KNOWNSEC

Degree of secrecy ★ | Project team open

Smart Contract Audit Report

Security status

Safe

★ ★ ★ ★ ★

Principal tester: **KnownSec blockchain security research team**

Release notes

Revised	Time	Revised by	Version
Written document	20201119	KnownSec blockchain security research team	V1.0

Document information

Document name	Document version number	Document number	Confidentiality level
TronContract Smart contract audit report	V1.0	TRON-ZNHY-20201119	**Open project Team**

The statement

KnownSec only issues this report on the facts that have occurred or exist before the issuance of this report, and shall assume the corresponding responsibility therefor.KnownSec is not in a position to judge the security status of its smart contract and does not assume responsibility for the facts that occur or exist after issuance.The security audit analysis and other contents of this report are based solely on the documents and information provided by the information provider to KnownSec as of the issuance of this report.KnownSec assumes that the information provided was not missing, altered, truncated or suppressed.If the information provided is missing, altered, deleted, concealed or reflected in a way inconsistent with the actual situation, KnownSec shall not be liable for any loss or adverse effect caused thereby.

Directory

1. Review

The effective test time of this report is from November 18, 2020 to November 19, 2020. During this period, the security and standardization of the TronContract smart contract code will be audited and used as the statistical basis for the report.

In this test, KnownSec engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (see Chapter 3), and the comprehensive evaluation was passed.

The results of this smart contract security audit: Pass

Since this test is conducted in a non-production environment, all codes are updated, the test process is communicated with the relevant interface personnel, and relevant test operations are carried out under the control of operational risks, so as to avoid production and operation risks and code security risks in the test process.

The target information of this test:

entry	description
Token name	TronContract
Code type	TRON Smart Contract
Code language	Solidity

Contract Documents and Hash:

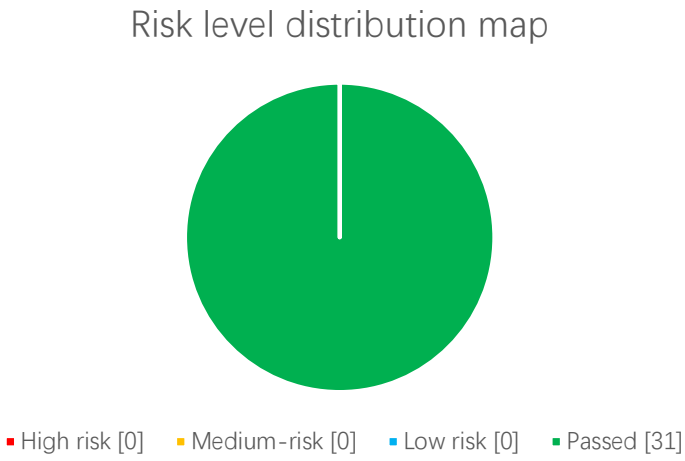
The contract documents	MD5
TronContract.sol	86447ac3a48e50643ead59af546d2982

2. Code vulnerability analysis

2.1. Vulnerability level distribution

This vulnerability risk is calculated by level:

Statistics on the number of security risk levels			
High Risk	Medium Risk	Low Risk	Pass
0	0	0	31



2.2 Summary of audit results

Audit results			
Audit item	Audit item	Audit item	Audit item
Business Security Testing	**Pledge logic design**	pass	After testing, there are no safety issues.
	Take away the pledge logic design	pass	After testing, there are no safety issues.
	Withdrawal reward logic design	pass	After testing, there are no safety issues.
Basic code vulnerability check	Reentry attack detection	pass	After testing, there are no safety issues.
	Replay attack detection	pass	After testing, there are no safety issues.
	Rearrangement attack detection	pass	After testing, there are no safety issues.
	Numerical overflow detection	pass	After testing, there are no safety issues.
	Arithmetic accuracy error	pass	After testing, there are no safety issues.
	Access control defect detection	pass	After testing, there are no safety issues.
	tx.origin authentication	pass	After testing, there are no safety issues.
	call injection attack	pass	After testing, there are no safety issues.
	Return value call verification	pass	After testing, there are no safety issues.
	Uninitialized storage pointer	pass	After testing, there are no safety issues.
	Wrong use of random number detection	pass	After testing, there are no safety issues.

	Transaction order dependency detection	pass	After testing, there are no safety issues.
	Denial of service attack detection	pass	After testing, there are no safety issues.
	Fake recharge vulnerability detection	pass	After testing, there are no safety issues.
	Additional token issuance vulnerability detection	pass	After testing, there are no safety issues.
	Frozen account bypass detection	pass	After testing, there are no safety issues.
	Compiler version security	pass	After testing, there are no safety issues.
	Not recommended encoding	pass	After testing, there are no safety issues.
	Redundant code	pass	After testing, there are no safety issues.
	Use of safe arithmetic library	pass	After testing, there are no safety issues.
	Use of require/assert	pass	After testing, there are no safety issues.
	Energy consumption detection	pass	After testing, there are no safety issues.
	fallback function safety	pass	After testing, there are no safety issues.
	owner permission control	pass	After testing, there are no safety issues.
	Low-level function safety	pass	After testing, there are no safety issues.

	Variable coverage	pass	After testing, there are no safety issues.
	Timestamp dependent attack	pass	After testing, there are no safety issues.
	Unsafe interface use	pass	After testing, there are no safety issues.

3. Business Security Testing

3.1 Pledge logic design[Pass]

Perform security audits on the pledge logic in the contract, check whether the validity of the parameters, the type of contract, and whether the pledge logic is properly designed during the pledge.

Test result: After testing, there are no security issues in the contract.

```
function makeDeposit(address payable ref, uint8 modelType,uint8 payType) external payable _checkPoolInit _checkPlayerInit(msg.sender) _updateA3Time _updateA3Status { //knownsec// Pledge logic
        //Verify whether the activity starts
        require(now>=START_TIME,"Activity not started");
        Player storage player = players[msg.sender];
        //Verify that the contract type is correct
        require(modelType <= DEPOSITS_TYPES_COUNT, "Wrong deposit type");
        //Check recharge amount
        require(
            msg.value >= MINIMAL_DEPOSIT||msg.value <=MAXIMAL_DEPOSIT,
            "Beyond the limit"
        );

        if(modelType==2&&!a3Valve.opening){
            return;
        }
        //Do not recommend yourself
        require(player.active || ref != msg.sender, "Referal can't refer to itself");
        //Check whether the recharge amount is in compliance

require(modelIsBlong2(modelType,MODELTYPE.C)||_checkDepositLimit(msg.value,payType),"T
```

```
ype error");
        require(!_checkBOverLimit(modelType,msg.value,msg.sender),"exceed the limit");
        PROJECT_LEADER.transfer(msg.value.mul(LEADER_COMMISSION).div(100));
        MAINTAINER.transfer(msg.value.mul(MAINTAINER_COMMISSION).div(100));
         _teamCount(ref,msg.value,player.active);
        //Statistics of new registered users
        if (!player.active) {
            playersCount = playersCount.add(1);
            player.active = true;
            if(players[ref].linkEnable){
                player.referrer = ref;
                players[ref].refsCount = players[ref].refsCount.add(1);
            }
        }
        //A contract activates the referral link
        if(modelIsBlong2(modelType,MODELTYPE.A)){
            if(!player.linkEnable){
                player.linkEnable = true;
            }
        }
        //Calculate the pledge reward
        uint256 amount = msg.value.mul(PLANS_PERCENTS[modelType]).div(10000);
        depositsCounter = depositsCounter.add(1);
        player.deposits.push( //knownsec// Pledge
            Deposit({
                id: depositsCounter,
                amount: msg.value,
                modelType: modelType,
                freezeTime: now,
                loanLimit: amount,
                withdrawn: 0,
                lastWithdrawn: now,
                afterVoting: 0
```

```
            })
        );

        uint8 _type = uint8(modelBlong2(modelType));
        player.accumulatives[_type] = player.accumulatives[_type].add(msg.value);

        if(modelIsBlong2(modelType,MODELTYPE.C)){
            if(player.vip<2){
                //500 thousand TRX account is automatically upgraded to VIP1 account
                if(player.accumulatives[_type]>=VIP1){
                    player.vip = 1;
                    //1 million TRX account is automatically upgraded to VIP2 account
                    if(player.accumulatives[_type]>=VIP2){
                        player.vip = 2;
                    }
                }
            }
        }

        //Expiration date of contract
        uint256 _expirationTime = now.add(PLANS_PERIODS[modelType]);
        //User becomes invalid user time
        if(_expirationTime>player.expirationTime){
            player.expirationTime = _expirationTime;
        }
        player.playerDepositAmount = player.playerDepositAmount.add(msg.value);
        totalDepositAmount = totalDepositAmount.add(msg.value);
        emit NewDeposit(depositsCounter, msg.sender, _getReferrer(msg.sender), modelType,
msg.value);
    }
```

```
function makeDepositAgain(uint256 depositId) external payable _checkPoolDestory
```

```
_checkPoolInit _checkPlayerInit(msg.sender) _updateA3Time _updateA3Status{ //knownsec//
Secondary pledge
        Player storage player = players[msg.sender];

        require(player.lastWithdrawTime.add(WITHDRAW_DURATION)<now,"error");
        require(depositId < player.deposits.length, "Out of range");
        Deposit storage deposit = player.deposits[depositId];
        require(modelIsBlong2(deposit.modelType,MODELTYPE.C),"Unsupported type");

        //Check recharge amount
        require(
            msg.value >= MINIMAL_DEPOSIT||msg.value <=MAXIMAL_DEPOSIT,
            "Beyond the limit"
        );

        require(
            deposit.freezeTime.add(PLANS_PERIODS[deposit.modelType])                <=
block.timestamp, //knownsec// Check whether the freezing time has passed
            "Not allowed now"
        );

        PROJECT_LEADER.transfer(msg.value.mul(LEADER_COMMISSION).div(100));
        MAINTAINER.transfer(msg.value.mul(MAINTAINER_COMMISSION).div(100));

         _teamCount(player.referrer,msg.value,player.active);

        if(deposit.afterVoting<3){
            deposit.afterVoting = deposit.afterVoting.add(1);
        }

        uint256 lastDeposit = deposit.amount;
        uint256                                amount                                =
msg.value.mul(PLANS_PERCENTS[deposit.modelType].add(deposit.afterVoting.mul(1000))).div(
```

```
10000);
        deposit.loanLimit = deposit.loanLimit.add(amount);
        deposit.freezeTime = now;
        deposit.lastWithdrawn = now;
        player.accumulatives[2] = player.accumulatives[2].add(msg.value);
        if(player.vip<2){
            if(player.accumulatives[2]>=VIP1){
                player.vip = 1;
                if(player.accumulatives[2]>=VIP2){
                    player.vip = 2;
                }
            }
        }
        uint256 _expirationTime = now.add(PLANS_PERIODS[deposit.modelType]);
        if(_expirationTime>player.expirationTime){
            player.expirationTime = _expirationTime;
        }
        player.playerWithdrawAmount = player.playerWithdrawAmount.add(lastDeposit);
        totalWithdrawAmount = totalWithdrawAmount.add(lastDeposit);

        player.playerDepositAmount = player.playerDepositAmount.add(msg.value);
        totalDepositAmount = totalDepositAmount.add(msg.value);
        deposit.amount = msg.value;
        player.lastWithdrawTime = now;

        _withdraw(msg.sender,lastDeposit);
        emit TakeAwayDeposit(msg.sender, deposit.modelType, lastDeposit);
        emit NewDeposit(depositsCounter, msg.sender, _getReferrer(msg.sender),
deposit.modelType, deposit.amount);

    }
```

Security advice: None.

3.2 Take away the pledge logic design[Pass]

Perform security audits on the logic of taking the pledge from the contract to check whether the parameters are valid when the pledge is taken, whether the reward has been withdrawn, and whether the district's most pledged logic is reasonably designed.

Test result: After testing, there are no security issues in the contract.

```
function takeAwayDeposit(uint256 depositId) external _checkPoolDestory _checkPoolInit _checkPlayerInit(msg.sender) _updateA3Time _updateA3Status returns (uint256) { //knownsec// Take away pledge
        Player storage player = players[msg.sender];
        require(player.lastWithdrawTime.add(WITHDRAW_DURATION)<now,"error");
        //Check the serial number of contract
        require(depositId < player.deposits.length, "Out of range");
        Deposit memory deposit = player.deposits[depositId];
        //Check whether the revenue is extracted
        require(deposit.withdrawn>=deposit.loanLimit.mul(99).div(100), "First need to withdraw reward");
        //Check whether the contract expires
        require(
            deposit.freezeTime.add(PLANS_PERIODS[deposit.modelType]) <= block.timestamp,
            "Not allowed now"
        );
        //Type B contracts do not support withdrawals
        require(!modelIsBlong2(deposit.modelType,MODELTYPE.B),"Unsupported type");
        //Check whether the amount is sufficient
        require(address(this).balance >= deposit.amount, "TRX not enought");
        if (depositId < player.deposits.length.sub(1)) {
          player.deposits[depositId] = player.deposits[player.deposits.length.sub(1)];
        }
```

```
        player.deposits.pop();
        player.lastWithdrawTime = now;
        player.playerWithdrawAmount = player.playerWithdrawAmount.add(deposit.amount);
        totalWithdrawAmount = totalWithdrawAmount.add(deposit.amount);
        msg.sender.transfer(deposit.amount);
        emit TakeAwayDeposit(msg.sender, deposit.modelType, deposit.amount);
    }
```

Safety advice: None.

3.3 Withdrawal reward logic design[Pass]

Perform a security audit on the cash withdrawal reward logic in the contract, check whether the parameters are valid when the pledge is taken, whether the reward has been withdrawn, and whether the district's most pledged logic is reasonably designed.

Test result: After testing, there are no security issues in the contract.

```
function withdrawReward(uint256 depositId) external _checkPoolDestory _checkPoolInit _checkPlayerInit(msg.sender) _updateA3Time _updateA3Status returns (uint256) { //knownsec// Withdrawal loan amount
        Player storage player = players[msg.sender];
        require(player.lastWithdrawTime.add(WITHDRAW_DURATION)<now,"error");
        require(depositId < player.deposits.length, "Out of range");
        Deposit storage deposit = player.deposits[depositId];
        uint256 currTime = now;

require(modelIsBlong2(deposit.modelType,MODELTYPE.C)||deposit.lastWithdrawn.add(WITHDRAW_DURATION)<currTime||deposit.freezeTime.add(PLANS_PERIODS[deposit.modelType])<= block.timestamp, "less than 8 hours");

        uint256 amount = outputReward(msg.sender,depositId);
        require(amount!=0,"Already withdrawn");
```

```
        deposit.withdrawn = deposit.withdrawn.add(amount);
        deposit.lastWithdrawn = currTime;
        require(deposit.withdrawn<=deposit.loanLimit,"error   "); //knownsec// The amount withdrawn must be less than the upper limit of the loan

        if(modelIsBlong2(deposit.modelType,MODELTYPE.B)){
            if(deposit.withdrawn==deposit.loanLimit){
                if (depositId < player.deposits.length.sub(1)) {
                    player.deposits[depositId] = player.deposits[player.deposits.length.sub(1)];
                }
                player.deposits.pop();
            }
        }
        uint256 _vipReward;
        if(deposit.modelType!=2){
            _vipReward= getVipReward(player.vip,amount);
            allocateTeamReward(amount,msg.sender,deposit.modelType);
        }
        player.playerWithdrawAmount = player.playerWithdrawAmount.add(amount.add(_vipReward));
        totalWithdrawAmount = totalWithdrawAmount.add(amount.add(_vipReward));
        player.lastWithdrawTime = now;
        _withdraw(msg.sender, amount.add(_vipReward));
        emit Withdraw(msg.sender, deposit.amount, PLANS_PERCENTS[deposit.modelType], amount.add(_vipReward));
        return amount.add(_vipReward);
    }
```

Safety advice: None.

4. Basic code vulnerability check

4.1. Reentry attack detection [Pass]

Re-entry holes are The most famous ethereum intelligent contract holes that have led to The DAO hack of Ethereum.

The call.value() function in Soldesert consumes all the gas it receives when it is used to send Ether, and there is a risk of a reentrant attack if the operation to send Ether is called to the call.value() function before it actually reduces the balance in the sender's account.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.2. Replay attack detection [Pass]

If the requirement of delegation management is involved in the contract, attention should be paid to the non-reusability of verification to avoid replay attack

In the asset management system, there are often cases of entrusted management in which the principal gives the assets to the agent for management and the principal pays a certain fee to the agent. This business scenario is also common in smart contracts.

Detection results: After detection, the call function is not used in the intelligent contract, so there is no such vulnerability.

Safety advice: None.

4.3. Rearrangement attack detection [Pass]

A rearrangement attack is an attempt by a miner or other party to "compete" with an intelligent contract participant by inserting their information into a list or mapping, thereby giving the attacker an opportunity to store their information in the contract.

Detection results: After detection, there is no relevant vulnerability in the intelligent contract code.

Safety advice: None.

4.4. Numerical overflow detection [Pass]

The arithmetic problem in intelligent contract refers to integer overflow and integer underflow.

Instead of trying to contain something deep inside the body, which is capable of processing a maximum of 256 digits (2^256-1), a maximum increase of 1 would allow the body to drain down to zero.Similarly, when the number is unsigned, 0 minus 1 overflows to get the maximum number value.

Integer overflow and underflow are not a new type of vulnerability, but they are particularly dangerous in smart contracts. Overflow scenarios can lead to incorrect results, especially if the possibility is not anticipated, and can affect the reliability and security of the program.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.5. Arithmetic precision error [Pass]

Solidity as a programming language and common programming language similar data structure design, such as: variables, constants, and functions, arrays, functions, structure and so on, Solidity and common programming language also has a larger difference - no floating-point Solidity, Solidity and all the numerical computing results can only be an integer, decimal will not happen, also not allowed to define the decimal data type.The numerical calculation in the contract is essential, and the design of numerical calculation may cause relative error, such as the same-level calculation: 5/2*10=20, and 5*10/2=25, resulting in error, and the error will be larger and more obvious when the data is larger.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.6. Access control detection [Pass]

Reasonable permissions should be set for different functions in the contract

Check whether the functions in the contract have correctly used keywords such as public and private for visibility modification, and check whether the contract has correctly defined and used modifier to restrict access to key functions, so as to avoid problems caused by overstepping authority.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.7. tx. origin Authentication [Pass]

tx. origin, a global variable that iterates over the entire call stack and returns the address of the account that originally sent the call (or transaction).Using this variable for authentication in a smart contract makes the contract vulnerable to phishing attacks.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.8. Call injection attack [Pass]

When the call function is called, strict permission control should be done, or the dead call function should be written directly.

Detection results: After detection, the call function is not used in the intelligent contract, so there is no such vulnerability.

Safety advice: None.

4.9. Return value call validation [Pass]

This problem occurs mostly in smart contracts associated with currency transfers, so it is also known as silent failed or unchecked send.

A transfer method, such as transfer(), send(), or call.value(), could all be used to send Ether to an address, with the difference between throw and state rollback if the transfer fails;Only 2300GAS will be passed for invocation to prevent reentrant attack;Send returns false on failure;Only 2300GAS will be passed for invocation to prevent reentrant attack;Call.value returns false on failure;Passing all available gas for invocation (which can be restricted by passing in the GAS_value parameter) does not effectively prevent a reentrant attack.

If the return value of the send and call.value transfer function above is not checked in the code, the contract will continue to execute the following code, possibly causing unexpected results due to the failure of Ether to send.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.10. Uninitialized storage pointer [Pass]

A special data structure is allowed to be struct in Soldesert, and local variables inside the function are stored in storage or memory by default.

Presence of storage and memory are two different concepts, which would involve trying to involve a pointer to an uninitialized reference, whereas an uninitialized local stroage would cause variables to point to other stored variables, leading to variable overwrite, or even more serious consequences, and struct variables should be avoided in development from initializing struct variables in functions.

Detection results: After detection, the intelligent contract code does not use the structure, there is no such problem.

Safety advice: None.

4.11. Error using random number [Pass]

In intelligent contracts may need to use a random number, although the Solidity of functions and variables can access the value of the unpredictable obviously such as block. The number and block. The timestamp, but they usually or more open than it looks, or is affected by the miners, that is, to some extent, these random Numbers is predictable, so a malicious user can copy it and usually rely on its unpredictability to attack the function.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.12. Transaction order dependence [Pass]

Since miners always get gas fees through a code that represents an externally owned address (EOA), users can specify higher fees for faster transactions.Because the Ethereum blockchain is public, everyone can see the content of other people's pending transactions. This means that if a user submits a valuable solution, a malicious user can steal the solution and copy its transaction at a higher cost to preempt the original solution.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.13. Denial of service attack [Pass]

In ethereum's world, denial of service is deadly, and smart contracts that suffer from this type of attack may never return to normal functioning. The reasons for intelligent contract denial of service can be many, including malicious behavior while on the receiving end of a transaction, gas depletion due to the artificial addition of needed gas for computing functions, abuse of access control to access private components of intelligent contracts, exploitation of obtuse and negligence, and so on.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.14. False recharge vulnerability [Pass]

In the transfer function of the token contract, the balance check over the originator (MSG. sender) becomes an if judgment method. When the veto [MSg. sender] < value, enter the else logic part and return false, no exception will become available. We believe that the if/else gentle judgment method is an unrigorous coding method in the transfer sensitive function scene.

Detection results: After detection, there is no security problem in the intelligent

contract code.

Safety advice: None.

4.15. Issue of token loopholes [Pass]

Check to see if there are functions in the scrip contract that could increase the scrip total after initializing the scrip total.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.16. Freeze account to bypass [Pass]

Check whether the source account, the originating account and the target account are not checked when the token is transferred in the token contract.

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.17. Compiler version security [Pass]

Check that a secure compiler version is used in the contract code implementation

Detection results: After detection, the compiler version of the intelligent contract code is more than 0.5.8, there is no such security problem.

Safety advice: None.

4.18. Coding Method not recommended [Pass]

Check the contract code implementation to see if there are any officially recommended or deprecated encoding options

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.19. Redundant code [Pass]

Check if the contract code implementation contains redundant code

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.20. Use of secure arithmetic library [Pass]

Check if the SafeMath security arithmetic library is used in the contract code implementation

Detection results: The SafeMath security arithmetic library has been used in the intelligent contract code. There is no security problem.

Safety advice: None.

4.21. Use of require/ Assert [Pass]

Check the reasonableness of the use of require and Assert statements in your

contractual code implementation

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.22. Energy consumption detection [Pass]

Check whether the energy consumption exceeds the maximum block limit.

Test result: After testing, the security problem does not exist in the smart contract code.

Safety advice: None.

4.23. Fallback function security [Pass]

Check that the Fallback function is used correctly in the contract code implementation

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.24. Owner permission control [Pass]

Check if the Owner in the contract code implementation has too many permissions.For example, arbitrarily modify other account balances, etc.

Detection results: After detection, there is no security problem in the intelligent

contract code.

Safety advice: None.

4.25. Low-level function security [Pass]

Check for security vulnerabilities caused by the use of the low-level functions called/Delegatecall used by the contract code implementation

The execution context of the call function is in the contract being invoked;The execution context of the Delegatecall function is in the contract where the function is currently called

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.26. Variable coverage [Pass]

Check the contract code implementation for security issues caused by variable overwriting

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.27. Timestamp dependency attack [Pass]

The timestamp of the data block usually USES the miner's local time, which can

fluctuate in the range of about 900 seconds. When other nodes accept a new block, they only need to verify that the timestamp is later than the previous block and within 900 seconds of the local time.A miner can profit by setting the timestamp of the block to meet conditions as favorable to him as possible.

Check to see if there is any key functionality that depends on the timestamp in the contract code implementation

Detection results: After detection, there is no security problem in the intelligent contract code.

Safety advice: None.

4.28. Use of unsafe interfaces [Pass]

Check whether unsafe interfaces are used in the contract code implementation

Detection results: After detection, there is no security problem in the intelligent contract code.

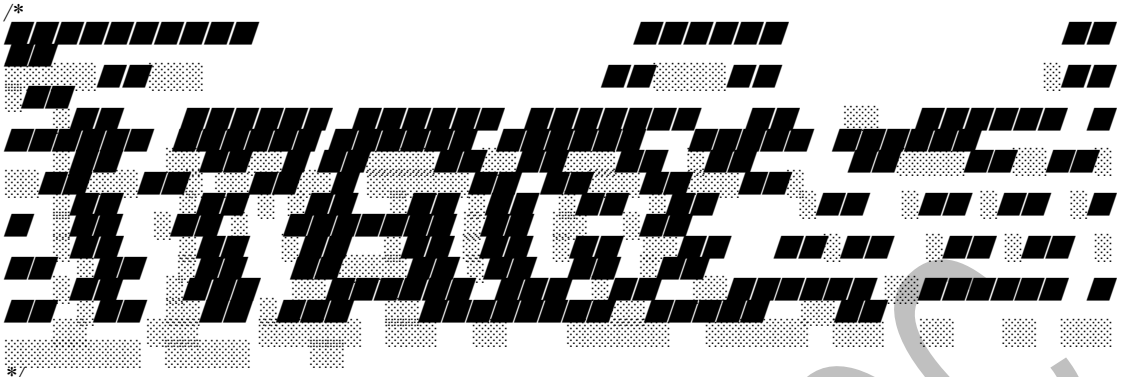
Safety advice: None.

5. Appendix A: Contract code

Source code for this test:

TronContract.sol

```
/*
```



```
*/
pragma solidity ^0.5.1;

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) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }


    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        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) {
        return div(a, b, "SafeMath: division by zero");
    }


    function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

        require(b > 0, errorMessage);
        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;
    }
}

contract Context {


    constructor () internal { }


    function _msgSender() internal view returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view returns (bytes memory) {
        this;
        return msg.data;
    }
}

contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);


    constructor () internal {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }


    function owner() public view returns (address) {
        return _owner;
    }


    modifier onlyOwner() {
        require(isOwner(), "Ownable: caller is not the owner");
        _;
    }


    function isOwner() public view returns (bool) {
        return _msgSender() == _owner;
    }


    function renounceOwnership() public onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }


    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }


    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}

contract Model {
   enum MODELTYPE {A,B,C}
   uint256[6] internal PLANS_PERCENTS = [500, 2000, 1000, 11000, 13000, 2000];
   uint256[6] internal PLANS_PERIODS = [5 days, 15 days, 3 days, 10 days, 20 days, 30 days];
   uint256[15] internal MODEL_A_REWARDS_PERCENTS =[100,50,60,20,20,20,20,20,20,20,30,30,30,30,30];
   uint256[15] internal MODEL_B_REWARDS_PERCENTS =[30,10,20,2,2,2,2,2,2,2,5,5,5,5,5];
   uint256[15] internal MODEL_C_REWARDS_PERCENTS =[30,10,20,2,2,2,2,2,2,2,5,5,5,5,5];
   uint256[15][6] internal MODEL_REWARDS_PERCENTS;
   uint256[7] internal MODEL_AB_DEPOSIT_LIMIT =[100 trx,500 trx,1000 trx,5000 trx,10000 trx,50000
trx,100000 trx];
   uint8[3] internal VIP_REWARD_PERCENTS = [0,5,10];
   constructor() public{
        MODEL_REWARDS_PERCENTS[0] = MODEL_A_REWARDS_PERCENTS;
        MODEL_REWARDS_PERCENTS[1] = MODEL_A_REWARDS_PERCENTS;
        MODEL_REWARDS_PERCENTS[2] = MODEL_A_REWARDS_PERCENTS;
        MODEL_REWARDS_PERCENTS[3] = MODEL_B_REWARDS_PERCENTS;
        MODEL_REWARDS_PERCENTS[4] = MODEL_B_REWARDS_PERCENTS;
```

```
        MODEL_REWARDS_PERCENTS[5] = MODEL_C_REWARDS_PERCENTS;
    }
    //Query contract type(A 0,B 1,C 2)
    function modelBlong2(uint8 depositType) internal pure returns (MODELTYPE tys){
        require(depositType>=0&&depositType<6,"depositType error");
        if(depositType==0||depositType==1||depositType==2){
            return MODELTYPE.A;
        }else if(depositType==3||depositType==4){
            return MODELTYPE.B;
        }else{
            return MODELTYPE.C;
        }
    }

    function modelIsBlong2(uint8 depositType,MODELTYPE tys) internal pure returns (bool){
        return modelBlong2(depositType)==tys;
    }

}
contract TronContract is Ownable,Model{
    using SafeMath for uint256; //knownsec// Call SafeMath function to prevent overflow
    constructor() public payable{
        require(msg.value>=30 trx,"Cannot create a contract"); //knownsec// 30trx is required to create a
contract

        a3Valve = A3Valve(0,false,CREATE_TIME);
    }
    struct Deposit {
        //contract NO
        uint256 id;
        //investment amount
        uint256 amount;
        //Contract Subdivision type0~5
        uint8 modelType;
        uint256 freezeTime;
        //Withdrawal amount
        uint256 withdrawn;
        //Total incentive amount pledged
        uint256 loanLimit;
        //Last withdrawal time
        uint256 lastWithdrawn;
        //After shots
        uint256 afterVoting;
    }

    struct Player{
        //Referral address
        address payable referrer;
        //Whether to activate the recommended link (need to invest more than 100trx in Contract A)
        bool linkEnable;
        //Recommended awards
        uint256 referralReward;
        //Current pledge record
        Deposit[] deposits;
        //As the first recharge mark, activate after completion
        bool active;
        //recommended
        uint256 refsCount;
        //User VIP level
        uint8 vip;
        //A,B,C total investment
        uint256[3] accumulatives;
        //The last time the contract expires
        uint256 expirationTime;
        //Total team size
        uint256 teamCount;
        //Total number of TRX deposits
        uint256 playerDepositAmount;
        //Total number of TRX extracted
        uint256 playerWithdrawAmount;
        //Team performance
        uint256 teamPerformance;

        uint256 lastWithdrawTime;
    }

    uint256 totalDepositAmount;

    uint256 totalWithdrawAmount;

    struct A3Valve{
        //The previous day total capital pool
        uint256 previousTotalSupply;
        //Whether the A3 contract is activated
        bool opening;
        //The day before the funds count time
        uint256 previousRecordTime;
```

```
    }

    address payable constant public PROJECT_LEADER = address(0x416081D6D2B59D168D0DB37B75D06322C3951456D6);

    address payable constant public MAINTAINER = address(0x41C240327C0A474D0A860701A47FE3FF4677680D05);

    uint8 constant private LEADER_COMMISSION = 8;

    uint8 constant private MAINTAINER_COMMISSION = 2;

    //Minimum recharge amount
    uint256 public constant MINIMAL_DEPOSIT = 100 trx;
     //Maximum recharge amount
    uint256 public constant MAXIMAL_DEPOSIT = 100000 trx;

    uint256 public constant DESTORY_LIMIT = 100 trx;
    //Transaction record delimiter
    uint256 private constant ROWS_IN_DEPOSIT = 10;
    //Total number of transaction types
    uint8 private constant DEPOSITS_TYPES_COUNT = 6;
    //Transaction records show the total
    uint256 private constant POSSIBLE_DEPOSITS_ROWS_COUNT = 200;
    //Vip1 shall accumulate the amount of recharge
    uint256 private constant VIP1 = 500000 trx;
    //The amount of viP2 recharge should be accumulated
    uint256 private constant VIP2 = 1000000 trx;
    //Number of players
    uint256 public playersCount;
    //Recharge counter
    uint256 public depositsCounter;
    //The restart time of the capital pool
    uint256 public clearStartTime;
    mapping(address => Player) public players;
    //A3 contract switch
    A3Valve public a3Valve;
    //Contract start time
    uint256 private constant CREATE_TIME = 1605225600;
    //Activity start time
    uint256 private constant START_TIME = 1605240000;
    uint256 private constant ONE_DAY = 1 days;
    //Withdrawal cooldown time
    uint256 private constant WITHDRAW_DURATION = 8 hours;
    //The total team bonus is 3.3
    uint8 private constant teamRewardLimit = 33;
    uint8 private constant ROWS = 10;
    //Capital pool version
    uint256 version;
     //The player version
    mapping(address => uint256) public versionMaps;
    //Reward to be extracted
    mapping(address => uint256) private referRewardMap;
    event NewDeposit(
        uint256 depositId,
        address account,
        address referrer,
        uint8 modelType,
        uint256 amount
    );
    event Withdraw(address account,    uint256 originalAmount, uint256 level_percent, uint256 amount);
    event TransferReferralReward(address player, uint256 amount);
    event AllocateReferralReward(address ref, address player,uint256 _amount,uint256 percent, uint8 modelType,uint256 refReward);
    event TakeAwayDeposit(address account, uint8 depositType, uint256 amount);

    function getA3Status() public view returns(bool){
        return a3Valve.opening;
    }
    function getBalance() public view returns (uint){
        return address(this).balance;
    }
    function getDuration() public view returns (uint256 ){
        return now.sub(CREATE_TIME).div(ONE_DAY).add(1);
    }
    //Access to investment restrictions 0~6
    function _getPayType() internal view returns(uint256){
        uint256    _duration = now.sub(CREATE_TIME).div(ONE_DAY);

        if(_duration<3){
            _duration = 2;
        }
        if(_duration>6){
            _duration = 6;
        }
        return _duration;
    }
```

```
    function referRewardMaps(address player) external view returns(uint256){
        if(!checkUpdate(player)){
            return referRewardMap[player];
        }
    }
    //Check whether the limit is exceeded
    function _checkDepositLimit(uint256 _amount,uint8 payType) private view returns (bool){
        if(_getPayType()<payType){
            return false;
        }
        uint256 dictAmount = MODEL_AB_DEPOSIT_LIMIT[payType];

        if(dictAmount!=_amount){
            return false;
        }else{
            return true;
        }
    }
    //Check whether contract B exceeds the limit
    function _checkBOverLimit(uint8 modelType,uint256 _amount,address _player) private view returns (bool){
        if(modelIsBlong2(modelType,MODELTYPE.B)){
            if(_getTypeTotal(msg.sender,MODELTYPE.B).add(_amount)>players[_player].accumulatives[2]){
                return true;
            }
        }
    }
    //Team Performance statistics
    function _teamCount(address _ref,uint256 amount,bool active) private{
        address player = _ref;
        for (uint256 i = 0; i < MODEL_REWARDS_PERCENTS[0].length; i++) {
            if (player == address(0)||!players[player].linkEnable) {
                break;
            }
            if(!active){
                players[player].teamCount++;
            }
            players[player].teamPerformance = players[player].teamPerformance.add(amount);
            player = players[player].referrer;
        }
    }
    //Update A3 switching time
    modifier _updateA3Time() {
        uint256 _duration = now.sub(a3Valve.previousRecordTime).div(ONE_DAY);
        if(_duration>1){
            a3Valve.previousRecordTime = a3Valve.previousRecordTime.add(_duration.mul(ONE_DAY));
            a3Valve.previousTotalSupply = address(this).balance;
        }
        _;
    }
    //Update A3 switch status
    modifier _updateA3Status(){
        _;
        uint256 previousTotalSupply = a3Valve.previousTotalSupply;
        if(previousTotalSupply==uint256(0)){
            a3Valve.opening = false;
        }else if(previousTotalSupply>address(this).balance){
            //Drop more than 1% to open A3

a3Valve.opening=(previousTotalSupply.sub(address(this).balance)).mul(100).div(previousTotalSupply)>10;
        }else{
            //Increase more than 2% to close A3

a3Valve.opening=(address(this).balance.sub(previousTotalSupply)).mul(100).div(previousTotalSupply)<20;
        }
    }
    //pledge
    function makeDeposit(address payable ref, uint8 modelType,uint8 payType)        external payable
_checkPoolInit _checkPlayerInit(msg.sender) _updateA3Time _updateA3Status { //knownsec// Pledge logic
        //Verify whether the activity starts
        require(now>=START_TIME,"Activity not started");
        Player storage player = players[msg.sender];
        //Verify that the contract type is correct
        require(modelType <= DEPOSITS_TYPES_COUNT, "Wrong deposit type");
        //Check recharge amount
        require(
            msg.value >= MINIMAL_DEPOSIT||msg.value <=MAXIMAL_DEPOSIT, //knownsec// The logical
operator is used incorrectly. When the value of msg.value is greater than MAXIMAL_DEPOSIT, because || before
msg.value >= MINIMAL_DEPOSIT is established, the following content will not be judged, so you should use &&
to judge
            "Beyond the limit"
        );

        if(modelType==2&&!a3Valve.opening){
            return;
        }
        //Do not recommend yourself
```

```
        require(player.active || ref != msg.sender, "Referal can't refer to itself");
        //Check whether the recharge amount is in compliance
        require(modelIsBlong2(modelType,MODELTYPE.C)||_checkDepositLimit(msg.value,payType),"Type
error");
        require(!_checkBOverLimit(modelType,msg.value,msg.sender),"exceed the limit");
        PROJECT_LEADER.transfer(msg.value.mul(LEADER_COMMISSION).div(100));
        MAINTAINER.transfer(msg.value.mul(MAINTAINER_COMMISSION).div(100));
            teamCount(ref,msg.value,player.active);
        //Statistics of new registered users
        if (!player.active) {
            playersCount = playersCount.add(1);
            player.active = true;
            if(players[ref].linkEnable){
                player.referrer = ref;
                players[ref].refsCount = players[ref].refsCount.add(1);
            }
        }
        //A contract activates the referral link
        if(modelIsBlong2(modelType,MODELTYPE.A)){
            if(!player.linkEnable){
                player.linkEnable = true;
            }
        }
        //Calculate the pledge reward
        uint256 amount = msg.value.mul(PLANS_PERCENTS[modelType]).div(10000);
        depositsCounter = depositsCounter.add(1);
        player.deposits.push( //knownsec// Pledge
            Deposit({
                id: depositsCounter,
                amount: msg.value,
                modelType: modelType,
                freezeTime: now,
                loanLimit: amount,
                withdrawn: 0,
                lastWithdrawn: now,
                afterVoting: 0
            })
        );

        uint8 _type = uint8(modelBlong2(modelType));
        player.accumulatives[_type] = player.accumulatives[_type].add(msg.value);

        if(modelIsBlong2(modelType,MODELTYPE.C)){
            if(player.vip<2){
                //500 thousand TRX account is automatically upgraded to VIP1 account
                if(player.accumulatives[_type]>=VIP1){
                    player.vip = 1;
                    //1 million TRX account is automatically upgraded to VIP2 account
                    if(player.accumulatives[_type]>=VIP2){
                        player.vip = 2;
                    }
                }
            }
        }

        //Expiration date of contract
        uint256 _expirationTime = now.add(PLANS_PERIODS[modelType]);
        //User becomes invalid user time
        if(_expirationTime>player.expirationTime){
            player.expirationTime = _expirationTime;
        }
        player.playerDepositAmount = player.playerDepositAmount.add(msg.value);
        totalDepositAmount = totalDepositAmount.add(msg.value);
        emit NewDeposit(depositsCounter, msg.sender, _getReferrer(msg.sender), modelType, msg.value);
    }
    //C contract renewed
    function makeDepositAgain(uint256 depositId) external payable checkPoolDestory _checkPoolInit
_checkPlayerInit(msg.sender) _updateA3Time _updateA3Status{ //knownsec// Secondary pledge
        Player storage player = players[msg.sender];

        require(player.lastWithdrawTime.add(WITHDRAW_DURATION)<now,"error");
        require(depositId < player.deposits.length, "Out of range");
        Deposit storage deposit = player.deposits[depositId];
        require(modelIsBlong2(deposit.modelType,MODELTYPE.C),"Unsupported type");

        //Check recharge amount
        require(
            msg.value >= MINIMAL_DEPOSIT||msg.value <=MAXIMAL_DEPOSIT, //knownsec// There is
also a logical operator problem here, you should use &&
            "Beyond the limit"
        );

        require(
            deposit.freezeTime.add(PLANS_PERIODS[deposit.modelType]) <= block.timestamp, //knownsec//
检测是否过了冻结时间
            "Not allowed now"
        );
```

```
        PROJECT_LEADER.transfer(msg.value.mul(LEADER_COMMISSION).div(100));
        MAINTAINER.transfer(msg.value.mul(MAINTAINER_COMMISSION).div(100));

          _teamCount(player.referrer,msg.value,player.active);

        if(deposit.afterVoting<3){
            deposit.afterVoting = deposit.afterVoting.add(1);
        }

        uint256 lastDeposit = deposit.amount;
        uint256                                                amount                                                =
msg.value.mul(PLANS_PERCENTS[deposit.modelType].add(deposit.afterVoting.mul(1000))).div(10000);
        deposit.loanLimit = deposit.loanLimit.add(amount);
        deposit.freezeTime = now;
        deposit.lastWithdrawn = now;
        player.accumulatives[2] = player.accumulatives[2].add(msg.value);
        if(player.vip<2){
            if(player.accumulatives[2]>=VIP1){
                player.vip = 1;
                if(player.accumulatives[2]>=VIP2){
                    player.vip = 2;
                }
            }
        }
        uint256 _expirationTime = now.add(PLANS_PERIODS[deposit.modelType]);
        if(_expirationTime>player.expirationTime){
            player.expirationTime = _expirationTime;
        }
        player.playerWithdrawAmount = player.playerWithdrawAmount.add(lastDeposit);
        totalWithdrawAmount = totalWithdrawAmount.add(lastDeposit);

        player.playerDepositAmount = player.playerDepositAmount.add(msg.value);
        totalDepositAmount = totalDepositAmount.add(msg.value);
        deposit.amount = msg.value;
        player.lastWithdrawTime = now;

        _withdraw(msg.sender,lastDeposit);
        emit TakeAwayDeposit(msg.sender, deposit.modelType, lastDeposit);
        emit     NewDeposit(depositsCounter,     msg.sender,     _getReferrer(msg.sender),     deposit.modelType,
deposit.amount);

    }


    function _withdraw(address payable _wallet, uint256 _amount) private {
        require(address(this).balance >= _amount, "TRX not enougth");
        _wallet.transfer(_amount);
    }
    //Out this operation
    function     takeAwayDeposit(uint256     depositId)     external     _checkPoolDestory     _checkPoolInit
_checkPlayerInit(msg.sender) _updateA3Time _updateA3Status returns (uint256) { //knownsec// Take away pledge
        Player storage player = players[msg.sender];
        require(player.lastWithdrawTime.add(WITHDRAW_DURATION)<now,"error");
        //Check the serial number of contract
        require(depositId < player.deposits.length, "Out of range");
        Deposit memory deposit = player.deposits[depositId];
        //Check whether the revenue is extracted
        require(deposit.withdrawn>=deposit.loanLimit.mul(99).div(100), "First need to withdraw reward");
        //Check whether the contract expires
        require(
            deposit.freezeTime.add(PLANS_PERIODS[deposit.modelType]) <= block.timestamp,
            "Not allowed now"
        );
        //Type B contracts do not support withdrawals
        require(!modelIsBlong2(deposit.modelType,MODELTYPE.B),"Unsupported type");
        //Check whether the amount is sufficient
        require(address(this).balance >= deposit.amount, "TRX not enought");
        if (depositId < player.deposits.length.sub(1)) {
            player.deposits[depositId] = player.deposits[player.deposits.length.sub(1)];
        }
        player.deposits.pop();
        player.lastWithdrawTime = now;
        player.playerWithdrawAmount = player.playerWithdrawAmount.add(deposit.amount);
        totalWithdrawAmount = totalWithdrawAmount.add(deposit.amount);
        msg.sender.transfer(deposit.amount);
        emit TakeAwayDeposit(msg.sender, deposit.modelType, deposit.amount);
    }
    function _getReferrer(address _player) private view returns (address payable) {
        return players[_player].referrer;
    }
    //Obtain A, B, C type of effective total investment
    function _getTypeTotal(address _player,MODELTYPE tys) private view returns(uint256 totalAmount) {
        if(!checkUpdate(_player)){
            Player memory player = players[_player];
            uint256 _typeTotal = 0;
            if(player.expirationTime>now){
```

```
                    for(uint256 i =0;i<player.deposits.length;i++){
                        Deposit memory _deposit = player.deposits[i];
                        //Obtain a valid contract

if(modelIsBlong2(_deposit.modelType,tys)&&_deposit.freezeTime.add(PLANS_PERIODS[_deposit.modelType])>
now){
                            _typeTotal = _typeTotal.add(_deposit.amount);
                        }
                    }
                }
                return _typeTotal;
            }

        }
        function _getTeamTotalLimit(address _player) public view returns (uint256 teamTotalLimit){
            return players[_player].accumulatives[0].mul(teamRewardLimit).div(10);
        }
        //Allocate team rewards
        function allocateTeamReward(uint256 _amount, address _player, uint8 modelType) private { //knownsec//分配
团队奖励
            address player = _player;
            address payable ref = _getReferrer(player);
            uint256 refReward;
            for (uint256 i = 0; i < MODEL_REWARDS_PERCENTS[modelType].length; i++) {
                //Illegal referrer to skip
                if (ref == address(0x0)||!players[ref].linkEnable) {
                    break;
                }
                //Invalid user
                if(players[ref].expirationTime<now){
                    break;
                }
                //Invalid user
                if(checkUpdate(_player)){
                    break;
                }
                if(players[ref].refsCount<i.add(1)){
                    continue;
                }
                refReward = (_amount.mul(MODEL_REWARDS_PERCENTS[modelType][i]).div(1000));
                //Award cap A class investment 3.3 times
                uint256 teamTotalLimit = _getTeamTotalLimit(ref);
                //No reward will be given beyond the limit
                if(players[ref].referralReward.add(refReward)>teamTotalLimit){
                    refReward = 0;
                }
                //User recommendation reward
                players[ref].referralReward = players[ref].referralReward.add(refReward);
                referRewardMap[ref] = referRewardMap[ref].add(refReward);
                emit                    AllocateReferralReward(ref,                                    player,
_amount,MODEL_REWARDS_PERCENTS[modelType][i], modelType, refReward);
                player = ref;
                ref = players[ref].referrer;
            }
        }
        function withdrawReferReward() external _checkPoolDestory _checkPoolInit _checkPlayerInit(msg.sender)
_updateA3Time _updateA3Status returns (uint256){ //knownsec// Withdraw referral rewards
            uint256 refReward = referRewardMap[msg.sender];
            require(players[msg.sender].lastWithdrawTime.add(WITHDRAW_DURATION)<now,"error");
            require(refReward>0,"error ");
            require(address(this).balance >= refReward,"error");

            players[msg.sender].playerWithdrawAmount                                                    =
players[msg.sender].playerWithdrawAmount.add(refReward);
            totalWithdrawAmount = totalWithdrawAmount.add(refReward);
            referRewardMap[msg.sender] = 0;
            players[msg.sender].lastWithdrawTime = now;
            msg.sender.transfer(refReward);
            emit TransferReferralReward(msg.sender, refReward);
        }

         function getLastWithdrawTime(address _player) external    view returns (uint256 withdrawTime){
            if(!checkUpdate(_player)){
                return players[_player].lastWithdrawTime;
            }
        }

        //Extractable income
        function outputReward(address _player,uint256 depositId) public view returns (uint256){
            if(!checkUpdate(_player)){
                Player memory player = players[_player];
                Deposit memory deposit = player.deposits[depositId];
                if(modelIsBlong2(deposit.modelType,MODELTYPE.C)){
                    return deposit.loanLimit.sub(deposit.withdrawn);
                }
                if(deposit.freezeTime.add(PLANS_PERIODS[deposit.modelType])<=now){
                    return deposit.loanLimit.sub(deposit.withdrawn);
```

```
                }else{
                    return
deposit.loanLimit.mul(now.sub(deposit.lastWithdrawn)).div(PLANS_PERIODS[deposit.modelType]);
                }
            }
        }
        //Withdrawal loan amount
        function withdrawReward(uint256 depositId) external _checkPoolDestory checkPoolInit
_checkPlayerInit(msg.sender) _updateA3Time _updateA3Status returns (uint256) { //knownsec// Withdrawal loan
amount
            Player storage player = players[msg.sender];
            require(player.lastWithdrawTime.add(WITHDRAW_DURATION)<now,"error");
            require(depositId < player.deposits.length, "Out of range");
            Deposit storage deposit = player.deposits[depositId];
            uint256 currTime = now;

require(modelIsBlong2(deposit.modelType,MODELTYPE.C)||deposit.lastWithdrawn.add(WITHDRAW_DURATIO
N)<currTime||deposit.freezeTime.add(PLANS_PERIODS[deposit.modelType]) <= block.timestamp, "less than 8
hours");

            uint256 amount = outputReward(msg.sender,depositId);
            require(amount!=0,"Already withdrawn");
            deposit.withdrawn = deposit.withdrawn.add(amount);
            deposit.lastWithdrawn = currTime;
            require(deposit.withdrawn<=deposit.loanLimit,"error "); //knownsec// The amount withdrawn must be
less than the upper limit of the loan

            if(modelIsBlong2(deposit.modelType,MODELTYPE.B)){
                if(deposit.withdrawn==deposit.loanLimit){
                    if (depositId < player.deposits.length.sub(1)) {
                        player.deposits[depositId] = player.deposits[player.deposits.length.sub(1)];
                    }
                    player.deposits.pop();
                }
            }
            uint256 _vipReward;
            if(deposit.modelType!=2){
                _vipReward= getVipReward(player.vip,amount);
                allocateTeamReward(amount,msg.sender,deposit.modelType);
            }
            player.playerWithdrawAmount = player.playerWithdrawAmount.add(amount.add(_vipReward));
            totalWithdrawAmount = totalWithdrawAmount.add(amount.add(_vipReward));
            player.lastWithdrawTime = now;
            _withdraw(msg.sender, amount.add(_vipReward));
            emit Withdraw(msg.sender, deposit.amount, PLANS_PERCENTS[deposit.modelType],
amount.add(_vipReward));
            return amount.add(_vipReward);
        }
        function getVipReward(uint8 _vip,uint256 amount) internal view returns(uint256){
            return amount.mul(VIP_REWARD_PERCENTS[_vip]).div(100);
        }
        modifier _checkPlayerInit(address _player){
            if(checkUpdate(_player)){
                clearPlayer(_player);
            }
            _;
        }
        //Verify that the user version number is consistent with the current version
        function checkUpdate(address _player) private view returns (bool){
            uint256 subVersion = version.sub(versionMaps[_player]);
            if(subVersion==0){
                return false;
            }else if(subVersion==1){
                if(now.sub(clearStartTime)<ONE_DAY){
                    return false;
                }else{
                    return true;
                }
            }else{
                return true;
            }
        }
        //The pool is below the DESTORY_LIMIT, triggering a restart
        modifier _checkPoolDestory(){
            _;
            if(clearStartTime==0){
                if(address(this).balance<DESTORY_LIMIT){
                    clearStartTime = now;
                    version = version.add(1);
                }
            }
        }
        //Inconsistent version Numbers user clears transaction records
        function clearPlayer(address _player) private{
            Player storage player = players[_player];
            delete player.deposits;
            player.expirationTime = 0;
```

```
        player.lastWithdrawTime = 0;
        referRewardMap[_player] = 0;
        versionMaps[_player] = version;
    }
    //Verify that the pool is restarted
    modifier _checkPoolInit() {
        if(clearStartTime!=0){
            if(now.sub(clearStartTime)>=ONE_DAY){
                clearStartTime = 0;
            }
        }
        _;
    }
     //The entire network information
    function getGlobalStats() external view returns (uint256[5] memory stats) {
        stats[0] = totalDepositAmount;
        stats[1] = address(this).balance;
        stats[2] = totalWithdrawAmount;
        stats[3] = playersCount;
        stats[4] = clearStartTime;
        if(clearStartTime!=0){
            if(now.sub(clearStartTime)>ONE_DAY){
                stats[4] = 0;
            }
        }

    }
    //The pledge to record
    function getDeposits(address _player) public view returns (uint256[POSSIBLE_DEPOSITS_ROWS_COUNT]
memory deposits) {
        if(!checkUpdate(_player)){
            Player memory player = players[_player];
            for (uint256 i = 0; i < player.deposits.length; i++) {
                uint256[ROWS_IN_DEPOSIT] memory deposit = depositStructToArray(i,player.deposits[i]);
                for (uint256 row = 0; row < ROWS_IN_DEPOSIT; row++) {
                    deposits[i.mul(ROWS_IN_DEPOSIT).add(row)] = deposit[row];
                }
            }
        }

    }
     //paging
    function getDeposits(address _player,uint256 page) public view returns (uint256[100] memory deposits) {
        Player memory player = players[_player];

        if(!checkUpdate(_player)){
            uint256 start = page.mul(ROWS);
            uint256 init = start;
            uint256 _totalRow = player.deposits.length;
            if(start.add(ROWS)< _totalRow){
                _totalRow = start.add(ROWS);
            }
            for (start; start <  _totalRow; start++) {
                uint256[ROWS_IN_DEPOSIT]                memory                deposit                =
depositStructToArray(start,player.deposits[start]);
                for (uint256 row = 0; row < ROWS_IN_DEPOSIT; row++) {
                    deposits[(start.sub(init)).mul(ROWS_IN_DEPOSIT).add(row)] = deposit[row];
                }
            }
        }

    }
     //Personal information
    function getPersonalStats(address _player) external view returns (uint256[14] memory stats) {
        Player memory player = players[_player];
        stats[0] = player.accumulatives[0];
        stats[1] = _getTypeTotal(_player,MODELTYPE.A);
        stats[2] = player.accumulatives[1];
        stats[3] = _getTypeTotal(_player,MODELTYPE.B);
        stats[4] = player.accumulatives[2];
        stats[5] = _getTypeTotal(_player,MODELTYPE.C);
        uint256 teamTotalLimit = _getTeamTotalLimit(_player);
        if(teamTotalLimit<player.referralReward){
            stats[6] = 0;
        }else{
            stats[6] = teamTotalLimit.sub(player.referralReward);
        }
        stats[7] = player.referralReward;
        stats[8] = player.vip;
        stats[9] = player.refsCount;
        stats[10] = player.teamCount;
        stats[11] = player.playerDepositAmount;
        stats[12] = player.playerWithdrawAmount;
        stats[13] = player.teamPerformance;
    }
    function depositStructToArray(uint256 depositId,Deposit memory deposit) private view returns
(uint256[ROWS_IN_DEPOSIT] memory depositArray) {
```

```
            depositArray[0] = depositId;
            depositArray[1] = deposit.amount;
            depositArray[2] = deposit.modelType;
            depositArray[3] = PLANS_PERCENTS[deposit.modelType].add(deposit.afterVoting.mul(1000));
            depositArray[4] = PLANS_PERIODS[deposit.modelType];
            depositArray[5] = deposit.freezeTime;
            depositArray[6] = deposit.withdrawn;
            depositArray[7] = deposit.loanLimit;
            depositArray[8] = deposit.id;
            depositArray[9] = deposit.lastWithdrawn;
        }
    }
```

6. Appendix B: Vulnerability risk rating criteria

Smart contract vulnerability rating standards	
Vulnerability rating	Vulnerability rating description
High-risk vulnerabilities	Vulnerabilities that can directly cause the loss of token contracts or user funds, such as: value overflow loopholes that can cause the value of tokens to zero, fake recharge loopholes that can cause exchanges to lose tokens, and can cause contract accounts to lose ETH or tokens. Access loopholes, etc.;
Mid-risk vulnerability	Vulnerabilities that can cause loss of ownership of token contracts, such as: access control defects of key functions, call injection leading to bypassing of access control of key functions, etc.;
Low-risk vulnerabilities	Vulnerabilities that can cause the token contract to not work properly, such as: denial of service vulnerability caused by sending ETH to malicious addresses, and denial of service vulnerability caused by exhaustion of gas.

7. Appendix C: Introduction to vulnerability testing tools

7.1 Manticore

A Manticore is a symbolic execution tool for analyzing binary files and smart contracts. A Manticore consists of a symbolic Ethereum virtual machine (EVM), an EVM disassembler/assembler, and a convenient interface for automatic compilation and analysis of the Solarium body.It also incorporates Ethersplay, a Bit of Traits of Bits visual disassembler for EVM bytecode, for visual analysis. Like binaries, Manticore provides a simple command-line interface and a Python API for analyzing EVM bytecode.

7.2 Oyente

Oyente is a smart contract analysis tool that can be used to detect common bugs in smart contracts, such as reentrancy, transaction ordering dependencies, and so on.More conveniently, Oyente's design is modular, so this allows power users to implement and insert their own inspection logic to check the custom properties in their contracts.

7.3 securify. Sh

Securify verifies the security issues common to Ethereum's smart contracts, such as unpredictability of trades and lack of input verification, while fully automated and analyzing all possible execution paths, and Securify has a specific language for identifying vulnerabilities that enables the securities to focus on current security and other reliability issues at all times.

7.4 Echidna

Echidna is a Haskell library designed for fuzzy testing EVM code.

7.5 MAIAN

MAIAN is an automated tool used to find holes in Ethereum's smart contracts. MAIAN processes the bytecode of the contract and tries to set up a series of transactions to find and confirm errors.

7.6 ethersplay

Ethersplay is an EVM disassembler that includes correlation analysis tools.

7.7 IDA - evm entry

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

7.8 want - ide

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

7.9 KnownSec Penetration Tester kit

KnownSec penetration tester's toolkit, developed, collected and used by KnownSec penetration tester engineers, contains batch automated testing tools, self-developed tools, scripts or utilization tools, etc. dedicated to testers.