KNOWNSEC

Degree of secrecy ★ | Project team open

Smart Contract Audit Report

Security status

Safe

★ ★ ★ ★ ★

Principal tester: Knownsec blockchain security team

Version Summary

Content	Date	Version
Editing Document	20210314	V1.0

Report Information

Title	Version	Document Number	Type
YouSwap Smart Contract Audit Report	V1.0	8b882477a824d8d907e72fbc940fbdf	Open to project team

Copyright Notice

Table of Contents

1. Introduction

The effective test time of this report is from From March 12, 2021 to March 13, 2021 . During this period, the security and standardization of **the smart contract code of the YouSwap** will be audited and used as the statistical basis for the report.

In this audit report, engineers conducted a comprehensive analysis of the common vulnerabilities of smart contracts (Chapter 3). **The smart contract code of the YouSwap** is comprehensively assessed as **SAFE**.

Results of this smart contract security audit: SAFE

Since the testing is under non-production environment, all codes are the latest version. In addition, the testing process is communicated with the relevant engineer, and testing operations are carried out under the controllable operational risk to avoid production during the testing process, such as: Operational risk, code security risk.

Report information of this audit:

Report Number: a8b882477a824d8d907e72fbc940fbdf

Report query address link:

https://attest.im/attestation/searchResult?qurey=a8b882477a824d8d907e72fbc940fbdf

Target information of the YouSwap audit:

Target information	
Token name	YouSwap
Contract address	TokenYou: 0x1d32916CFA6534D261AD53E2498AB95505bd2510 IDO: 0x59E6639bEf44164BA44Aa58771b94DF30BC023A6
Code type	Token code, fundraising agreement code, Ethereum smart contract code

Code language	solidity

Contract documents and hash:

Contract documents	MD5
TokenYou. sol	7d6a52a6d6fc88d388d3da123258fec1
IDO. sol	6e0de7c208535b77bb57e0ed967aef2f

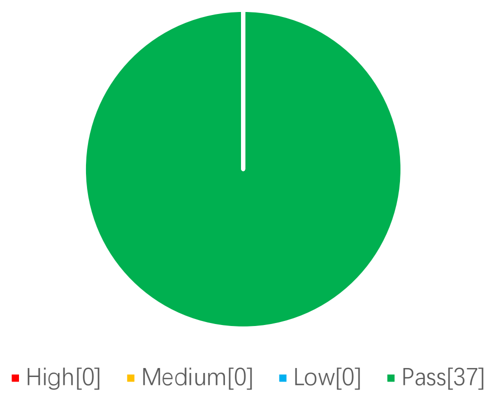
2. Code vulnerability analysis

2.1 Vulnerability Level Distribution

Vulnerability risk statistics by level：

Vulnerability risk level statistics table			
High	Medium	Low	Pass
0	0	0	37

Risk level distribution



2.2 Audit Result

Result of audit			
Audit Target	**Audit**	**Status**	**Audit Description**
Business security testing	IDO contract variables and constructors	Pass	After testing, there is no such safety vulnerability.
	IDO contract whitelist function	Pass	After testing, there is no such safety vulnerability.
	IDO contract privateOffering function	Pass	After testing, there is no such safety vulnerability.
	IDO contract publicOffering function	Pass	After testing, there is no such safety vulnerability.
	IDO contract withdrawPriIDO function	Pass	After testing, there is no such safety vulnerability.
	IDO contract withdrawPubIDO function	Pass	After testing, there is no such safety vulnerability.
	TokenYou contract minting function	Pass	After testing, there is no such safety vulnerability.
	TokenYou contract destruction function	Pass	After testing, there is no such safety vulnerability.
	TokenYou contract minting authority management function	Pass	After testing, there is no such safety vulnerability.
	Basic token functions of TokenYou contract	Pass	After testing, there is no such safety vulnerability.
	Compiler version security	Pass	After testing, there is no such safety vulnerability.

Basic code vulnerability detection	**Redundant code**	Pass	After testing, there is no such safety vulnerability.
	Use of safe arithmetic library	Pass	After testing, there is no such safety vulnerability.
	Not recommended encoding	Pass	After testing, there is no such safety vulnerability.
	Reasonable use of require/assert	Pass	After testing, there is no such safety vulnerability.
	fallback function safety	Pass	After testing, there is no such safety vulnerability.
	tx.oriigin authentication	Pass	After testing, there is no such safety vulnerability.
	Owner permission control	Pass	After testing, there is no such safety vulnerability.
	Gas consumption detection	Pass	After testing, there is no such safety vulnerability.
	call injection attack	Pass	After testing, there is no such safety vulnerability.
	Low-level function safety	Pass	After testing, there is no such safety vulnerability.
	Vulnerability of additional token issuance	Pass	After testing, there is no such safety vulnerability.
	Access control defect detection	Pass	After testing, there is no such safety vulnerability.
	Numerical overflow detection	Pass	After testing, there is no such safety vulnerability.

	Arithmetic accuracy error	Pass	After testing, there is no such safety vulnerability.
	Wrong use of random number detection	Pass	After testing, there is no such safety vulnerability.
	Unsafe interface use	Pass	After testing, there is no such safety vulnerability.
	Variable coverage	Pass	After testing, there is no such safety vulnerability.
	Uninitialized storage pointer	Pass	After testing, there is no such safety vulnerability.
	Return value call verification	Pass	After testing, there is no such safety vulnerability.
	Transaction order dependency detection	Pass	After testing, there is no such safety vulnerability.
	Timestamp dependent attack	Pass	After testing, there is no such safety vulnerability.
	Denial of service attack detection	Pass	After testing, there is no such safety vulnerability.
	Fake recharge vulnerability detection	Pass	After testing, there is no such safety vulnerability.
	Reentry attack detection	Pass	After testing, there is no such safety vulnerability.
	Replay attack detection	Pass	After testing, there is no such safety vulnerability.
	Rearrangement attack detection	Pass	After testing, there is no such safety vulnerability.

3. Analysis of code audit results

3.1. IDO contract variables and constructors【PASS】

Audit analysis: The IDO contract variable definition and the design of the constructor are reasonable.

```
contract IDO is Ownable {
    using SafeMath for uint256;

    //Private offering
    mapping(address => uint256) private _ordersOfPriIDO; //knownsec// Store the number of private tokens for whitelisted users
    uint256 public startHeightOfPriIDO; //knownsec// Store the starting block number of the private sale
    uint256 public endHeightOfPriIDO; //knownsec// Store the starting block number of the private sale
    uint256 public totalUsdtAmountOfPriIDO = 0; //knownsec// Total amount of private sale USDT
    uint256 public supplyYouForPriIDO = 5 * 10 ** 11;//50 万 //knownsec// Private placement You token supply
    uint256 public reservedYouOfPriIDO = 0; //knownsec// The number of You tokens issued by the private sale
    uint256 public constant upperLimitUsdtOfPriIDO = 500 * 10 ** 6;//500USDT //knownsec// User private placement upper limit
    bool public priOfferingFinished = false; //knownsec// Private placement completion mark
    bool private _priIDOWithdrawFinished = false; //knownsec// Withdrawal total amount completed mark
    //knownsec// Private equity related events
    event PrivateOffering(address indexed participant, uint256 amountOfYou, uint256 amountOfUsdt);
    event PrivateOfferingClaimed(address indexed participant, uint256 amountOfYou);

    //Public offering
    mapping(address => uint256) private _ordersOfPubIDO; //knownsec// Store the amount of over-raised USDT for over-raised users
    uint256 public targetUsdtAmountOfPubIDO = 5 * 10 ** 10;//5 万 USDT //knownsec// Over-funding target USDT quantity
    uint256 public targetYouAmountOfPubIDO = 5 * 10 ** 11;//50 万 YOU //knownsec// Over-raising target You issued quantity
    uint256 public totalUsdtAmountOfPubIDO = 0; //knownsec// Total amount of over-raised USDT
    uint256 public startHeightOfPubIDO; //knownsec// Super fundraising start block number
    uint256 public endHeightOfPubIDO; //knownsec// Over-funding end block number
    uint256 public constant bottomLimitUsdtOfPubIDO = 100 * 10 ** 6; //100USDT //knownsec// Over-funding requires a minimum amount of USDT to participate
    bool private _pubIDOWithdrawFinished = false; //knownsec// Withdrawal and over-raising total completed mark
    //knownsec// Super fundraising related events
    event PublicOffering(address indexed participant, uint256 amountOfUsdt);
    event PublicOfferingClaimed(address indexed participant, uint256 amountOfYou);
    event PublicOfferingRefund(address indexed participant, uint256 amountOfUsdt);

    mapping(address => uint8) private _whiteList;

    address private constant _usdtToken = 0xdAC17F958D2ee523a2206206994597C13D831ec7;
    address private _youToken;

    address private constant _vault = 0x6B5C21a770dA1621BB28C9a2b6F282E5FC9154d5;

    constructor(address youToken) public {
        _youToken = youToken;
        //knownsec// Initialize related parameters
        startHeightOfPriIDO = 12043919;
        endHeightOfPriIDO = 12045359;

        startHeightOfPubIDO = 12061199;
        endHeightOfPubIDO = 12062639;
    }
```

Recommendation: nothing.

3.2. IDO contract whitelist function【PASS】

Audit analysis: The whitelist function of the IDO contract normally restricts the caller's participation in accordance with the contract business requirements.

```
//knownsec// Whitelist modifier
 modifier inWhiteList() {
     require(_whiteList[msg.sender] == 1, "YouSwap: NOT_IN_WHITE_LIST");
     _;
 }

 function isInWhiteList(address account) external view returns (bool) {
     return _whiteList[account] == 1;
 }
 //knownsec// Add whitelist
 function addToWhiteList(address account) external onlyOwner {
     _whiteList[account] = 1;
 }
 //knownsec// Batch add whitelist
 function addBatchToWhiteList(address[] calldata accounts) external onlyOwner {
     for(uint i=0;i<accounts.length;i++) {
         _whiteList[accounts[i]] = 1;
     }
 }
 //knownsec// Remove whitelist
 function removeFromWhiteList(address account) external onlyOwner {
     _whiteList[account] = 0;
 }
```

Recommendation： nothing.

3.3. IDO contract privateOffering function【PASS】

Audit analysis: The privateOffering function of the IDO contract is used to participate in the private sale, restricting the participation of whitelisted users, and other ordinary users are not allowed to participate in the private sale.

```
    //knownsec// Participate in private placement
    function privateOffering(uint256 amountOfUsdt) inWhiteList external returns (bool)   {
        //knownsec// Participation conditions are restricted
        require(block.number >= startHeightOfPriIDO, 'YouSwap:NOT_STARTED_YET');
        require(!priOfferingFinished && block.number <= endHeightOfPriIDO, 'YouSwap:PRIVATE_OFFERING_ALREADY_FINISHED');
        require(_ordersOfPriIDO[msg.sender] == 0, 'YouSwap: ATTENDED_ALREADY'); //knownsec// Can only participate in one private sale
        require(amountOfUsdt <= upperLimitUsdtOfPriIDO, 'YouSwap: EXCEED_THE_UPPER_LIMIT');

        require(reservedYouOfPriIDO < supplyYouForPriIDO, 'YouSwap:INSUFFICIENT_YOU');
        uint256 amountOfYou = amountOfUsdt.mul(10);
        //0.1USDT/YOU
        //knownsec// Determine whether the private placement is completed
        if (reservedYouOfPriIDO.add(amountOfYou) > supplyYouForPriIDO) {
            amountOfYou = supplyYouForPriIDO.sub(reservedYouOfPriIDO);
            amountOfUsdt = amountOfYou.div(10);

            priOfferingFinished = true;
        }
        _transferFrom(_usdtToken, amountOfUsdt); //knownsec// After authorizing this contract in advance, transfer the private USDT to this contract address

        _ordersOfPriIDO[msg.sender] = amountOfYou; //knownsec// Write the record of the number of private sale You tokens
        reservedYouOfPriIDO = reservedYouOfPriIDO.add(amountOfYou); //knownsec// Update
on the cumulative number of You tokens issued
        totalUsdtAmountOfPriIDO     =     totalUsdtAmountOfPriIDO.add(amountOfUsdt);
//knownsec// Update on the total amount of private equity participation
        emit PrivateOffering(msg.sender, amountOfYou, amountOfUsdt);

        return true;
    }
```

Recommendation: nothing.

3.4. IDO contract publicOffering function【PASS】

Audit analysis: The publicOffering function of the IDO contract is used to participate in the over-funding, and any user can participate in the over-funding.

```
    //knownsec// Super fundraising completion query function
    function pubOfferingFinished() public view returns (bool) {
        return block.number > endHeightOfPubIDO;
    }
    //knownsec// Participate in super fundraising
    function publicOffering(uint256 amountOfUsdt) external returns (bool)   {
        //knownsec// Participation conditions are restricted
        require(block.number                >=                startHeightOfPubIDO,
'YouSwap:PUBLIC_OFFERING_NOT_STARTED_YET');
        require(block.number                <=                endHeightOfPubIDO,
'YouSwap:PUBLIC_OFFERING_ALREADY_FINISHED');
        require(amountOfUsdt      >=      bottomLimitUsdtOfPubIDO,      'YouSwap:
100USDT_AT_LEAST');

        _transferFrom(_usdtToken, amountOfUsdt); //knownsec// After authorizing this contract
in advance, the over-raised USDT will be transferred to this contract address
```

```
        _ordersOfPubIDO[msg.sender] = _ordersOfPubIDO[msg.sender].add(amountOfUsdt);
//knownsec// Update the number of users over-raised
        totalUsdtAmountOfPubIDO = totalUsdtAmountOfPubIDO.add(amountOfUsdt);
//knownsec// Update the total amount

        emit PublicOffering(msg.sender, amountOfUsdt);

        _whiteList[msg.sender] = 1; //knownsec// Automatically become a whitelisted user after
participating in the super fundraising

        return true;
    }
```

Recommendation: nothing.

3.5. IDO contract withdrawPriIDO function【PASS】

Audit analysis: The withdrawPriIDO function of the IDO contract is used to withdraw the private USDT, which can only be performed by the owner.

```
      //knownsec// Withdraw private placement USDT
    function withdrawPriIDO() onlyOwner external {
        //knownsec// Withdrawal restrictions
        require(block.number > endHeightOfPriIDO, 'YouSwap:
BLOCK_HEIGHT_NOT_REACHED');
        require(!_priIDOWithdrawFinished, 'YouSwap: PRI_IDO_WITHDRAWN_ALREADY');
//knownsec// Can only be withdrawn once

        _transfer(_usdtToken, _vault, totalUsdtAmountOfPriIDO); //knownsec// Transfer all
private USDT in the contract to the designated address
        _priIDOWithdrawFinished = true;
    }
```

Recommendation： nothing.

3.6. IDO contract withdrawPubIDO function【PASS】

Audit analysis: The IDO contract's withdrawPubIDO function is used to withdraw over-funded USDT, which can only be performed by the owner.

```
//knownsec// Withdraw super funded USDT
    function withdrawPubIDO() onlyOwner external {
        //knownsec// Withdrawal restrictions
        require(block.number        >        endHeightOfPubIDO,        'YouSwap:
BLOCK_HEIGHT_NOT_REACHED');
        require(!_pubIDOWithdrawFinished,                              'YouSwap:
PUB_IDO_WITHDRAWN_ALREADY'); //knownsec// Can only be withdrawn once

        uint256 amountToWithdraw = totalUsdtAmountOfPubIDO;
        //knownsec// If the amount of participation exceeds the target amount, only the target
over-raised amount can be withdrawn
        if (totalUsdtAmountOfPubIDO > targetUsdtAmountOfPubIDO) {
            amountToWithdraw = targetUsdtAmountOfPubIDO;
        }

        _transfer(_usdtToken, _vault, amountToWithdraw); //knownsec// Transfer all over-raised
USDT in the contract to the designated address
    }
```

Recommendation： nothing.

3.7. TokenYou contract minting function【PASS】

Audit analysis: The minting function of the TokenYou contract is used to mint You tokens to the specified address, and only minter permission users can perform this operation.

Recommendation： nothing.

3.8. TokenYou contract destruction function【PASS】

Audit analysis: The TokenYou contract destruction function is used to destroy a certain number of You tokens in the account.

Recommendation： nothing.

3.9. TokenYou contract minting authority management function【PASS】

Audit analysis: TokenYou contract minting permission is used to manage minting functions, only the owner can add and remove minter operations.

Recommendation： nothing.

3.10. Basic token functions of TokenYou contract【PASS】

Audit analysis: The basic functions of TokenYou contract comply with the basic functions of ERC20 tokens.

```
    /**
 *Submitted for verification at Etherscan.io on 2021-03-14
*/


//SPDX-License-Identifier: SimPL-2.0
pragma solidity ^0.6.0;


library SafeMath {
    /**
```

```
  * @dev Returns the addition of two unsigned integers, reverting on
  * overflow.
  *
  * Counterpart to Solidity's `+` operator.
  *
  * Requirements:
  *
  * - Addition cannot overflow.
  */
function add(uint256 a, uint256 b) internal pure returns (uint256) {
       uint256 c = a + b;
       require(c >= a, "SafeMath: addition overflow");

       return c;
}

/**
  * @dev Returns the subtraction of two unsigned integers, reverting on
  * overflow (when the result is negative).
  *
  * Counterpart to Solidity's `-` operator.
  *
  * Requirements:
  *
  * - Subtraction cannot overflow.
  */
function sub(uint256 a, uint256 b) internal pure returns (uint256) {
       return sub(a, b, "SafeMath: subtraction overflow");
}

/**
  * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
  * overflow (when the result is negative).
```

```
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message
on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be 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;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns
(uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

contract Ownable {

    address private _owner;

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

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), msg.sender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }
    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(_owner == msg.sender, "YouSwap: CALLER_IS_NOT_THE_OWNER");
        _;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0); //knownsec// Give up owner permissions
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "YouSwap: NEW_OWNER_IS_THE_ZERO_ADDRESS");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner; //knownsec// Transfer owner permissions to newOwner
    }
}
```

```
interface ITokenYou {
    /**
    * @dev Returns the amount of tokens in existence.
    */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns
(bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
```

```
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);

    event MaxSupplyChanged(uint256 oldValue,uint256 newValue);

    function resetMaxSupply(uint256 newValue) external;

    function mint(address recipient, uint256 amount) external;

    function burn(uint256 amount) external;

    function burnFrom(address account, uint256 amount) external;
}

contract TokenYou is Ownable,ITokenYou {
    using SafeMath for uint256;

    string private constant _name = 'YouSwap';
    string private constant _symbol = 'YOU';
    uint8 private constant _decimals = 6; //knownsec// The default token precision is 6
    uint256 private _totalSupply = 0;
    uint256 private _transfers = 0; //knownsec// Store the number of token transfers
    uint256 private _holders = 0; //knownsec// Number of storage token holders
    uint256 private _maxSupply = 2*10**14;
    mapping(address => uint256) private _balanceOf;
    mapping(address => mapping(address => uint256)) private _allowances;

    mapping(address => uint8) private _minters;

    constructor() public {
}

/**
  * @dev Returns the name of the token.
  */
function name() public pure returns (string memory) {
    return _name;
}

/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
function symbol() public pure returns (string memory) {
    return _symbol;
}

/**
 * @dev Returns the number of decimals used to get its user representation.
 * For example, if `decimals` equals `2`, a balance of `505` tokens should
 * be displayed to a user as `5,05` (`505 / 10 ** 2`).
 *
 * Tokens usually opt for a value of 18, imitating the relationship between
 * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
 * called.
 *
 * NOTE: This information is only used for _display_ purposes: it in
 * no way affects any of the arithmetic of the contract, including
 * {ITokenYou-balanceOf} and {ITokenYou-transfer}.
 */
function decimals() public pure returns (uint8) {
    return _decimals;
}
```

```
/**
  * @dev See {ITokenYou-totalSupply}.
  */
function totalSupply() public view override returns (uint256) {
        return _totalSupply;
}

/**
* @dev See {ITokenYou-maxSupply}.
*/
function maxSupply() public view returns (uint256) {
        return _maxSupply;
}

function transfers() public view returns (uint256) {
        return _transfers;
}

function holders() public view returns (uint256) {
        return _holders;
}

/**
  * @dev See {ITokenYou-balanceOf}.
  */
function balanceOf(address account) public view override returns (uint256) {
        return _balanceOf[account];
}

/**
  * @dev See {ITokenYou-transfer}.
  *
```

```
     * Requirements:
     *
     * - `recipient` cannot be the zero address.
     * - the caller must have a balance of at least `amount`.
     */
    function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(msg.sender, recipient, amount);
        return true;
    }

    /**
     * @dev See {ITokenYou-allowance}.
     */
    function allowance(address owner, address spender) public view virtual override returns
(uint256) {
        return _allowances[owner][spender];
    }

    /**
     * @dev See {ITokenYou-approve}.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public override returns (bool) {
        _approve(msg.sender, spender, amount);
        return true;
    }

    /**
     * @dev See {ITokenYou-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20}.
     *
     * Requirements:
     *
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public override
returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(sender, msg.sender, _allowances[sender][msg.sender].sub(amount, "YouSwap:
TRANSFER_AMOUNT_EXCEEDS_ALLOWANCE"));
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {ITokenYou-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public returns (bool) {
        _approve(msg.sender, spender, _allowances[msg.sender][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {ITokenYou-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public    returns (bool)
{
        _approve(msg.sender, spender, _allowances[msg.sender][spender].sub(subtractedValue,
"YouSwap: DECREASED_ALLOWANCE_BELOW_ZERO"));
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal {
        //knownsec// Zero address check
        require(sender != address(0), "YouSwap: TRANSFER_FROM_THE_ZERO_ADDRESS");
        require(recipient != address(0), "YouSwap: TRANSFER_TO_THE_ZERO_ADDRESS");
        require(amount > 0, "YouSwap: TRANSFER_ZERO_AMOUNT"); //knownsec// Limit on the number of transfers

        if(_balanceOf[recipient] == 0) _holders++; //knownsec// Update the number of token holders when the token balance of the receiving address is empty
        //knownsec// Update the token balance of both parties in the transfer
        _balanceOf[sender] = _balanceOf[sender].sub(amount, "YouSwap: TRANSFER_AMOUNT_EXCEEDS_BALANCE");
        _balanceOf[recipient] = _balanceOf[recipient].add(amount);

        _transfers ++; //knownsec// Update the number of transfers

        if(_balanceOf[sender]==0) _holders--; //knownsec// Reduce the number of token holders when the token balance of the sender address is empty after the transfer

        emit Transfer(sender, recipient, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements:
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal {
        require(account != address(0), "YouSwap: BURN_FROM_THE_ZERO_ADDRESS");
//knownsec// Zero address check

        _balanceOf[account] = _balanceOf[account].sub(amount, "YouSwap:
BURN_AMOUNT_EXCEEDS_BALANCE");
        if(_balanceOf[account]==0) _holders --; //knownsec// Reduce the number of token
holders when the token balance of the destroyed address is empty after the token is destroyed
        _totalSupply = _totalSupply.sub(amount); //knownsec// Update the total amount of tokens
        _transfers++; //knownsec// Increase the number of transfers
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner` s tokens.
     *
     * This internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 amount) internal {
        //knownsec// Zero address check
        require(owner != address(0), "YouSwap: APPROVE_FROM_THE_ZERO_ADDRESS");
        require(spender != address(0), "YouSwap: APPROVE_TO_THE_ZERO_ADDRESS");

        _allowances[owner][spender] = amount; //knownsec// Modify authorization value
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Destroys `amount` tokens from the caller.
     *
     * See {TokenYou-_burn}.
     */
    function burn(uint256 amount) external override{
        _burn(msg.sender, amount); //knownsec// Destroy a certain number of tokens of the caller
    }

    /**
     * @dev Destroys `amount` tokens from `account`, deducting from the caller's
     * allowance.
     *
     * See {TokenYou-_burn} and {TokenYou-allowance}.
     *
     * Requirements:
     *
     * - the caller must have allowance for ``accounts``'s tokens of at least
     * `amount`.
     */
    function burnFrom(address account, uint256 amount) external override{
        uint256 decreasedAllowance = allowance(account, msg.sender).sub(amount, "YouSwap:
BURN_AMOUNT_EXCEEDS_ALLOWANCE");

        _approve(account, msg.sender, decreasedAllowance); //knownsec// Modify authorization
value
```

```
        _burn(account, amount); //knownsec// Destroy a certain number of tokens from authorized addresses
    }
    //knownsec// Restrict only users with minting authority to mint coins
    modifier isMinter() {
        require(_minters[msg.sender]==1, "YouSwap: IS_NOT_A_MINTER");
        _;
    }

    function isContract(address account) internal view returns (bool) {
        uint256 size;
        assembly { size := extcodesize(account) }
        return size > 0;
    }

    function mint(address recipient, uint256 amount) external override isMinter{
        require(_totalSupply.add(amount) <= _maxSupply, 'YouSwap: EXCEEDS_MAX_SUPPLY'); //knownsec// Minting limit, the total amount of tokens after minting cannot exceed the maximum number of minted coins
        _totalSupply = _totalSupply.add(amount); //knownsec// Update the total amount of tokens

        if(_balanceOf[recipient] == 0) _holders++; //knownsec// Update the number of token holders when the minting address balance is empty
        _balanceOf[recipient] = _balanceOf[recipient].add(amount); //knownsec// Modify the coin balance of the coin address

        _transfers++; //knownsec// Increase the number of transfers
        emit Transfer(address(0), recipient, amount);
    }

    function addMinter(address account) external onlyOwner{
        require(isContract(account), "YouSwap: MUST_BE_A_CONTRACT_ADDRESS");
        _minters[account] = 1; //knownsec// Grant the specified address minting authority
    }

    function removeMinter(address account) external onlyOwner{
        _minters[account] = 0; //knownsec// Revoke the minting authority of the specified address
    }

    function resetMaxSupply(uint256 newValue) external override onlyOwner{
        require(newValue > _totalSupply && newValue< _maxSupply, 'YouSwap: NOT_ALLOWED'); //knownsec// This token can be minted; through owner control, the upper limit of minting can be modified (can only be adjusted down)
        emit MaxSupplyChanged(_maxSupply,newValue);
        _maxSupply = newValue; //knownsec// Change the maximum number of coins
    }
}
```

Recommendation： nothing.

4. Basic code vulnerability detection

4.1. Compiler version security【PASS】

Check whether a safe compiler version is used in the contract code implementation.

Audit result: After testing, the IDO contract compiler version is specified in the smart contract code as 0.5.16, TokenYou

The contract compiler version is above 0.6.0, and this security issue does not exist.

Recommendation: nothing.

4.2. Redundant code【PASS】

Check whether the contract code implementation contains redundant code.

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

Recommendation: nothing.

4.3. Use of safe arithmetic library【PASS】

Check whether the SafeMath safe arithmetic library is used in the contract code implementation.

Audit result: After testing, the SafeMath safe arithmetic library has been used in the smart contract code, and there is no such security problem.

Recommendation: nothing.

4.4. Not recommended encoding【PASS】

Check whether there is an encoding method that is not officially recommended or abandoned in the contract code implementation

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

Recommendation: nothing.

4.5. Reasonable use of require/assert【PASS】

Check the rationality of the use of require and assert statements in the contract code implementation.

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

Recommendation: nothing.

4.6. Fallback function safety【PASS】

Check whether the fallback function is used correctly in the contract code implementation.

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

Recommendation: nothing.

4.7. tx.origin authentication【PASS】

tx.origin is a global variable of Solidity that traverses 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 attacks like phishing.

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

Recommendation: nothing.

4.8. Owner permission control【PASS】

Check whether the owner in the contract code implementation has excessive authority. For example, arbitrarily modify other account balances, etc.

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

Recommendation: nothing.

4.9. Gas consumption detection【PASS】

Check whether the consumption of gas exceeds the maximum block limit.

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

Recommendation: nothing.

4.10. call injection attack【PASS】

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

Audit result: After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation: nothing.

4.11. Low-level function safety【PASS】

Check whether there are security vulnerabilities in the use of low-level functions (call/delegatecall) in the contract code implementation

The execution context of the call function is in the called contract; the execution context of the delegatecall function is in the contract that currently calls the function.

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

Recommendation: nothing.

4.12. Vulnerability of additional token issuance【PASS】

Check whether there is a function that may increase the total amount of tokens in the token contract after initializing the total amount of tokens.

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

Recommendation: nothing.

4.13. Access control defect detection【PASS】

Different functions in the contract should set reasonable permissions.

Check whether each function in the contract correctly uses keywords such as public and private for visibility modification, check whether the contract is correctly defined and use modifier to restrict access to key functions to avoid problems caused by unauthorized access.

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

Recommendation: nothing.

4.14. Numerical overflow detection【PASS】

The arithmetic problems in smart contracts refer to integer overflow and integer underflow.

Solidity can handle up to 256-bit numbers (2^256-1). If the maximum number increases by 1, it will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum digital value.

Integer overflow and underflow are not a new type of vulnerability, but they are especially dangerous in smart contracts. Overflow conditions can lead to incorrect results, especially if the possibility is not expected, which may affect the reliability and safety of the program.

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

Recommendation: nothing.

4.15. Arithmetic accuracy error【PASS】

As a programming language, Solidity has data structure design similar to ordinary programming languages, such as variables, constants, functions, arrays, functions, structures, etc. There is also a big difference between Solidity and ordinary programming languages-Solidity does not float Point type, and all the numerical calculation results of Solidity will only be integers, there will be no decimals, and it is not allowed to define decimal type data. Numerical calculations in the contract are indispensable, and the design of numerical calculations may cause relative errors. For example, the same level of calculations: 5/2*10=20, and 5*10/2=25, resulting in errors, which are larger in data The error will be larger and more obvious.

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

Recommendation: nothing.

4.16. Incorrect use of random numbers【PASS】

Smart contracts may need to use random numbers. Although the functions and variables provided by Solidity can access values that are obviously unpredictable, such as block.number and block.timestamp, they are usually more public than they appear 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.

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

Recommendation： nothing.

4.17. Unsafe interface usage【PASS】

Check whether unsafe interfaces are used in the contract code implementation.

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

Recommendation： nothing.

4.18. Variable coverage【PASS】

Check whether there are security issues caused by variable coverage in the contract code implementation.

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

Recommendation: nothing.

4.19. Uninitialized storage pointer【PASS】

In solidity, a special data structure is allowed to be a struct structure, and the local variables in the function are stored in storage or memory by default.

The existence of storage (memory) and memory (memory) are two different concepts. Solidity allows pointers to point to an uninitialized reference, while uninitialized local storage will cause variables to point to other storage variables, leading to variable coverage, or even more serious As a consequence, you should avoid initializing struct variables in functions during development.

Audit result: After testing, the smart contract code does not use structure, there is no such problem.

Recommendation: nothing.

4.20. Return value call verification【PASS】

This problem mostly occurs in smart contracts related to currency transfer, so it is also called silent failed delivery or unchecked delivery.

In Solidity, there are transfer(), send(), call.value() and other currency transfer methods, which can all be used to send ETH to an address. The difference is: When the transfer fails, it will be thrown and the state will be rolled back; Only 2300gas will be passed for calling to prevent reentry attacks; false will be returned when send fails; 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 passed for calling (can be Limit by passing in gas_value parameters), which 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 lead to unexpected results due to ETH sending failure.

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

Recommendation: nothing.

4.21. Transaction order dependency【PASS】

Since miners always get gas fees through codes that represent externally owned addresses (EOA), users can specify higher fees 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.

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

4.22. Timestamp dependency attack【PASS】

The timestamp of the data block usually uses the local time of the miner, and this time can fluctuate in the range of about 900 seconds. When other nodes accept a new block, it only needs to verify whether the timestamp is later than the previous block and The error with local time is within 900 seconds. A miner can profit from it by setting the timestamp of the block to satisfy the conditions that are beneficial to him as much as possible.

Check whether there are key functions that depend on the timestamp in the contract code implementation.

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

Recommendation: nothing.

4.23. Denial of service attack【PASS】

In the world of Ethereum, denial of service is fatal, and a smart contract that has suffered 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 as the transaction recipient, artificially increasing the gas required for computing functions to cause gas exhaustion, abusing access control to access the private component of the smart contract, using confusion and negligence, etc. Wait.

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

Recommendation: nothing.

4.24. Fake recharge vulnerability【PASS】

The transfer function of the token contract uses the if judgment method to check the balance of the transfer initiator (msg.sender). When balances[msg.sender] <value, enter the else logic part and return false, and finally no exception is thrown. We believe that only if/else this kind of gentle judgment method is an imprecise coding method in sensitive function scenarios such as transfer.

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

Recommendation: nothing.

4.25. Reentry attack detection【PASS】

The **call.value()** function in Solidity consumes all the gas it receives when it is used to send ETH. When the **call.value()** function to send ETH occurs before the actual reduction of the sender's account balance, There is a risk of reentry attacks.

Audit results: After auditing, the vulnerability does not exist in the smart contract code.

Recommendation: nothing.

4.26. Replay attack detection【PASS】

If the contract involves the need for entrusted management, attention should be paid to the non-reusability of verification to avoid replay attacks

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

Audit results: After testing, the smart contract does not use the call function, and this vulnerability does not exist.

Recommendation: nothing.

4.27. Rearrangement attack detection【PASS】

A rearrangement attack refers to a miner or other party trying to "compete" with smart contract participants by inserting their own information into a list or mapping, so that the attacker has the opportunity to store their own information in the contract. in.

Audit results: After auditing, the vulnerability does not exist in the smart contract code.

Recommendation: nothing.

5. Appendix A：Contract code

Source code：

TokenYou

https://etherscan.io/address/0x1d32916cfa6534d261ad53e2498ab95505bd2510#code

IDO

https://etherscan.io/address/0x59e6639bef44164ba44aa58771b94df30bc023a6#code

6. Appendix B: Vulnerability rating standard

Smart contract vulnerability rating standards	
Level	Level Description
High	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.; 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.; 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.
Medium	High-risk vulnerabilities that require specific addresses to trigger, such as value overflow vulnerabilities that can be triggered by token contract owners; access control defects for non-critical functions, and logical design defects that cannot cause direct capital losses, etc.
Low	Vulnerabilities that are difficult to be triggered, vulnerabilities with limited damage after triggering, such as value overflow vulnerabilities that require a large amount of ETH or tokens to trigger, vulnerabilities where attackers cannot

	directly profit after triggering value overflow, and the transaction sequence triggered by specifying high gas depends on the risk Wait.

7. Appendix C：Introduction to auditing tools

7.1 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, Bit of Traits of Bits visual disassembler for EVM bytecode, used for visual analysis. Like binary files, Manticore provides a simple command line interface and a Python for analyzing EVM bytecode API.

7.2 Oyente

Oyente is a smart contract analysis tool. Oyente can be used to detect common bugs in smart contracts, such as reentrancy, transaction sequencing dependencies, etc. More convenient, Oyente's design is modular, so this allows advanced users to implement and Insert their own detection logic to check the custom attributes in their contract.

7.3 securify.sh

Securify can verify common security issues of Ethereum smart contracts, such as disordered transactions and lack of input verification. It analyzes all possible execution paths of the program while fully automated. In addition, Securify also has a specific language for specifying vulnerabilities, which makes Securify can keep an eye on current security and other reliability issues at any time.

7.4 Echidna

Echidna is a Haskell library designed for fuzzing EVM code.

7.5 MAIAN

MAIAN is an automated tool for finding vulnerabilities in Ethereum smart contracts. Maian processes the bytecode of the contract and tries to establish a series of transactions to find and confirm the error.

7.6 ethersplay

ethersplay is an EVM disassembler, which contains relevant analysis tools.

7.7 ida-evm

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

7.8 Remix-ide

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

7.9 Knownsec Penetration Tester Special Toolkit

Pen-Tester tools collection is created by KnownSec team. It contains plenty of Pen-Testing tools such as automatic testing tool, scripting tool, Self-developed tools etc.

KNOWNSEC

Beijing KnownSec Information Technology Co., Ltd.

Advisory telephone	+86(10)400 060 9587
E-mail	sec@knownsec.com
Website	www.knownsec.com
Address	wangjing soho T2-B2509,Chaoyang District, Beijing