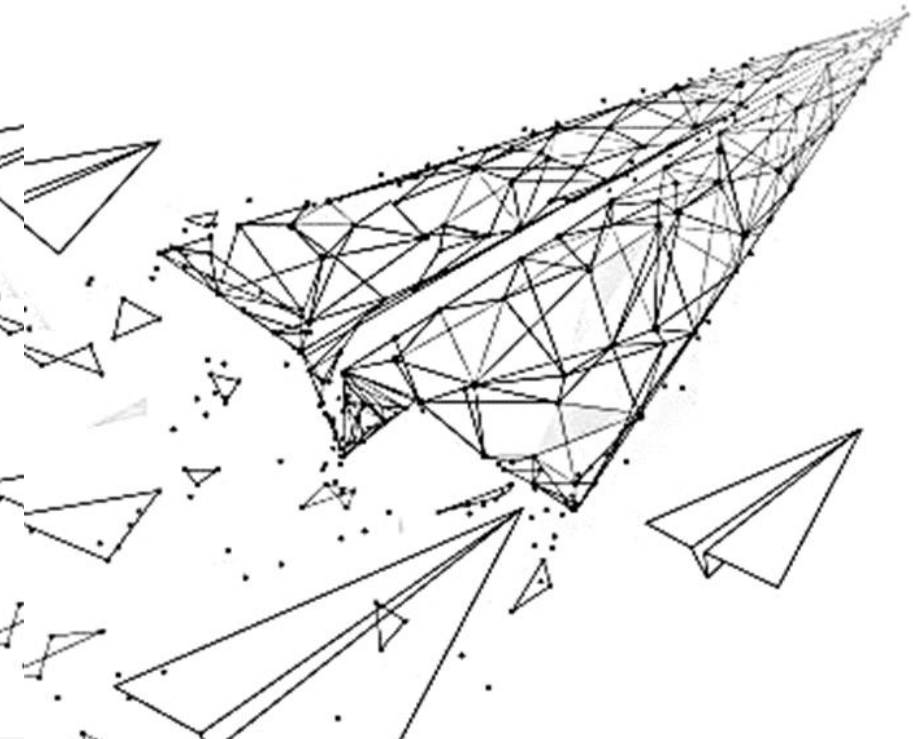
盾牌安全

2020

STARK AUDIT REPORT



File No.: DP- 20200927STARK

Table of Contents

I. Overview

The date of issuing this report is September 27, 2020. The security and specifications of the **STARK** smart contract code are audited and used as the statistical basis for the report.

In this test, a comprehensive analysis of common vulnerabilities in smart contracts (see Chapter 3) was conducted. The **STARK** contract code complies with the ERC-20 specification, and no known vulnerabilities have been found; thus, the comprehensive audit of **STARK** is **passed.**

Since this test process is carried out in a testing environment, all codes are the latest version. The test process communicates with the person in charge of the relevant interface, and conducts related test operations under the control of operational risks to avoid the production, operational code and security risks during test process.

Target Information For This Test: Module Name

Token: Stark Chain (STARK)

Code Type: Token code

Contract Address: **0x1eDC9bA729Ef6FB017ef9c687b1A37D48B6a166C**

Address of The Token:

https://cn.etherscan.com/token/0x1eDC9bA729Ef6FB017ef9c687b1A37D48B6a166C

Programming Language: solidity

II. Code Vulnerability Analysis

1. Vulnerability Level Distribution

The vulnerability risk level according to statistics:

High Risk 0

Medium Risk 0

Low Risk 0

Passed 11

2. Audit Results Summary

2.1. Reentry Attack Detection:

Check if the use of call.value() function is security, **passed.**

2.2. Numerical Overflow Detection:

Check if the add and sub functions are security to use, **passed.**

2.3. Access Control Defect Detection:

Check access control of each operation, **passed.**

2.4. Call of Unverified Return Value:

Check the transfer method to see if the return value is verified, **passed.**

2.5. Random Number Using Detection:

Check whether there is a unified content filter, **passed.**

2.6. Transaction Sequence Dependency Detection:

Check transaction sequence dependency, **passed.**

2.7. Denial of Service Attack Detection:

Check whether the code has resource abuse problems when using resources, **passed.**

2.8. Logical Design Defect Detection:

Check the security issues related to business design in the smart contract code, **passed.**

2.9. Fake Charge Vulnerability Detection:

Check whether there is a fake charge vulnerability in the smart contract code, **passed.**

2.10. Additional Issuance Vulnerability Detection:

Check whether there is a function of additional issuance in the smart contract code, **passed.**

2.11. Frozen Account Bypass:

Check whether there is a frozen account bypass problem in the smart contract code, **passed.**

III. Analysis of Code Audit Results

1. Reentry Attack Detection: [Passed]

The reentry vulnerability is the most famous Ethereum smart contract vulnerability, which has led to the Ethereum fork (The DAO hack).

The call.value() function in Solidity consumes all the gas it receives when it is used to send Ether. There is a risk of reentry attacks when the call.value() function sending Ether occurs before actually reducing the balance of the sender's account,

Test result: After testing, there is no relevant call in the smart contract code.

Suggestions: None.

2. Numerical Overflow Detection: [Passed]

The arithmetic problem in smart contracts refers to integer overflow and integer underflow.

Solidity can handle up to 256 digits (2^256-1), and increasing the maximum number by 1 will overflow to 0. Similarly, when the number is of unsigned type, 0 minus 1 will underflow 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. An overflow situation can lead to incorrect results, especially if the possibility is not expected, which may affect the reliability and security of the program.

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

Suggestions: None.

3. **Access Control Test:** **[Passed]**

Access control deficiencies are possible security risks in all programs. Smart contracts also have similar problems. The famous Parity Wallet smart contract has been affected by this problem.

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

Suggestions: None.

4. **Return Value Call Verification:** **[Passed]**

This problem mostly occurs in smart contracts related to token transfer, so it is also called silent failure transmission or unchecked transmission.

In Solidity, there are transfer methods such as transfer(), send(), call.value(), etc., which can be used to send Ether to an address, the difference is that: transfer will throw when the transfer fails, and the status will be rolled back; Only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when send fails to send; only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when call.value fails to be sent; all available gas will be called for Restricted by passing in the gas_value parameter), cannot effectively prevent reentry attacks. If the return value of the above send and call.value transfer functions is not checked in the code, the contract will continue to execute the following code, which may cause unexpected results due to the failure of Ether transmission.

Test result: After testing, there are no related vulnerabilities in the smart contract code.

Suggestions: None.

5. **Random Number Using:[Passed]**

Random numbers may be required in smart contracts. Although the functions and variables provided by Solidity can access obviously unpredictable values, such as block.number and block.timestamp, they are usually either more public than they seem or are affected by miners, these random numbers are predictable to a certain extent, so malicious users can usually copy it and rely on its unpredictability to attack the function.

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

Suggestions: None.

6. **Transaction Sequence Dependency: [Passed]**

Since miners always obtain gas fees through code representing externally owned addresses (EOA), users can specify a higher fee for faster transactions. Since 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 fee to preempt the original solution.

Test result: After testing, there is no risk of transaction order dependence attack in the approval function in the smart contract.

Suggestions: None.

7. **Denial of Service Attack: [Passed]**

In the Ethereum world, denial of service is fatal, and a smart contract that suffers this type of attack may never be able to return to its normal working state. There may be many reasons for the denial of service of the smart contract, including malicious behavior when acting as the recipient of the transaction, artificially increasing the gas required for the calculation function leads to the exhaustion of gas, abuse of access

control to access the private component of the smart contract, the use of confusion and negligence, etc.

Test result: After testing, there are no related vulnerabilities in the smart contract code.

Suggestions: None.

8. **Logic Design Defects:** **[Passed]**

Detect security issues related to business design in smart contract code.

Test result: After testing, there are no related vulnerabilities in the logic design of the smart contract code.

Suggestions: None

9. **Fake Charge Vulnerability:** **[Passed]**

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

Test result: After testing, there are no related vulnerabilities in the smart contract code.

Suggestions: None.

10. **Vulnerability of Additional Issuance:** **[Passed]**

After the initialization of the total amount of tokens, check whether there is a function in the token contract that may increase the total amount of tokens.

Test result: After testing, there are no additional token issuance vulnerabilities in the smart contract code.

Suggestions: None.

11. **Frozen Account Bypass:** **[Passed]**

Check whether there is any operation of unverified token source account, originating account and target account when transferring tokens in the token contract.

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

Suggestions: None.

IV. Appendix A: Contract Code

Note: For audit opinions and recommendations, see code comments

//AUDIT//...

```
//AUDIT// There is no overflow or conditional competition in the contract.
/**
 *Submitted for verification at Etherscan.io on 2020-08-19
*/
pragma solidity >=0.5.16 <0.6.9;
pragma experimental ABIEncoderV2;
//YOUWILLNEVERWALKALONE
interface tokenRecipient {
    function receiveApproval(address _from, uint256 _value, address _token, bytes calldata _extraData) external;
```

```
}

contract StarkChain {
    string public name;
    string public symbol;
    uint8 public decimals = 18;
    uint256 public totalSupply;
    address payable public fundsWallet;
    uint256 public maximumTarget;
    uint256 public lastBlock;
    uint256 public rewardTimes;
    uint256 public genesisReward;
    uint256 public premined;
    uint256 public nRewarMod;
    uint256 public nWtime;
    mapping (address => uint256) public balanceOf;
    mapping (address => mapping (address => uint256)) public allowance;
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed _owner, address indexed _spender, uint256 _value);
    event Burn(address indexed from, uint256 value);
    constructor(
        uint256 initialSupply,
        string memory tokenName,
        string memory tokenSymbol
```

```
) public {
        initialSupply = 8923155   * 10 ** uint256(decimals);
        tokenName = "Stark Chain";
        tokenSymbol = "STARK";
        lastBlock = 0;
        nRewarMod = 5700;
        nWtime = 7776000;
        genesisReward = (10**uint256(decimals)); // Ödül Miktarı
        maximumTarget = 100   * 10 ** uint256(decimals);
        fundsWallet = msg.sender;
        premined = 35850 * 10 ** uint256(decimals);
        balanceOf[msg.sender] = premined;
        balanceOf[address(this)] = initialSupply;
        totalSupply =   initialSupply + premined;
        name = tokenName;
        symbol = tokenSymbol;
    }
    function _transfer(address _from, address _to, uint _value) internal {
        require(_to != address(0x0));
        require(balanceOf[_from] >= _value);
        require(balanceOf[_to] + _value >= balanceOf[_to]);
        uint previousBalances = balanceOf[_from] + balanceOf[_to];
        balanceOf[_from] -= _value;
```

```
        balanceOf[_to] += _value;
        emit Transfer(_from, _to, _value);
        assert(balanceOf[_from] + balanceOf[_to] == previousBalances);
    }
    function transfer(address _to, uint256 _value) public returns (bool success) {
        _transfer(msg.sender, _to, _value);
        return true;//AUDIT// The return value conforms to the ERC-20 specification.
    }
    function transferFrom(address _from, address _to, uint256 _value) public returns (bool
success) {
        require(_value <= allowance[_from][msg.sender]);     // Check allowance
        allowance[_from][msg.sender] -= _value;
        _transfer(_from, _to, _value);
        return true;//AUDIT// The return value conforms to the ERC-20 specification.
    }
    function approve(address _spender, uint256 _value) public
        returns (bool success) {
        allowance[msg.sender][_spender] = _value;
        emit Approval(msg.sender, _spender, _value);
        return true;//AUDIT// The return value conforms to the ERC-20 specification.
    }
    function approveAndCall(address _spender, uint256 _value, bytes memory
_extraData)
```

```
        public
        returns (bool success) {
        tokenRecipient spender = tokenRecipient(_spender);
        if (approve(_spender, _value)) {
            spender.receiveApproval(msg.sender, _value, address(this), _extraData);
            return true;
        }
    }
    function burn(uint256 _value) public returns (bool success) {
        require(balanceOf[msg.sender] >= _value);     // Check if the sender has enough
        balanceOf[msg.sender] -= _value;                 // Subtract from the sender
        totalSupply -= _value;                                // Updates totalSupply
        emit Burn(msg.sender, _value);
        return true;
    }

    function burnFrom(address _from, uint256 _value) public returns (bool success) {
        require(balanceOf[_from] >= _value);                          // Check if the targeted
balance is enough
        require(_value <= allowance[_from][msg.sender]);      // Check allowance
        balanceOf[_from] -= _value;                                          // Subtract from the
targeted balance
```

```
        allowance[_from][msg.sender] -= _value;                    // Subtract from the
sender's allowance
        totalSupply -= _value;                                     // Update totalSupply
        emit Burn(_from, _value);
        return true;
    }
    function uintToString(uint256 v) internal pure returns(string memory str) {
        uint maxlength = 100;
        bytes memory reversed = new bytes(maxlength);
        uint i = 0;
        while (v != 0) {
            uint remainder = v % 10;
            v = v / 10;
            reversed[i++] = byte(uint8(48 + remainder));
        }
        bytes memory s = new bytes(i + 1);
        for (uint j = 0; j <= i; j++) {
            s[j] = reversed[i - j];
        }
        str = string(s);
    }
    function append(string memory a, string memory b) internal pure returns (string
memory) {
```

```
        return string(abi.encodePacked(a,"-",b));
    }
    function getCurrentBlockHash() public view returns (uint256) {
            return uint256(blockhash(block.number-1));
    }
    function getBlockHashAlgoritm(uint256 _blocknumber) public view returns(uint256,
uint256){
        uint256 crew = uint256(blockhash(_blocknumber)) % nRewarMod;
        return (crew, block.number-1);
    }
    function checkBlockReward() public view returns (uint256, uint256) {
        uint256 crew = uint256(blockhash(block.number-1)) % nRewarMod;
        return (crew, block.number-1);
    }
    struct stakeInfo {
      uint256 _stocktime;
      uint256 _stockamount;
    }
    address[] totalminers;
    mapping (address => stakeInfo) nStockDetails;
    struct rewarddetails {
        uint256 _artyr;
        bool _didGetReward;
```

```
        bool _didisign;
    }
    mapping (string => rewarddetails) nRewardDetails;
    struct nBlockDetails {
        uint256 _bTime;
        uint256 _tInvest;
    }
    mapping (uint256 => nBlockDetails) bBlockIteration;
    struct activeMiners {
        address bUser;
    }
    mapping(uint256 => activeMiners[]) aMiners;
    function totalMinerCount() view public returns (uint256) {
        return totalminers.length;
    }
    function addressHashs() view public returns (uint256) {
        return uint256(msg.sender) % 10000000000;
    }
    function stakerStatus(address _addr) view public returns(bool){
        if(nStockDetails[_addr]._stocktime == 0)
        {
            return false;
        }
```

```
        else
        {
            return true;
        }
    }

    function stakerAmount(address _addr) view public returns(uint256){
        if(nStockDetails[_addr]._stocktime == 0)
        {
            return 0;
        }
        else
        {
            return nStockDetails[_addr]._stockamount;
        }
    }
    function stakerActiveTotal() view public returns(uint256) {
        return aMiners[lastBlock].length;
    }
    function generalCheckPoint()    private view returns(string memory) {
        return append(uintToString(addressHashs()),uintToString(lastBlock));
    }
    function necessarySignForReward(uint256 _bnumber) public returns (uint256)    {
```

```
        require(stakerStatus(msg.sender) == true);
        require((block.number-1) - _bnumber    <= 100);
        require(nStockDetails[msg.sender]._stocktime + nWtime > now);
        require(uint256(blockhash(_bnumber)) % nRewarMod == 1);
        if(bBlockIteration[lastBlock]._bTime + 1800 < now)
        {
            lastBlock += 1;
            bBlockIteration[lastBlock]._bTime = now;
        }
        require(nRewardDetails[generalCheckPoint()]._artyr == 0);
        bBlockIteration[lastBlock]._tInvest += nStockDetails[msg.sender]._stockamount;
        nRewardDetails[generalCheckPoint()]._artyr = now;
        nRewardDetails[generalCheckPoint()]._didGetReward = false;
        nRewardDetails[generalCheckPoint()]._didisign = true;
        aMiners[lastBlock].push(activeMiners(msg.sender));
        return 200;
    }


    function rewardGet(uint256 _bnumber) public returns(uint256) {
        require(stakerStatus(msg.sender) == true);
        require((block.number-1) - _bnumber    > 100);
        require(uint256(blockhash(_bnumber)) % nRewarMod == 1);
```

```
require(nStockDetails[msg.sender]._stocktime + nWtime > now   );
require(nRewardDetails[generalCheckPoint()]._didGetReward == false);
require(nRewardDetails[generalCheckPoint()]._didisign == true);
uint256 halving = lastBlock / 365;
uint256 totalRA = 128 * genesisReward;

if(halving==0)
{
    totalRA = 128 * genesisReward;
}
else if(halving==1)
{
    totalRA = 256 * genesisReward;
}
else if(halving==2)
{
    totalRA = 512 * genesisReward;
}
else if(halving==3)
{
    totalRA = 1024 * genesisReward;
}
else if(halving==4)
```

```
{
    totalRA = 2048 * genesisReward;
}
else if(halving==5)
{
    totalRA = 4096 * genesisReward;
}
else if(halving==6)
{
    totalRA = 8192 * genesisReward;
}
else if(halving==7)
{
    totalRA = 4096 * genesisReward;
}
else if(halving==8)
{
    totalRA = 2048 * genesisReward;
}
else if(halving==9)
{
    totalRA = 1024 * genesisReward;
}
```

```
else if(halving==10)
{
    totalRA = 512 * genesisReward;
}
else if(halving==11)
{
    totalRA = 256 * genesisReward;
}
else if(halving==12)
{
    totalRA = 128 * genesisReward;
}
else if(halving==13)
{
    totalRA = 64 * genesisReward;
}
else if(halving==14)
{
    totalRA = 32 * genesisReward;
}
else if(halving==15)
{
    totalRA = 16 * genesisReward;
```

```
        }
        else if(halving==16)
        {
            totalRA = 8 * genesisReward;
        }
        else if(halving==17)
        {
            totalRA = 4 * genesisReward;
        }
        else if(halving==18)
        {
            totalRA = 2 * genesisReward;
        }
        else if(halving==19)
        {
            totalRA = 1 * genesisReward;
        }
        else if(halving>19)
        {
            totalRA = 1 * genesisReward;
        }
        uint256 usersReward = (totalRA * (nStockDetails[msg.sender]._stockamount * 100)
/ bBlockIteration[lastBlock]._tInvest) /   100;
```

```
        nRewardDetails[generalCheckPoint()]._didGetReward = true;
        _transfer(address(this), msg.sender, usersReward);
        return usersReward;
    }

    function startMining(uint256 mineamount) public returns (uint256) {
        uint256 realMineAmount = mineamount * 10 ** uint256(decimals);
        require(realMineAmount >= 10 * 10 ** uint256(decimals));
        require(nStockDetails[msg.sender]._stocktime == 0);
        maximumTarget +=  realMineAmount;
        nStockDetails[msg.sender]._stocktime = now;
        nStockDetails[msg.sender]._stockamount = realMineAmount;
        totalminers.push(msg.sender);
        _transfer(msg.sender, address(this), realMineAmount);
        return 200;
    }
    function tokenPayBack() public returns(uint256) {
        require(stakerStatus(msg.sender) == true);
        require(nStockDetails[msg.sender]._stocktime + nWtime < now  );
        nStockDetails[msg.sender]._stocktime = 0;
        _transfer(address(this),msg.sender,nStockDetails[msg.sender]._stockamount);
        return nStockDetails[msg.sender]._stockamount;
    }
```

```
    struct memoInfo {
        uint256 _receiveTime;
        uint256 _receiveAmount;
        address _senderAddr;
        string _senderMemo;
    }
    mapping(address => memoInfo[]) memoGetProcess;
    function sendMemoToken(uint256 _amount, address _to, string memory _memo)
public returns(uint256) {
        memoGetProcess[_to].push(memoInfo(now, _amount, msg.sender, _memo));
        _transfer(msg.sender, _to, _amount);
        return 200;
    }
    function sendMemoOnly(address _to, string memory _memo)  public returns(uint256) {
        memoGetProcess[_to].push(memoInfo(now,0, msg.sender, _memo));
        _transfer(msg.sender, _to, 0);
        return 200;
    }
    function yourMemos(address _addr, uint256 _index) view public returns(uint256,
    uint256,
    string memory,
    address) {
        uint256 rTime = memoGetProcess[_addr][_index]._receiveTime;
```

```
        uint256 rAmount = memoGetProcess[_addr][_index]._receiveAmount;
        string memory sMemo = memoGetProcess[_addr][_index]._senderMemo;
        address sAddr = memoGetProcess[_addr][_index]._senderAddr;
        if(memoGetProcess[_addr][_index]._receiveTime == 0){
            return (0, 0,"0", _addr);
        }else {
            return (rTime, rAmount,sMemo, sAddr);
        }
    }
    function yourMemosCount(address _addr) view public returns(uint256) {
        return  memoGetProcess[_addr].length;
    }

    function appendMemos(string memory a, string memory b,string memory c,string memory d) internal pure returns (string memory) {
         return string(abi.encodePacked(a,"#",b,"#",c,"#",d));
    }

    function addressToString(address _addr) public pure returns(string memory) {
      bytes32 value = bytes32(uint256(_addr));
      bytes memory alphabet = "0123456789abcdef";
      bytes memory str = new bytes(51);
      str[0] = "0";
```

```
        str[1] = "x";
        for (uint i = 0; i < 20; i++) {
            str[2+i*2] = alphabet[uint(uint8(value[i + 12] >> 4))];
            str[3+i*2] = alphabet[uint(uint8(value[i + 12] & 0x0f))];
        }
        return string(str);
    }
    function getYourMemosOnly(address _addr) view public returns(string[] memory) {

        uint total =    memoGetProcess[_addr].length;
        string[] memory messages = new string[](total);
        for (uint i=0; i < total; i++) {
            messages[i] =
appendMemos(uintToString(memoGetProcess[_addr][i]._receiveTime),memoGetProcess[_
addr][i]._senderMemo,uintToString(memoGetProcess[_addr][i]._receiveAmount),addressT
oString(memoGetProcess[_addr][i]._senderAddr));
        }
        return messages;
    }
}
```

V. Appendix B: Vulnerability Risk Rating Standards

Vulnerability Rating	Vulnerability Rating Instructions
High-risk Vulnerabilities	Vulnerabilities that can directly cause the loss of token or user funds, such as: numerical overflow vulnerabilities that can cause the value of the token to return to zero, fake recharge vulnerabilities that can cause the exchange to lose tokens, and reentrant vulnerabilities that can cause contract accounts to losses ETH or token , etc.; vulnerabilities that can cause the loss of ownership of token contracts, such as: access control defects of key functions, bypass of key function access control caused by call injection, etc.; vulnerabilities that can cause token contracts to not work properly, such as: Denial of service vulnerability caused by malicious address sending ETH, denial of service vulnerability due to gas exhaustion.
Medium-risk Vulnerability	High-risk vulnerabilities that require specific addresses to trigger, such as numeric overflow vulnerabilities that can only be triggered by the token contract owner; access control defects of non-critical functions, logical design defects that cannot cause direct capital loss, etc.
Low-risk Vulnerabilities	Vulnerabilities that are difficult to be triggered, vulnerabilities with limited damage after triggering, such as numerical overflow vulnerabilities that require a large amount of ETH or tokens to trigger, vulnerabilities that the attacker cannot directly profit after the numerical overflow is triggered, and the transaction sequence dependence risk triggered by specifying high gas etc.

VI. Appendix C: Introduction to Vulnerability Testing Tools

- **Manticore**

  Manticore is a symbolic execution tool for analyzing binary files and smart contracts. Manticore includes a symbolic Ethereum Virtual Machine (EVM), an EVM disassembler/assembler, and a convenient interface for automatic compilation and analysis of Solidity. It also integrates Ethersplay, a Bit of Traits of Bits visual disassembler for EVM bytecode, for visual analysis. Like binary files, Manticore provides a simple command-line interface and a Python API for analyzing EVM bytecode.

- **Oyente**

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

- **Securify.sh**

  Securify can verify the common security problems of Ethereum smart contracts, such as out-of-order transactions and lack of input verification. It analyzes all possible execution paths of the program while fully automated. In addition, Securify has a specific language for specifying vulnerabilities, which makes Securify can keep abreast of current security and other reliability issues.

- **Echidna**

  Echidna is a Haskell library designed for fuzzing EVM code.

- **MAIAN**

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

- **Ethersplay**

  Ethersplay is an EVM disassembler, which contains related analysis tools.

- **Ida-evm**

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

- **Remix-ide**

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

VII. Declaration

This report only issues the facts that have occurred or existed before the issuance. It is impossible to judge the security status of the project for the facts that occur or exist after the issuance.

The security audit analysis and other contents made in this report are only based on the documents and materials provided by the information provider as of the issuance of this report (referred to as "provided materials"). It is assumed that the information provided is not missing, tampered with, deleted or concealed. If the information provided has been missing, tampered with, deleted, concealed or reflected in a situation that is inconsistent with the actual situation, the resulting losses and adverse effects have nothing to do with this report.

STARK AUDIT REPOR

Official website：http://www.dpanquan.com

Official email：service@dpanquan.com