specify a higher gas cost to have their transactions packed and executed first.

2.5. DoS(Denial of Service)

DoS, or Denial of Service, can prevent the target from providing normal services. Due to the immutability of smart contracts, this type of attack can make it impossible to ever restore the contract to its normal working state. There are various reasons for the denial of service of a smart contract, including malicious revert when acting as the recipient of a transaction, gas exhaustion caused by code design flaws, etc.

2.6. Function Call Permissions

If smart contracts have high-privilege functions, such as coin minting, self-destruction, change owner, etc., permission restrictions on function calls are required to avoid security problems caused by permission leakage.

2.7. call/delegatecall Security

Solidity provides the call/delegatecall function for function calls, which can cause call injection vulnerability if not used properly. For example, the parameters of the call, if controllable, can control this contract to perform unauthorized operations or call dangerous functions of other contracts.

2.8. Returned Value Security

In Solidity, there are transfer(), send(), call.value() and other methods. The transaction will be rolled back if the transfer fails, while send and call.value will return false if the transfer fails. If the return is not correctly

judged, the unanticipated logic may be executed. In addition, in the implementation of the transfer/transferFrom function of the token contract, it is also necessary to avoid the transfer failure and return false, so as not to create fake recharge loopholes.

2.9. tx.origin Usage

The tx.origin represents the address of the initial creator of the transaction. If tx.origin is used for permission judgment, errors may occur; in addition, if the contract needs to determine whether the caller is the contract address, then tx.origin should be used instead of extcodesize.

2.10. Replay Attack

A replay attack means that if two contracts use the same code implementation, and the identity authentication is in the transmission of parameters, the transaction information can be replayed to the other contract to execute the transaction when the user executes a transaction to one contract.

2.11. Overriding Variables

There are complex variable types in Solidity, such as structures, dynamic arrays, etc. When using a lower version of the compiler, improperly assigning values to it may result in overwriting the values of existing state variables, causing logical exceptions during contract execution.

3. Business Security

3.1 Business Logic

Whether the business logic is designed clearly and flawlessly.

3.2 Business Implementations

Whether the code implementation conforms to comments, project whitepaper, etc.

Appendix 3 Disclaimer

This report is made in response to the project code. No description, expression or wording in this report shall be construed as an endorsement, affirmation or confirmation of the project. 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 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 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 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 Technology assumes no responsibility for the resulting loss or adverse effects. The audit report issued by Beosin Technology is based on the documents and materials provided by the contract provider, and relies on the technology currently possessed by Beosin. Due to the technical limitations of any organization, this report conducted by Beosin still has the possibility that the entire risk cannot be completely detected. Beosin disclaims any liability for the resulting losses.

The final interpretation of this statement belongs to Beosin Technology.

Appendix 4 About Beosin

BEOSIN is a leading global blockchain security company dedicated to the construction of blockchain security ecology, with team members coming from professors, post-docs, PhDs from renowned universities and elites from head Internet enterprises who have been engaged in information security industry for many years. BEOSIN has established in-depth cooperation with more than 100 global blockchain head enterprises; and has provided security audit and defense deployment services for more than 1,000 smart contracts, more than 50 blockchain platforms and landing application systems, and nearly 100 digital financial enterprises worldwide. Relying on technical advantages, BEOSIN has applied for nearly 50 software invention patents and copyrights.

Official Website

https://beosin.com

E-mail

contact@beosin.com

Twitter

https://twitter.com/Beosin_com