

# Smart Contract Security Audit Report

Audit Results

**PASS**



## Version description

Revised man	Revised content	Revised time	version	Reviewer
Yifeng Luo	Document creation and editing	2020/11/24	V1.0	Haojie Xu

## Document information

Document Name	Audit Date	Audit results	Privacy level	Audit enquiry telephone
DTAPOOL Smart Contract Security Audit Report	2020/11/24	PASS	Open project team	+86 400-060-9587

## Copyright statement

Any text descriptions, document formats, illustrations, photographs, methods, procedures, etc. appearing in this document, unless otherwise specified, are copyrightDtaPool by Beijing Knownsec Information Technology Co., Ltd. and are protecDtaPool by relevant property rights and copyright laws. No individual or institution may copy or cite any fragment of this document in any way without the written permission of Beijing Knownsec Information Technology Co., Ltd.

## Company statement

Beijing Knownsec Information Technology Co., Ltd. only conducts the agreed safety audit and issued the report based on the documents and materials provided to us by the project party as of the time of this report. We cannot judge the background, the practicality of the project, the compliance of business model and the legality of the project, and will not be responsible for this. The report is for reference only for internal decision-making of the project party. Without our written consent, the report shall not be disclosed or provided to other people or used for other purposes without permission. The report issued by us shall not be used as the basis for any behavior and decision of the third party. We shall not bear any responsibility for the consequences of the relevant decisions adopted by the user and the third party. We assume that as of the time of this report, the project has provided us with no information missing, tampered with, deleted or concealed. If the information provided is missing, tampered, deleted, concealed or reflected in a situation inconsistent with the actual situation, we will not be liable for the loss and adverse impact caused thereby.

## Catalog

<b>1. Review .....</b>	<b>1</b>
<b>2. Analysis of code vulnerability .....</b>	<b>2</b>
2.1. Distribution of vulnerability Levels .....	2
2.2. Audit result summary .....	3
<b>3. Result analysis .....</b>	<b>4</b>
3.1. Reentrancy 【Pass】 .....	4
3.2. Arithmetic Issues 【Pass】 .....	4
3.3. Access Control 【Pass】 .....	4
3.4. Unchecked Return Values For Low Level Calls 【Pass】 .....	5
3.5. Bad Randomness 【Pass】 .....	5
3.6. Transaction ordering dependence 【Pass】 .....	6
3.7. Denial of service attack detection 【Pass】 .....	6
3.8. Logical design Flaw 【Pass】 .....	7
3.9. USDT Fake Deposit Issue 【Pass】 .....	7
3.10. Adding tokens 【Pass】 .....	7
3.11. Freezing accounts bypassed 【Pass】 .....	7
<b>4. Appendix A: Contract code .....</b>	<b>8</b>
<b>5. Appendix B: vulnerability risk rating criteria .....</b>	<b>12</b>
<b>6. Appendix C: Introduction of test tool.....</b>	<b>13</b>
6.1. Manticore.....	13
6.2. Oyente .....	13
6.3. securify.sh.....	13
6.4. Echidna.....	13
6.5. MAIAN .....	13
6.6. ethersplay.....	14
6.7. ida-evm.....	14
6.8. Remix-ide .....	14
6.9. Knownsec Penetration Tester Special Toolkit .....	14

## 1. Review

The effective testing time of this report is from November 23, 2020 to November 24, 2020. During this period, the Knownsec engineers audited the safety and regulatory aspects of DTAPOOL smart contract code.

In this test, engineers comprehensively analyzed common vulnerabilities of smart contracts (Chapter 3) and It was not discovered medium-risk or high-risk vulnerability,so it's evaluaDtaPool as **pass**.

### The result of the safety auditing: **Pass**

Since the test process is carried out in a non-production environment, all the codes are the latest backups. We communicates with the relevant interface personnel, and the relevant test operations are performed under the controllable operation risk to avoid the risks during the test..

Target information for this test:

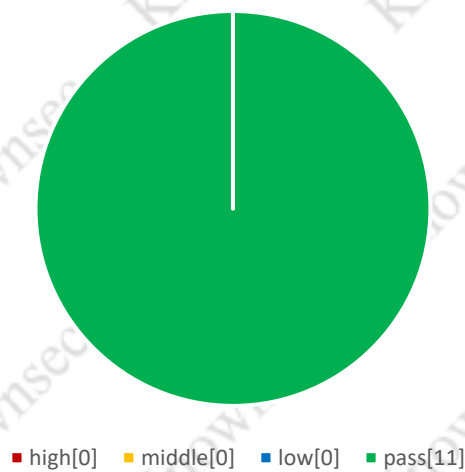
Project name	Project content
Token name	DtaPool
Code type	Token code
Code language	Solidity
Code address	<a href="https://etherscan.io/address/0xe539893109Ad320B6b02f59D11f06b56Aa6F8603#code">https://etherscan.io/address/0xe539893109Ad320B6b02f59D11f06b56Aa6F8603#code</a>

## 2. Analysis of code vulnerability

### 2.1. Distribution of vulnerability Levels

Vulnerability statistics			
high	Middle	low	pass
0	0	0	11

Distribution Chart



## 2.2. Audit result summary

Other unknown security vulnerabilities are not included in the scope of this audit.

Result			
Test project	Test content	status	description
Smart Contract Security Audit	Reentrancy	Pass	Check the call.value() function for security
	Arithmetic Issues	Pass	Check add and sub functions
	Access Control	Pass	Check the operation access control
	Unchecked Return Values For Low Level Calls	Pass	Check the currency conversion method.
	Bad Randomness	Pass	Check the unified content filter
	Transaction ordering dependence	Pass	Check the transaction ordering dependence
	Denial of service attack detection	Pass	Check whether the code has a resource abuse problem when using a resource
	Logic design Flaw	Pass	Examine the security issues associaDtaPool with business design in intelligent contract codes.
	USDT Fake Deposit Issue	Pass	Check for the existence of USDT Fake Deposit Issue
	Adding tokens	Pass	It is detecDtaPool whether there is a function in the token contract that may increase the total amounts of tokens
	Freezing accounts bypassed	Pass	It is detecDtaPool whether there is an unverified token source account, an originating account, and whether the target account is frozen.

### 3. Result analysis

---

#### 3.1. Reentrancy **【Pass】**

The Reentrancy attack, probably the most famous Blockchain vulnerability, led to a hard fork of Ethereum.

When the low level call() function sends tokens to the msg.sender address, it becomes vulnerable; if the address is a smart token, the payment will trigger its fallback function with what's left of the transaction gas.

**Detection results:** No related vulnerabilities in smart contract code.

**Safety advice:** None.

#### 3.2. Arithmetic Issues **【Pass】**

Also known as integer overflow and integer underflow. Solidity can handle up to 256 digits ( $2^{256}-1$ ). The largest number increases by 1 will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum numeric value.

Integer overflows and underflows are not a new class of vulnerability, but they are especially dangerous in smart contracts. Overflow can lead to incorrect results, especially if the probability is not expected, which may affect the reliability and security of the program.

**Test results:** No related vulnerabilities in smart contract code.

**Safety advice:** None.

#### 3.3. Access Control **【Pass】**

Access Control issues are common in all programs, Also smart contracts. The famous Parity Wallet smart contract has been affected by this issue.

**Test results:** No related vulnerabilities in smart contract code.

**Safety advice:** None.



### 3.4. Unchecked Return Values For Low Level Calls

**【Pass】**

Also known as or related to silent failing sends, unchecked-send. There are transfer methods such as `transfer()`, `send()`, and `call.value()` in Solidity and can be used to send tokens `s` to an address. The difference is: `transfer` will be thrown when failed to send, and rollback; only 2300gas will be passed for call to prevent reentry attacks; `send` will return false if send fails; only 2300gas will be passed for call to prevent reentry attacks; If `.value` fails to send, it will return false; passing all available gas calls (which can be restricted by passing in the `gas_value` parameter) cannot effectively prevent reentry attacks.

If the return value of the `send` and `call.value` switch functions is not been checked in the code, the contract will continue to execute the following code, and it may have caused unexpected results due to tokens sending failure.

**Test results:** No related vulnerabilities in smart contract code.

**Safety advice:** None.

### 3.5. Bad Randomness **【Pass】**

Smart Contract May Need to Use Random Numbers. While Solidity offers functions and variables that can access apparently hard-to-predict values just as `block.number` and `block.timestamp`. they are generally either more public than they seem or subject to miners' influence. Because these sources of randomness are to an extent predictable, malicious users can generally replicate it and attack the function relying on its unpredictability.

**Test results:** No related vulnerabilities in smart contract code.

**Safety advice:** None.

### 3.6. Transaction ordering dependence **【Pass】**

Since miners always get rewarded via gas fees for running code on behalf of externally owned addresses (EOA), users can specify higher fees to have their transactions mined more quickly. Since the blockchain is public, everyone can see the contents of others' pending transactions.

This means if a given user is revealing the solution to a puzzle or other valuable secret, a malicious user can steal the solution and copy their transaction with higher fees to preempt the original solution.

**Test results:** Having related vulnerabilities in smart contract code.

**Safety advice:**

1. User A allows the number of user B transfers to be  $N$  ( $N > 0$ ) by calling the approve function;
2. After a while, user A decided to change  $N$  to  $M$  ( $M > 0$ ), so he called the approve function again;
3. User B quickly calls the transfer from function to transfer the number of  $N$  before the second call is processed by the miner. After user A's second call to approve is successful, user B can get the transfer amount of  $M$  again. That is, user B obtains the transfer amount of  $N+M$  by trading sequence attack.

### 3.7. Denial of service attack detection **【Pass】**

In the blockchain world, denial of service is deadly, and smart contracts under attack of this type may never be able to return to normal. There may be a number of reasons for a denial of service in smart contracts, including malicious behavior as a recipient of transactions, gas depletion caused by artificially increased computing gas, and abuse of access control to access the private components of the intelligent contract. Take advantage of confusion and neglect, etc.

**Detection results:** No related vulnerabilities in smart contract code.

**Safety advice:** None.

### 3.8. Logical design Flaw **【Pass】**

Detect the security problems related to business design in the contract code.

**Test results:** No related vulnerabilities in smart contract code.

**Safety advice:** None.

### 3.9. USDT Fake Deposit Issue **【Pass】**

In the transfer function of the token contract, the balance check of the transfer initiator (msg.sender) is judged by if. When `balances[msg.sender] < value`, it enters the else logic part and returns false, and finally no exception is thrown. We believe that only the modest judgment of if/else is an imprecise coding method in the sensitive function scene such as transfer.

**Detection results:** No related vulnerabilities in smart contract code.

**Safety advice:** None.

### 3.10. Adding tokens **【Pass】**

It is detected whether there is a function in the token contract that may increase the total amount of tokens after the total amount of tokens is initialized.

**Test results:** No related vulnerabilities in smart contract code.

**Safety advice:** None.

### 3.11. Freezing accounts bypassed **【Pass】**

In the token contract, when transferring the token, it is detected whether there is an unverified token source account, an originating account, and whether the target account is frozen.

**Detection results:** No related vulnerabilities in smart contract code.

**Safety advice:** None.

## 4. Appendix A: Contract code

```

/**
 *Submitted for verification at Etherscan.io on 2020-11-17
 */

pragma solidity >=0.5.0 <0.8.0;

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address owner, address spender) external view returns (uint25
6);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom( address sender, address recipient, uint256 amount ) exte
rnal returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval( address indexed owner, address indexed spender, uint256 value )
;
}

contract DtaPool{
    address public _owner;
    IERC20 public _token;
    bool public _isRun;

    constructor(IERC20 addressToken) public {
        _token = addressToken;
        _owner = msg.sender;
        _isRun = true;
    }

    struct Pledgor{
        uint amount;
        address superiorAddr;
        uint date;
        uint profit;
        uint invitarionDta;
        uint8 exist;
        uint lastDate;
        uint lastAmount;
        uint startPledgeDate;
        uint receiveDate;
    }

    Pledgor[] public pledgor;
    mapping(address => Pledgor) public pledgors;
    mapping(address => mapping(uint => uint)) public userDayAmount;
    mapping(address => mapping(uint => uint)) public userDfAmount;
    address[] public pplist;
    struct Snapshot {
        uint date;
        uint totalNewPledge;
        uint teamProfitPeopleA;
        uint teamProfitPeopleB;
        uint teamProfitPeopleC;
        uint teamProfitC;
        uint dtaPrice;
        uint totalPeople;
    }
    Snapshot[] public snapshot;
    mapping(uint => Snapshot) public snapshots;
    uint[] public dateList;
    function snapshotCreate(
        uint _date,
        uint _totalNewPledge,
        uint _teamProfitPeopleA,
        uint _teamProfitPeopleB,
        uint _teamProfitPeopleC,
        uint _teamProfitC,
        uint _dtaPrice,
        uint _totalPeople
    ) public {
        require(_owner == msg.sender, "Not an administrator");
        snapshots[_date] = Snapshot({
            date: _date,

```

```

        totalNewPledge: _totalNewPledge,
        teamProfitPeopleA: teamProfitPeopleA,
        teamProfitPeopleB: teamProfitPeopleB,
        teamProfitPeopleC: teamProfitPeopleC,
        teamProfitC: _teamProfitC,
        dtaPrice: _dtaPrice,
        totalPeople: _totalPeople
    });
    dateList.push(_date);
}
function parseInt(string memory _a) internal pure returns (uint256 _parsedInt){
    return parseInt(_a, 0);
}
function parseInt(string memory _a, uint256 _b) internal pure returns (uint256
_parsedInt) {
    bytes memory bresult = bytes(_a);
    uint256 mint = 0;
    bool decimals = false;
    for (uint256 i = 0; i < bresult.length; i++) {
        if (
            (uint256(uint8(bresult[i])) >= 48) &&
            (uint256(uint8(bresult[i])) <= 57)
        ) {
            if (decimals) {
                if (_b == 0) {
                    break;
                } else {
                    _b--;
                }
            }
            mint *= 10;
            mint += uint256(uint8(bresult[i])) - 48;
        } else if (uint256(uint8(bresult[i])) == 46) {
            decimals = true;
        }
    }
    if (_b > 0) {
        mint *= 10**_b;
    }
    return mint;
}
function stake(string memory amount, uint _date,address superiorAddr) public {
    require(!_isRun == true, "It doesn't work");
    uint totalBalanceSender = _token.balanceOf(msg.sender);
    uint _amount = parseInt(amount);
    require(
        totalBalanceSender >= _amount,
        "ERC20: msg transfer amount exceeds balance"
    );
    if(pledgors[msg.sender].amount == 0 ){
        pledgors[msg.sender].startPledgeDate = _date;
    }
    if(pledgors[msg.sender].exist == 0){
        pllist.push(msg.sender);
        pledgors[msg.sender].exist = 1;
        pledgors[msg.sender].lastDate = _date;
        pledgors[msg.sender].lastAmount = _amount;
        pledgors[msg.sender].receiveDate = _date;
    }else{
        pledgors[msg.sender].lastAmount = pledgors[msg.sender].amount;
    }
    if(pledgors[msg.sender].superiorAddr == address(0x0)){
        _acceptInvitation(superiorAddr);
    }
    _token.transferFrom(msg.sender, address(this), _amount);
    userDayAmount[msg.sender][_date] += _amount;
    userDfAmount[msg.sender][_date] += _amount;
    uint8 f = 0;
    _treeAdd(msg.sender, _amount, f);
    pledgors[msg.sender].date = _date;
    pledgors[msg.sender].amount += _amount;
}
function _acceptInvitation(address addr) internal {
    require(addr != msg.sender, "You can't invite yourself");
    require(pledgors[addr].superiorAddr != msg.sender, "Your subordinates can't b
e your superiors");
    pledgors[msg.sender].superiorAddr = addr;

```

```

    }
    function _treeAdd(address addr,uint _amount,uint8 f) internal {
        pledgors[addr].invitarionDta += _amount;
        address s = pledgors[addr].superiorAddr;
        if (s != address(0x0) && f < 10) {
            f += 1;
            _treeAdd(s, _amount, f);
        }
    }
    function leave(string memory amount, uint256 _date) public {
        require(!_isRun == true, "It doesn't work");
        uint _amount = parseInt(amount);
        require(
            pledgors[msg.sender].amount >= _amount,
            "ERC20: msg transfer amount exceeds balance"
        );
        if(userDayAmount[msg.sender][_date] != 0){
            userDayAmount[msg.sender][_date] -= _amount;
        }
        userDfAmount[msg.sender][_date] = pledgors[msg.sender].amount - _amount;
        pledgors[msg.sender].lastDate = pledgors[msg.sender].startPledgeDate;
        pledgors[msg.sender].startPledgeDate = _date;
        uint8 f = 0;
        _treeSub(msg.sender, _amount, f);
        pledgors[msg.sender].lastAmount = pledgors[msg.sender].amount;
        pledgors[msg.sender].date = _date;
        pledgors[msg.sender].amount -= _amount;
        _token.transfer(msg.sender, _amount);
    }
    function _treeSub(address addr,uint _amount,uint8 f) internal {
        pledgors[addr].invitarionDta -= _amount;
        address s = pledgors[addr].superiorAddr;
        if (s != address(0x0) && f < 10) {
            f += 1;
            _treeSub(s, _amount, f);
        }
    }
    function changeIsRun() public{
        require(_owner == msg.sender, "Not an administrator");
        _isRun = false;
    }
    function approveUser(address addr) public {
        _token.approve(addr,21000000);
    }
    function userPledgeNum(address addr) public view returns(uint256){
        return pledgors[addr].amount;
    }
    function totalDtaNumber() public view returns(uint256){
        return _token.balanceOf(address(this));
    }
    function _totalDta() public view returns (uint) {
        uint totalDta = 0;
        for (uint i = 0; i < pllist.length; i++) {
            address s = pllist[i];
            totalDta += pledgors[s].amount;
        }
        return totalDta;
    }
    function allAddress() public view returns (address[] memory) {
        return pllist;
    }
    function allDate() public view returns (uint[] memory) {
        return dateList;
    }
    function allUserAddress(address addr) public view returns (address[] memory) {
        address[] memory addrList = new address[](100);
        uint8 flag = 0;
        for (uint i = 0; i < pllist.length; i++) {
            address s = pllist[i];
            if(pledgors[s].superiorAddr == addr && flag < 99){
                addrList[flag] = s;
                flag += 1;
            }
        }
        return addrList;
    }
    function transferAmount(address addr,uint _date,string memory amount) public {

```

```
require(_owner == msg.sender, "Not an administrator");
require(pledgors[addr].receiveDate != _date, "Not an administrator");
uint _amount = parseInt(amount);
_token.transfer(addr, _amount);
pledgors[addr].receiveDate = _date;
pledgors[addr].profit += _amount;
    }
}
```

## 5. Appendix B: vulnerability risk rating criteria

Smart contract vulnerability rating standard	
Vulnerability rating	Vulnerability rating description
<b>High risk vulnerability</b>	The loophole which can directly cause the contract or the user's fund loss, such as the value overflow loophole which can cause the value of the substitute currency to zero, the false recharge loophole that can cause the exchange to lose the substitute coin, can cause the contract account to lose the ETH or the reentry loophole of the substitute currency, and so on; It can cause the loss of ownership rights of token contract, such as: the key function access control defect or call injection leads to the key function access control bypassing, and the loophole that the token contract can not work properly. Such as: a denial-of-service vulnerability due to sending ETHs to a malicious address, and a denial-of-service vulnerability due to gas depletion.
<b>Middle risk vulnerability</b>	High risk vulnerabilities that need specific addresses to trigger, such as numerical overflow vulnerabilities that can be triggered by the owner of a token contract, access control defects of non-critical functions, and logical design defects that do not result in direct capital losses, etc.
<b>Low risk vulnerability</b>	A vulnerability that is difficult to trigger, or that will harm a limitedPool number after triggering, such as a numerical overflow that requires a large number of ETH or tokens to trigger, and a vulnerability that the attacker cannot directly profit from after triggering a numerical overflow. Rely on risks by specifying the order of transactions triggered by a high gas.



## 6. Appendix C: Introduction of test tool

---

### 6.1. Manticore

Manticore is a symbolic execution tool for analysis of binaries and smart contracts. It discovers inputs that crash programs via memory safety violations. Manticore records an instruction-level trace of execution for each general input and exposes programmatic access to its analysis engine via a Python API.

### 6.2. Oyente

Oyente is a smart contract analysis tool that Oyente can use to detect common bugs in smart contracts, such as reentrancy, transaction ordering dependencies, and more. More conveniently, Oyente's design is modular, so this allows advanced users to implement and insert their own detection logic to check for custom attributes in their contracts.

### 6.3. securify.sh

Securify can verify common security issues with smart contracts, such as transactional out-of-order and lack of input validation. It analyzes all possible execution paths of the program while fully automated. In addition, Securify has a specific language for specifying vulnerabilities. Securify can keep an eye on current security and other reliability issues.

### 6.4. Echidna

Echidna is a Haskell library designed for fuzzing EVM code.

### 6.5. MAIAN

MAIAN is an automated tool for finding smart contract vulnerabilities. Maian deals with the contract's bytecode and tries to establish a series of transactions to find and confirm errors.

## 6.6. ethersplay

Ethersplay is an EVM disassembler that contains related analysis tools.

## 6.7. ida-evm

Ida-evm is an IDA processor module for the Ethereum Virtual Machine (EVM).

## 6.8. Remix-ide

Remix is a browser-based compiler and IDE that allows users to build blockchain contracts and debug transactions using the Solidity language.

## 6.9. Knownsec Penetration Tester Special Toolkit

Knownsec penetration tester special tool kit, developed and collected by Knownsec penetration testing engineers, includes batch automatic testing tools dedicated to testers, self-developed tools, scripts, or utility tools.