KNOWNSEC

Smart Contract Security Audit Report

Audit Results

PASS

★★★★★

Version description

Revised man	Revised content	Revised time	version	Reviewer
Yifeng Luo	Document creation and editing	2020/11/05	V1.0	Haojie Xu

Document information

Document Name	Audit Date	Audit results	Privacy level	Audit enquiry telephone
TRXBANK Smart Contract Security Audit Report	2020/11/05	PASS	Open project team	+86 400-060-9587

Copyright statement

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

1. Review

The effective testing time of this report is from October 28, 2020 to November 5, 2020. During this period, the Knownsec engineers audited the safety and regulatory aspects of TRXBANK 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 evaluated 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:

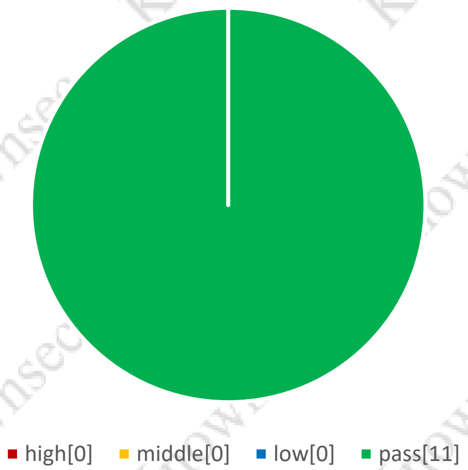
Project name	Project content
Token name	TRXBANK
Code type	Token code
Code language	Solidity
Code address	Code files

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 associated 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 detected whether there is a function in the token contract that may increase the total amounts of tokens
	Freezing accounts bypassed	Pass	It is detected 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.

Test 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 unpredictablility.

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: No related vulnerabilities in smart contract code.

Safety advice: None.

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.

Test 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

```
pragma solidity 0.5.10;

contract TRONBank {
  using SafeMath for uint256;

  uint256 constant public INVEST_MIN_AMOUNT = 100 trx;
  //uint256 constant public INVEST_MIN_AMOUNT = 100;
  uint256 constant public BASE_PERCENT = 10;
  uint256[] public REFERRAL_PERCENTS = [50, 20, 5, 5, 5];
  uint256 constant public MARKETING_FEE = 50;
  uint256 constant public PROJECT_DEV_FEE = 150;
  uint256 constant public PERCENTS_DIVIDER = 1000;
  uint256 constant public CONTRACT_BALANCE_STEP = 1000000 trx;
  //uint256 constant public CONTRACT_BALANCE_STEP = 1000000;
  uint256 constant public TIME_STEP = 10 days;

  uint256 constant internal QUOTA = 1000000 trx;

  uint256[] public VIP_LEVEL = [10000 trx, 50000 trx, 100000 trx, 2000000 trx, 50000
0 trx];
  //uint256[] public VIP_LEVEL = [10000, 50000, 100000, 2000000, 500000];

  uint256[] public VIP_LEVEL_PERCENT = [5, 10, 15, 20, 25];

  uint256 public totalUsers;
  uint256 public totalInvested;
  uint256 public totalWithdrawn;
  uint256 public totalDeposits;

  uint256 public startTime;

  address payable public marketingAddress;
  address payable public projectDevAddress;

  struct Deposit {
    uint256 amount;
    uint256 withdrawn;
    uint256 start;
  }

  struct User {
    Deposit[] deposits;
    uint256 checkpoint;
    address payable referrer;
    uint256 bonus;
    uint256 referCount;
    uint256 referAmout;
    uint256 directReferralReward;
    uint256 teamRevenueReward;
    uint256 teamNumber;
  }

  mapping (address => User) public users;

  mapping (address => uint256) public userWithdraw;

  mapping (address => uint256) public vipWithdraw;

  mapping(uint256 => uint256) public levelSeniority;

  event Newbie(address user);
  event NewDeposit(address indexed user, uint256 amount);
  event Withdrawn(address indexed user, uint256 amount);
  event RefBonus(address indexed referrer, address indexed referral, uint256 indexed
 level, uint256 amount);
  event FeePayed(address indexed user, uint256 totalAmount);

  constructor(address payable marketingAddr, address payable projectDevAddr) public
{
```

```
require(!isContract(marketingAddr) && !isContract(projectDevAddr));
    marketingAddress = marketingAddr;
    projectDevAddress = projectDevAddr;


    levelSeniority[0] = 0;
    levelSeniority[1] = 5;
    levelSeniority[2] = 8;
    levelSeniority[3] = 12;
    levelSeniority[4] = 20;
    levelSeniority[5] = 30;


    startTime = block.timestamp;

  }

  function invest(address payable referrer) public payable {
    require(msg.value >= INVEST_MIN_AMOUNT);

    marketingAddress.transfer(msg.value.mul(MARKETING_FEE).div(PERCENTS_DIVIDER));
    projectDevAddress.transfer(msg.value.mul(PROJECT_DEV_FEE).div(PERCENTS_DIVIDER));
    emit FeePayed(msg.sender, msg.value.mul(MARKETING_FEE.add(PROJECT_DEV_FEE)).div(P
ERCENTS_DIVIDER));

    User storage user = users[msg.sender];

    if (user.referrer == address(0) && users[referrer].deposits.length > 0 && referre
r != msg.sender) {
      user.referrer = referrer;
    }

    if (user.referrer != address(0)) {

      address upline = user.referrer;
      for (uint256 i = 0; i < 5; i++) {
        if (upline != address(0)) {
          uint256 amount = msg.value.mul(REFERRAL_PERCENTS[i]).div(PERCENTS_DIVIDER);
          users[upline].bonus = users[upline].bonus.add(amount);
          users[upline].teamNumber = users[upline].teamNumber.add(1);
emit RefBonus(upline, msg.sender, i, amount);
          upline = users[upline].referrer;
        } else break;
      }

    }

    users[referrer].directReferralReward = users[referrer].directReferralReward.add(m
sg.value.mul(REFERRAL_PERCENTS[0]).div(PERCENTS_DIVIDER));

    if (user.deposits.length == 0) {
      user.checkpoint = block.timestamp;
      totalUsers = totalUsers.add(1);
      users[referrer].referCount  = users[referrer].referCount.add(1);
      emit Newbie(msg.sender);
    }

    users[referrer].referAmout = users[referrer].referAmout.add(msg.value);

    user.deposits.push(Deposit(msg.value, 0, block.timestamp));

    totalInvested = totalInvested.add(msg.value);

    uint256 day = block.timestamp.sub(startTime).div(24*60*60);
    if(day < 7 ){
      for(uint256 i = 0;i < day;i++){
        require(day.mul(day+1).div(2).mul(QUOTA) > totalInvested);
      }
    }
    totalDeposits = totalDeposits.add(1);

    emit NewDeposit(msg.sender, msg.value);

  }
```

```
function withdraw() public {
    User storage user = users[msg.sender];

    uint256 userPercentRate = getUserPercentRate(msg.sender);

    uint256 totalAmount;
    uint256 dividends;

    for (uint256 i = 0; i < user.deposits.length; i++) {

      if (user.deposits[i].withdrawn < user.deposits[i].amount.mul(2)) {

        if (user.deposits[i].start > user.checkpoint) {

          dividends = (user.deposits[i].amount.mul(userPercentRate).div(PERCENTS_DIVID
ER))
            .mul(block.timestamp.sub(user.deposits[i].start))
            .div(TIME_STEP);

        } else {

          dividends = (user.deposits[i].amount.mul(userPercentRate).div(PERCENTS_DIVID
ER))
            .mul(block.timestamp.sub(user.checkpoint))
            .div(TIME_STEP);

        }

        if (user.deposits[i].withdrawn.add(dividends) > user.deposits[i].amount.mul(2)
) {
          dividends = (user.deposits[i].amount.mul(2)).sub(user.deposits[i].withdrawn)
;
        }

        user.deposits[i].withdrawn = user.deposits[i].withdrawn.add(dividends); /// ch
anging of storage data
        totalAmount = totalAmount.add(dividends);

      }
    }

    uint256 referralBonus = getUserReferralBonus(msg.sender);
    if (referralBonus > 0) {
      totalAmount = totalAmount.add(referralBonus);
      user.bonus = 0;
    }

    require(totalAmount > 0, "User has no dividends");

    uint256 contractBalance = address(this).balance;
    if (contractBalance < totalAmount) {
      totalAmount = contractBalance;
    }

    user.checkpoint = block.timestamp;

    msg.sender.transfer(totalAmount);

    userWithdraw[msg.sender] = userWithdraw[msg.sender].add(totalAmount);


    uint256 _vip = getSeniority(msg.sender);
    if(_vip > 0){
      vipWithdraw[msg.sender] = vipWithdraw[msg.sender].add(totalAmount);
    }

     address payable _referrer = users[msg.sender].referrer;
    for(uint256 i = 0;i < 30;i++){
      uint256 senior = getSeniority(_referrer);
      if(levelSeniority[senior] > i){
        _referrer.transfer(totalAmount.mul(VIP_LEVEL_PERCENT[senior]).div(100));
        users[_referrer].teamRevenueReward = users[_referrer].teamRevenueReward.add(to
talAmount.mul(VIP_LEVEL_PERCENT[senior]).div(100));
```

```
totalAmount = totalAmount.add(totalAmount.mul(VIP_LEVEL_PERCENT[senior]).div(100));
      }
    }

    totalWithdrawn = totalWithdrawn.add(totalAmount);

    emit Withdrawn(msg.sender, totalAmount);

  }


  function getSeniority(address user) public view returns(uint256) {
    uint256 utd = getUserTotalDeposits(user);
    uint256 vip = 0;
    for(uint256 i = 0;i<VIP_LEVEL.length;i++){
      if(utd < VIP_LEVEL[i]){
        return i;
      }else{
        vip = i;
      }
    }
    return vip;
  }

  function getContractBalance() public view returns (uint256) {
    return address(this).balance;
  }

  function getContractBalanceRate() public view returns (uint256) {
    uint256 result = 0;
    uint256 contractBalance = address(this).balance;
    uint256 contractBalancePercent = contractBalance.div(CONTRACT_BALANCE_STEP).mul(
5);
    result =  BASE_PERCENT.add(contractBalancePercent);
    if(result > 180){
      result = 180;
    }
    return result;
  }

  function getUserPercentRate(address userAddress) public view returns (uint256) {
    User storage user = users[userAddress];

    uint256 contractBalanceRate = getContractBalanceRate();
    if (isActive(userAddress)) {
      uint256 timeMultiplier = (now.sub(user.checkpoint)).mul(50).div(TIME_STEP);
      contractBalanceRate = contractBalanceRate.add(timeMultiplier);
    } else {
       contractBalanceRate;
    }
    if(contractBalanceRate > 180){
      contractBalanceRate = 180;
    }
    return contractBalanceRate;
  }

  function getUserDividends(address userAddress) public view returns (uint256) {
    User storage user = users[userAddress];

    uint256 userPercentRate = getUserPercentRate(userAddress);

    uint256 totalDividends;
    uint256 dividends;

    for (uint256 i = 0; i < user.deposits.length; i++) {

      if (user.deposits[i].withdrawn < user.deposits[i].amount.mul(2)) {

        if (user.deposits[i].start > user.checkpoint) {

          dividends = (user.deposits[i].amount.mul(userPercentRate).div(PERCENTS_DIVI
DER))
            .mul(block.timestamp.sub(user.deposits[i].start))
            .div(TIME_STEP);

        } else {
```

```
        dividends = (user.deposits[i].amount.mul(userPercentRate).div(PERCENTS_DIVID
ER))
          .mul(block.timestamp.sub(user.checkpoint))
          .div(TIME_STEP);

      }

      if (user.deposits[i].withdrawn.add(dividends) > user.deposits[i].amount.mul(2)
) {
        dividends = (user.deposits[i].amount.mul(2)).sub(user.deposits[i].withdrawn)
;
      }

      totalDividends = totalDividends.add(dividends);

      /// no update of withdrawn because that is view function

    }

  }

  return totalDividends;
 }

 function getUserCheckpoint(address userAddress) public view returns(uint256) {
  return users[userAddress].checkpoint;
 }

 function getUserReferrer(address userAddress) public view returns(address) {
  return users[userAddress].referrer;
 }

 function getUserReferralBonus(address userAddress) public view returns(uint256) {
  return users[userAddress].bonus;
 }

 function getUserAvailable(address userAddress) public view returns(uint256) {
  return getUserReferralBonus(userAddress).add(getUserDividends(userAddress));
 }

 function isActive(address userAddress) public view returns (bool) {
  User storage user = users[userAddress];

  if (user.deposits.length > 0) {
   if (user.deposits[user.deposits.length-1].withdrawn < user.deposits[user.deposit
s.length-1].amount.mul(2)) {
    return true;
   }
  }
 }

 function getUserDepositInfo(address userAddress, uint256 index) public view returns
(uint256, uint256, uint256) {
     User storage user = users[userAddress];

  return (user.deposits[index].amount, user.deposits[index].withdrawn, user.deposit
s[index].start);
 }

 function getUserAmountOfDeposits(address userAddress) public view returns(uint256)
{
  return users[userAddress].deposits.length;
 }

 function getUserTotalDeposits(address userAddress) public view returns(uint256) {
     User storage user = users[userAddress];

  uint256 amount;

  for (uint256 i = 0; i < user.deposits.length; i++) {
   amount = amount.add(user.deposits[i].amount);
  }

  return amount;
 }
```

```
function isContract(address addr) internal view returns (bool) {
        uint size;
        assembly { size := extcodesize(addr) }
        return size > 0;
    }

}

library SafeMath {

    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b <= a, "SafeMath: subtraction overflow");
        uint256 c = a - b;

        return c;
    }

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        uint256 c = a / b;

        return c;
    }

  function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }


    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}
```

5. Appendix B: vulnerability risk rating criteria

Smart contract vulnerability rating standard	
Vulnerability rating	Vulnerability rating description
High risk vulnerability	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.
Middle risk vulnerability	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.
Low risk vulnerability	A vulnerability that is difficult to trigger, or that will harm a limited 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 generated 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.