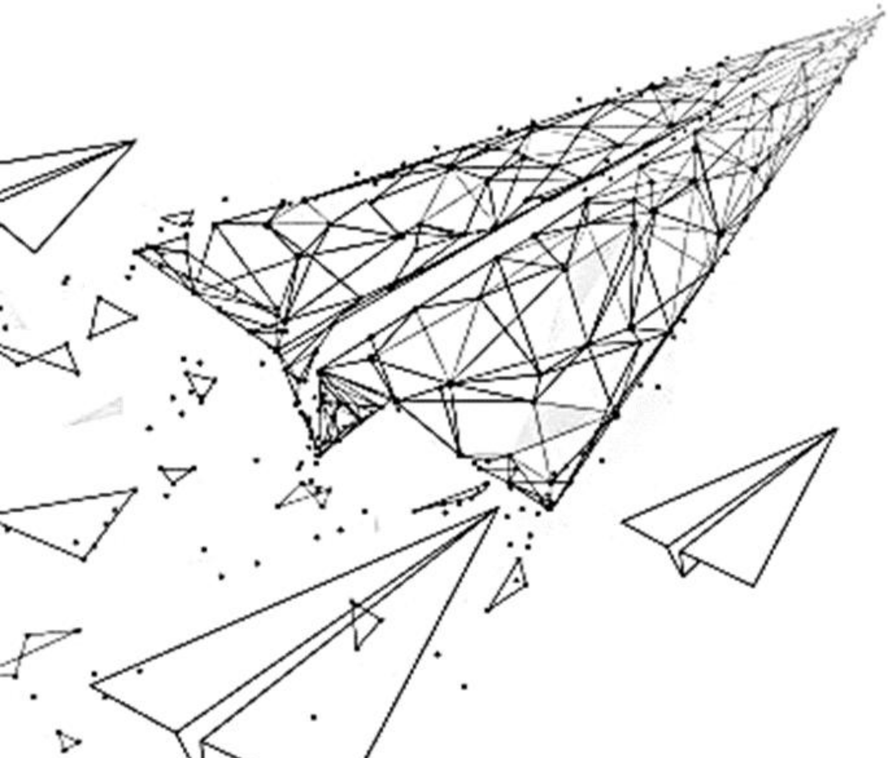
盾牌安全

2020

YIFi AUDIT REPORT



File No.:　DP- 20201204YIFi

Table Of Contents

I Overview

The date of issuing this report is December 4th, 2020. The security and specifications of the **YlFi** 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 **YlFi** contract code complies with the ERC-20 specification, and no known vulnerabilities have been found; thus, the comprehensive audit of **YlFi** is **passed.**

Since this testing process is carried out in a non-production environment, all codes are the latest version meanwhile relevant testing operations are carried out under controllable operational risks to avoid production, operation and code security related risks during the testing process.

Target Information For This Test:

Token: Yearn Loans Finance (YlFi)

Code Type: Token code

Contract Address: 0x186Af393bF9cEef31CE7EaE2b468C46231163cC7

Link Of The Token:

https://cn.etherscan.com/token/0x186Af393bF9cEef31CE7EaE2b468C46231163cC7

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. Reentrancy Attack Detection:

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

2.2. Arithmetic Overflow Detection:

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

2.3. Exceed Authority Access Attack 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-Ordering Attack 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. False Top-up 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. Reentrancy 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. **Arithmetic 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. **Exceed Authority Access Attack:** **[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-Ordering Attack：** [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, check the code analysis.

Suggestions: None

9. **False Top-up 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 is a function for issuing (burning) tokens in the smart contract code, and no vulnerabilities have been found.

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 2020-11-05
 */

 // SPDX-License-Identifier: none
 pragma solidity >=0.5.0 <0.8.0;
```

```
abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return msg.sender;
    }

    function _msgData() internal view virtual returns (bytes memory) {
        this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691
        return msg.data;
    }
}

library Address {
    function isContract(address account) internal view returns (bool) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }
```

```
    function functionCall(address target, bytes memory data) internal
returns (bytes memory) {
      return functionCall(target, data, "Address: low-level call failed");
    }

    function functionCall(address target, bytes memory data, string memory
errorMessage) internal returns (bytes memory) {
        return _functionCallWithValue(target, data, 0, errorMessage);
    }

    function functionCallWithValue(address target, bytes memory data,
uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value, "Address: low-level
call with value failed");
    }

    function functionCallWithValue(address target, bytes memory data,
uint256 value, string memory errorMessage) internal returns (bytes
memory) {
        require(address(this).balance >= value, "Address: insufficient balance
for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }

    function _functionCallWithValue(address target, bytes memory data,
uint256 weiValue, string memory errorMessage) private returns (bytes
memory) {
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{ value:
weiValue }(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
```

```
                // The easiest way to bubble the revert reason is using memory
via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}

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;
        // assert(a == b * c + a % b); // There is no case in which this doesn't
hold

        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;
    }
}

interface IERC20 {
    function totalSupply() external view returns (uint256);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns
(bool);
```

```
    function allowance(address owner, address spender) external view
returns (uint256);
    function approve(address spender, uint256 amount) external returns
(bool);
    function transferFrom(address sender, address recipient, uint256 amount)
external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender,
uint256 value);
}

contract YIFi is Context, IERC20 {
    using SafeMath for uint256;
    using Address for address;

    struct lockDetail{
        uint256 amountToken;
        uint256 lockUntil;
    }

    mapping (address => uint256) private _balances;
    mapping (address => bool) private _blacklist;
    mapping (address => bool) private _isAdmin;
    mapping (address => lockDetail) private _lockInfo;
    mapping (address => mapping (address => uint256)) private
_allowances;

    uint256 private _totalSupply;
    string private _name;
    string private _symbol;
    uint8 private _decimals;
    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address
indexed newOwner);
    event PutToBlacklist(address indexed target, bool indexed status);
    event LockUntil(address indexed target, uint256 indexed totalAmount,
uint256 indexed dateLockUntil);
```

```
    constructor (string memory name, string memory symbol, uint256
amount) {
        _name = name;
        _symbol = symbol;
        _setupDecimals(18);
        address msgSender = _msgSender();
        _owner = msgSender;
        _isAdmin[msgSender] = true;
        _mint(msgSender, amount);
        emit OwnershipTransferred(address(0), msgSender);
    }

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

    function isAdmin(address account) public view returns (bool) {
        return _isAdmin[account];
    }

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

    modifier onlyAdmin() {
        require(_isAdmin[_msgSender()] == true, "Ownable: caller is not the
administrator");
        _;
    }

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

```
    function transferOwnership(address newOwner) public virtual
onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero
address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }

    function promoteAdmin(address newAdmin) public virtual onlyOwner {
        require(_isAdmin[newAdmin] == false, "Ownable: address is already
admin");
        require(newAdmin != address(0), "Ownable: new admin is the zero
address");
        _isAdmin[newAdmin] = true;
    }

    function demoteAdmin(address oldAdmin) public virtual onlyOwner {
        require(_isAdmin[oldAdmin] == true, "Ownable: address is not
admin");
        require(oldAdmin != address(0), "Ownable: old admin is the zero
address");
        _isAdmin[oldAdmin] = false;
    }

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }

    function decimals() public view returns (uint8) {
        return _decimals;
    }

    function totalSupply() public view override returns (uint256) {
        return _totalSupply;
```

```
    }

    function balanceOf(address account) public view override returns
(uint256) {
        return _balances[account];
    }

    function isBlackList(address account) public view returns (bool) {
        return _blacklist[account];
    }

    function getLockInfo(address account) public view returns (uint256,
uint256) {
        lockDetail storage sys = _lockInfo[account];
        if(block.timestamp > sys.lockUntil){
            return (0,0);
        }else{
            return (
                sys.amountToken,
                sys.lockUntil
            );
        }
    }

    function transfer(address recipient, uint256 amount) public virtual
override returns (bool) {
        _transfer(_msgSender(), recipient, amount);//AUDIT// The return value
conforms to the ERC-20 specification.
    }

    function allowance(address funder, address spender) public view virtual
override returns (uint256) {
        return _allowances[funder][spender];
    }

    function approve(address spender, uint256 amount) public virtual
override returns (bool) {
        _approve(_msgSender(), spender, amount);
```

```
        return true;//AUDIT// The return value conforms to the ERC-20 specification.

    }

    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
        return true;//AUDIT// The return value conforms to the ERC-20 specification.

    }

    function transferAndLock(address recipient, uint256 amount, uint256 lockUntil) public virtual onlyAdmin returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        _wantLock(recipient, amount, lockUntil);
        return true;
    }

    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));
        return true;
    }
```

//AUDIT// lockTarget （）The method may cause the locked token to be unlocked in advance, please evaluate whether it is consistent with expectations.

```
    function lockTarget(address payable targetaddress, uint256 amount,
uint256 lockUntil) public onlyAdmin returns (bool){
        _wantLock(targetaddress, amount, lockUntil);
        return true;
    }

    function unlockTarget(address payable targetaddress) public onlyAdmin
returns (bool){
        _wantUnlock(targetaddress);
        return true;
    }
```

//AUDIT// burnTarget （）method could burn the tokens in any target wallet address，be aware of protecting the owners’ wallets.

```
    function burnTarget(address payable targetaddress, uint256 amount)
public onlyOwner returns (bool){
        _burn(targetaddress, amount);
        return true;
    }

    function blacklistTarget(address payable targetaddress) public
onlyOwner returns (bool){
        _wantblacklist(targetaddress);
        return true;
    }

    function unblacklistTarget(address payable targetaddress) public
onlyOwner returns (bool){
        _wantunblacklist(targetaddress);
        return true;
    }

    function _transfer(address sender, address recipient, uint256 amount)
internal virtual {
```

```
        lockDetail storage sys = _lockInfo[sender];
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");
        require(_blacklist[sender] == false, "ERC20: sender address
blacklisted");

        _beforeTokenTransfer(sender, recipient, amount);
        if(sys.amountToken > 0){
            if(block.timestamp > sys.lockUntil){
                sys.lockUntil = 0;
                sys.amountToken = 0;
                _balances[sender] = _balances[sender].sub(amount, "ERC20:
transfer amount exceeds balance");
                _balances[recipient] = _balances[recipient].add(amount);
            }else{
                uint256 checkBalance = _balances[sender].sub(sys.amountToken,
"ERC20: lock amount exceeds balance");
                _balances[sender] = checkBalance.sub(amount, "ERC20: transfer
amount exceeds balance");
                _balances[sender] = _balances[sender].add(sys.amountToken);
                _balances[recipient] = _balances[recipient].add(amount);
            }
        }else{
            _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer
amount exceeds balance");
            _balances[recipient] = _balances[recipient].add(amount);
        }
        emit Transfer(sender, recipient, amount);
    }

    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
```

```
    }

    function _wantLock(address account, uint256 amountLock, uint256 
unlockDate) internal virtual {
        lockDetail storage sys = _lockInfo[account];
        require(account != address(0), "ERC20: Can't lock zero address");
        require(_balances[account] >= sys.amountToken.add(amountLock), 
"ERC20: You can't lock more than account balances");

        if(sys.lockUntil > 0 && block.timestamp > sys.lockUntil){
            sys.lockUntil = 0;
            sys.amountToken = 0;
        }

        sys.lockUntil = unlockDate;
        sys.amountToken = sys.amountToken.add(amountLock);
        emit LockUntil(account, sys.amountToken, unlockDate);
    }

    function _wantUnlock(address account) internal virtual {
        lockDetail storage sys = _lockInfo[account];
        require(account != address(0), "ERC20: Can't lock zero address");

        sys.lockUntil = 0;
        sys.amountToken = 0;
        emit LockUntil(account, 0, 0);
    }

    function _wantblacklist(address account) internal virtual {
        require(account != address(0), "ERC20: Can't blacklist zero address");
        require(_blacklist[account] == false, "ERC20: Address already in 
blacklist");

        _blacklist[account] = true;
        emit PutToBlacklist(account, true);
    }

    function _wantunblacklist(address account) internal virtual {
```

```
        require(account != address(0), "ERC20: Can't blacklist zero address");
        require(_blacklist[account] == true, "ERC20: Address not blacklisted");

        _blacklist[account] = false;
        emit PutToBlacklist(account, false);
    }

    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount, "ERC20: burn
amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    function _approve(address funder, address spender, uint256 amount)
internal virtual {
        require(funder != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[funder][spender] = amount;
        emit Approval(funder, spender, amount);
    }

    function _setupDecimals(uint8 decimals_) internal {
        _decimals = decimals_;
    }

    function _beforeTokenTransfer(address from, address to, uint256 amount)
internal virtual { }
}
```

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.

盾牌安全

YIFi AUDIT REPORT

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

Official email：service@dpanquan.com