KNOWNSEC

# Smart Contract Security Audit Report

Audit Results

PASS

★ ★ ★ ★ ★

## Version description

| Revised man | Revised content | Revised time | version | Reviewer |
|---|---|---|---|---|
| Yifeng Luo | Document creation and editing | 2020/11/18 | V1.0 | Haojie Xu |

## Document information

| Document Name | Audit Date | Audit results | Privacy level | Audit enquiry telephone |
|---|---|---|---|---|
| titanswap Smart Contract Security Audit Report | 2020/11/18 | PASS | Open project team | +86 400-060-9587 |

## Copyright statement

## Company statement

Beijing Knownsec Information Technology Co., Ltd. only conducts the agreed safety audit and issued the report based on the documents and materials provided to us by the project party as of the time of this report. We cannot judge the background, the practicality of the project, the compliance of business model and the legality of the project , and will not be responsible for this. The report is for reference only for internal decision-making of the project party. Without our written consent, the report shall not be disclosed or provided to other people or used for other purposes without permission. The report issued by us shall not be used as the basis for any behavior and decision of the third party. We shall not bear any responsibility for the consequences of the relevant decisions adopted by the user and the third party. We assume that as of the time of this report, the project has provided us with no information missing, tampered with, deleted or concealed. If the information provided is missing, tampered, deleted, concealed or reflected in a situation inconsistent with the actual situation, we will not be liable for the loss and adverse impact caused thereby.

# Catalog

# 1. Review

The effective testing time of this report is from November 13, 2020 to November 18, 2020. During this period, the Knownsec engineers audited the safety and regulatory aspects of titanswap smart contract code.

In this test, engineers comprehensively analyzed common vulnerabilities of smart contracts (Chapter 3) and It was not discovered medium-risk or high-risk vulnerability,so it's evaluated as **pass**.

**The result of the safety auditing: Pass**

Since the test process is carried out in a non-production environment, all the codes are the latest backups. We communicates with the relevant interface personnel, and the relevant test operations are performed under the controllable operation risk to avoid the risks during the test..

Target information for this test:

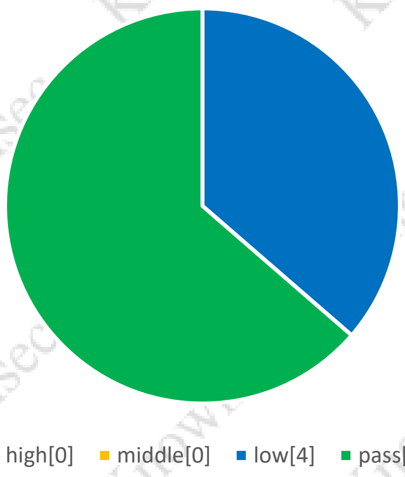
| Project name | Project content |
|---|---|
| Token name | titanswap |
| Code type | Token code |
| Code language | Solidity |
| Code address | https://github.com/titanswapOfficial/titanswap |

# 2. Analysis of code vulnerability

## 2.1. Distribution of vulnerability Levels

| Vulnerability statistics | | | |
|---|---|---|---|
| high | Middle | low | pass |
| 0 | 0 | 4 | 7 |

Distribution Chart



## 2.2. Audit result summary

Other unknown security vulnerabilities are not included in the scope of this audit.

| Result | | | |
|---|---|---|---|
| Test project | Test content | status | description |
| Smart Contract Security Audit | Reentrancy | Low risk | Check the call.value() function for security |
| | Arithmetic Issues | Pass | Check add and sub functions |
| | Access Control | Pass | Check the operation access control |
| | Unchecked Return Values For Low Level Calls | Pass | Check the currency conversion method. |
| | Bad Randomness | Pass | Check the unified content filter |
| | Transaction ordering dependence | Low risk | Check the transaction ordering dependence |
| | Denial of service attack detection | Low risk | Check whether the code has a resource abuse problem when using a resource |
| | Logic design Flaw | Pass | Examine the security issues associated with business design in intelligent contract codes. |
| | USDT Fake Deposit Issue | Pass | Check for the existence of USDT Fake Deposit Issue |
| | Adding tokens | Low risk | It is detected whether there is a function in the token contract that may increase the total amounts of tokens |
| | Freezing accounts bypassed | Pass | It is detected whether there is an unverified token source account, an originating account, and whether the target account is frozen. |

# 3. Result analysis

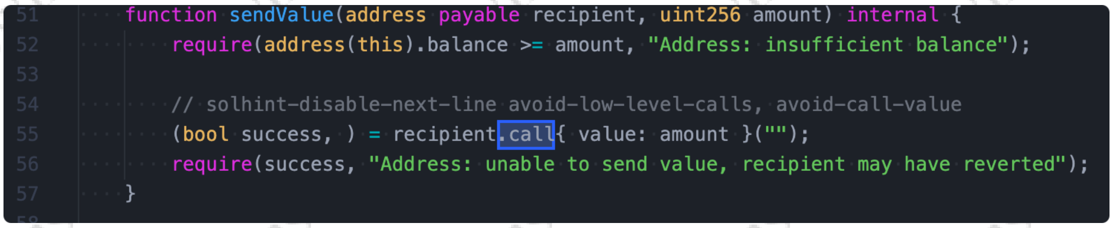
## 3.1. Reentrancy 【Low risk】

The Reentrancy attack, probably the most famous Blockchain vulnerability， led to a hard fork of Ethereum.

When the low level call() function sends tokens to the msg.sender address, it becomes vulnerable; if the address is a smart token, the payment will trigger its fallback function with what's left of the transaction gas.

**Test results：** there are related vulnerabilities in the smart contract code:

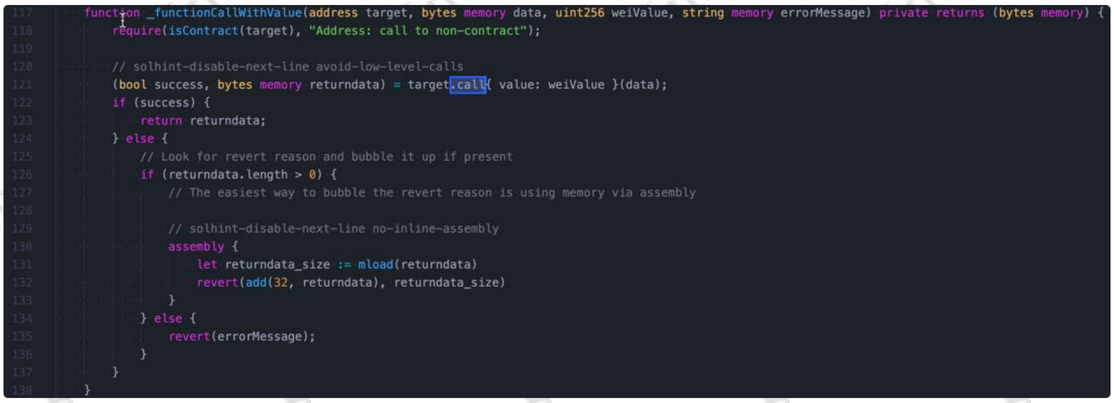
contracts/libraries/Address.sol



```
    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");
    }
```

contracts/libraries/Address.sol



```
    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);
            }
        }
    }
```

**Safety advice：**

(1) Try to use send() and transfer() functions.

(2) If you use a low-level calling function like the call() function, you should perform the internal state change first, and then use the low-level calling function.

(3) Try to avoid calling external contracts when writing smart contracts.

## 3.2. Arithmetic Issues【Pass】

Also known as integer overflow and integer underflow. Solidity can handle up to 256 digits (2^256-1), The largest number increases by 1 will overflow to 0. Similarly, when the number is an unsigned type, 0 minus 1 will underflow to get the maximum numeric value.

Integer overflows and underflows are not a new class of vulnerability, but they are especially dangerous in smart contracts. Overflow can lead to incorrect results, especially if the probability is not expected, which may affect the reliability and security of the program.

**Test results**: No related vulnerabilities in smart contract code.

**Safety advice**：None.

## 3.3. Access Control【Pass】

Access Control issues are common in all programs,Also smart contracts. The famous Parity Wallet smart contract has been affected by this issue.

**Test results**: No related vulnerabilities in smart contract code.

**Safety advice**：None.

## 3.4. Unchecked Return Values For Low Level Calls【Pass】

Also known as or related to silent failing sends, unchecked-send. There are transfer methods such as transfer(), send(), and call.value() in Solidity and can be used to send tokens s to an address. The difference is: transfer will be thrown when failed to send, and rollback; only 2300gas will be passed for call to prevent reentry attacks; send will return false if send fails; only 2300gas will be passed for call to prevent reentry attacks; If .value fails to send, it will return false; passing all available gas calls (which can be restricted by passing in the gas_value parameter) cannot effectively prevent reentry attacks.

If the return value of the send and call.value switch functions is not been checked in the code, the contract will continue to execute the following code,and it may have caused unexpected results due to tokens sending failure.

**Test results**: No related vulnerabilities in smart contract code.

**Safety advice**: None.

## 3.5. Bad Randomness【Pass】

Smart Contract May Need to Use Random Numbers. While Solidity offers functions and variables that can access apparently hard-to-predict values just as block.number and block.timestamp. they are generally either more public than they seem or subject to miners' influence. Because these sources of randomness are to an extent predictable, malicious users can generally replicate it and attack the function relying on its unpredictablility.

**Test results**: No related vulnerabilities in smart contract code.

**Safety advice**: None.

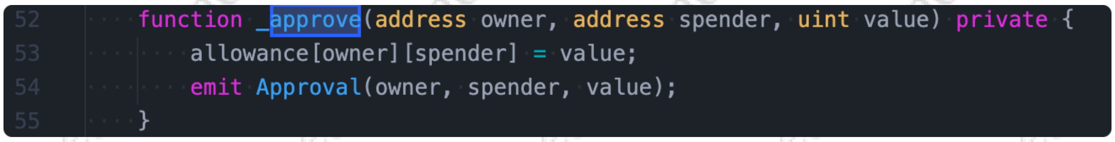
## 3.6. Transaction ordering dependence【Low risk】

Since miners always get rewarded via gas fees for running code on behalf of externally owned addresses (EOA), users can specify higher fees to have their transactions mined more quickly. Since the blockchain is public, everyone can see the contents of others' pending transactions.

This means if a given user is revealing the solution to a puzzle or other valuable secret, a malicious user can steal the solution and copy their transaction with higher fees to preempt the original solution.

**Test results**: Having related vulnerabilities in smart contract code.

contracts/TitanSwapV1ERC20.sol



```
    function _approve(address owner, address spender, uint value) private {
        allowance[owner][spender] = value;
        emit Approval(owner, spender, value);
    }
```

**Safety advice**:

1. User A allows the number of user B transfers to be N (N > 0) by calling the approve function;

2. After a while, user A decided to change N to M (M > 0), so he called the approve function again;

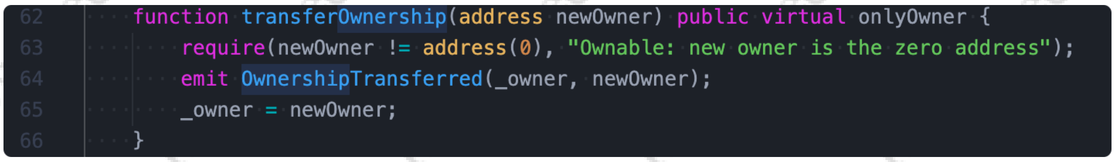
3. User B quickly calls the transfer from function to transfer the number of N before the second call is processed by the miner. After user A's second call to approve is successful, user B can get the transfer amount of M again. That is, user B obtains the transfer amount of N+M by trading sequence attack.

## 3.7. Denial of service attack detection 【Low risk】

In the blockchain world, denial of service is deadly, and smart contracts under attack of this type may never be able to return to normal. There may be a number of reasons for a denial of service in smart contracts, including malicious behavior as a recipient of transactions, gas depletion caused by artificially increased computing gas, and abuse of access control to access the private components of the intelligent contract. Take advantage of confusion and neglect, etc.

**Test results**: After testing, there is an error in the smart contract code because of the user's owner access control strategy, which will cause the user to permanently lose control.

contracts/interfaces/Ownable.sol



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

**Safety advice**: For the conversion of control authority, attention should be paid to the determination of user ownership to avoid permanent loss of control.

## 3.8. Logical design Flaw 【Pass】

Detect the security problems related to business design in the contract code.

**Test results**: No related vulnerabilities in smart contract code.

**Safety advice**: None.

## 3.9. USDT Fake Deposit Issue【Pass】

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

**Detection results**: No related vulnerabilities in smart contract code.

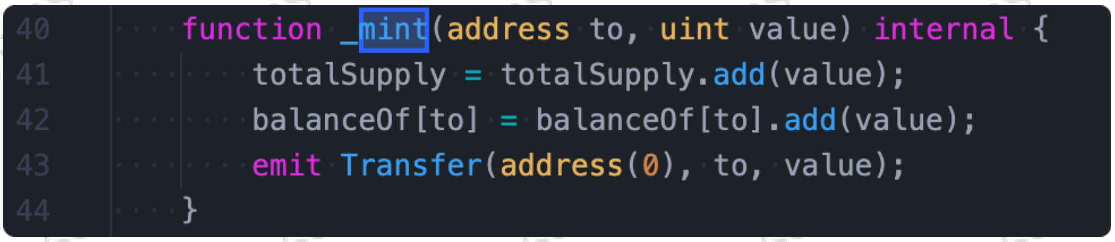
**Safety advice:** None.

## 3.10. Adding tokens【Low risk】

It is detected whether there is a function in the token contract that may increase the total amount of tokens after the total amount of tokens is initialized.

**Test results**: Having related vulnerabilities in smart contract code.

contracts/TitanSwapV1ERC20.sol


```
    function _mint(address to, uint value) internal {
        totalSupply = totalSupply.add(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(address(0), to, value);
    }
```

**Safety advice**:

This problem is not a security problem, but some exchanges will limit the use of the additional issue function, and the specific situation needs to be determined according to the requirements of the exchange.

## 3.11. Freezing accounts bypassed【Pass】

In the token contract, when transferring the token, it is detected whether there is an unverified token source account, an originating account, and whether the target account is frozen.

**Detection results:** No related vulnerabilities in smart contract code.

**Safety advice:** None.

# 4. Appendix A：Contract code

```
// contracts/TitanSwapV1ERC20.sol
pragma solidity =0.6.12;

import "./libraries/SafeMath.sol";
import "./interfaces/ITitanSwapV1ERC20.sol";

contract TitanSwapV1ERC20 {
    using SafeMath for uint;

    string public constant name = 'TitanSwap V1';
    string public constant symbol = 'Titan-V1';
    uint8 public constant decimals = 18;
    uint  public totalSupply;
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint)) public allowance;

    bytes32 public DOMAIN_SEPARATOR;
    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256
deadline)");
    bytes32 public constant PERMIT_TYPEHASH =
0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;
    mapping(address => uint) public nonces;

    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    constructor() public {
        uint chainId;
        assembly {
            chainId := chainid()
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256('EIP712Domain(string name,string version,uint256 chainId,address
verifyingContract)'),
                keccak256(bytes(name)),
                keccak256(bytes('1')),
                chainId,
                address(this)
            )
        );
    }

    function _mint(address to, uint value) internal {
        totalSupply = totalSupply.add(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(address(0), to, value);
    }

    function _burn(address from, uint value) internal {
        balanceOf[from] = balanceOf[from].sub(value);
        totalSupply = totalSupply.sub(value);
        emit Transfer(from, address(0), value);
    }

    function _approve(address owner, address spender, uint value) private {
        allowance[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    function _transfer(address from, address to, uint value) private {
        balanceOf[from] = balanceOf[from].sub(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(from, to, value);
    }

    function approve(address spender, uint value) external returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }

    function transfer(address to, uint value) external returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
    }

    function transferFrom(address from, address to, uint value) external returns (bool) {
        if (allowance[from][msg.sender] != uint(-1)) {
            allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);
        }
        _transfer(from, to, value);
        return true;
    }

    function permit(address owner, address spender, uint value, uint deadline, uint8 v,
bytes32 r, bytes32 s) external {
        require(deadline >= block.timestamp, 'TitanSwapV1: EXPIRED');
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value,
nonces[owner]++, deadline))
            )
        );
        address recoveredAddress = ecrecover(digest, v, r, s);
        require(recoveredAddress != address(0) && recoveredAddress == owner, 'UniswapV2:
INVALID_SIGNATURE');
        _approve(owner, spender, value);
    }


}

// contracts/TitanSwapV1Factory.sol
pragma solidity =0.6.12;

import "./interfaces/ITitanSwapV1Factory.sol";
import './TitanSwapV1Pair.sol';

contract TitanSwapV1Factory is ITitanSwapV1Factory {
    address public override feeTo;
    address public override feeToSetter;

    mapping(address => mapping(address => address)) public override getPair;
    address[] public override allPairs;

    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
    }

    function allPairsLength() external override view returns (uint) {
        return allPairs.length;
    }

    function createPair(address tokenA, address tokenB) external override returns (address
pair) {
        require(tokenA != tokenB, 'TitanSwapV1: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB,
tokenA);
        require(token0 != address(0), 'TitanSwapV1: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'TitanSwapV1: PAIR_EXISTS'); //
single check is sufficient
        bytes memory bytecode = type(TitanSwapV1Pair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        TitanSwapV1Pair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    function setFeeTo(address _feeTo) external override {
        require(msg.sender == feeToSetter, 'TitanSwapV1: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external override {
        require(msg.sender == feeToSetter, 'TitanSwapV1: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }
}

// contracts/TitanSwapV1LimitOrder.sol
pragma solidity =0.6.12;

import "./interfaces/IWETH.sol";
import "./interfaces/IERC20.sol";
import "./interfaces/ITitanSwapV1Router01.sol";
import "./interfaces/ITitanSwapV1Pair.sol";
import "./libraries/TransferHelper.sol";
import "./libraries/SafeMath.sol";
import "./libraries/TitanSwapV1Library.sol";

interface ITitanSwapV1LimitOrder {
    // event Transfer(address indexed from, address indexed to, uint value);
     event Deposit(uint orderId,address indexed pair,address indexed user,uint
amountIn,uint amountOut,uint fee);

     function setDepositAccount(address) external;
     function depositExactTokenForTokenOrder(address sellToken,address pair,uint
amountIn,uint amountOut) external payable;
     // deposit swapExactEthForTokens
     function depositExactEthForTokenOrder(address pair,uint amountIn,uint amountOut)
external payable;
      // deposit swapExactTokenForETH
     function depositExactTokenForEth(address sellToken,address pair,uint amountIn,uint
amountOut) external payable;

     function cancelTokenOrder(uint orderId) external;


     function executeExactTokenForTokenOrder(uint orderId, address[] calldata path, uint
deadline) external;
     function executeExactETHForTokenOrder(uint orderId, address[] calldata path, uint
deadline) external payable;
     function executeExactTokenForETHOrder(uint orderId, address[] calldata path, uint
deadline) external;


     function queryOrder(uint orderId) external view
returns(address,address,uint,uint,uint);
     function existOrder(uint orderId) external view returns(bool);
     function withdrawFee(address payable to) external;
     function setEthFee(uint _ethFee) external;
}

contract TitanSwapV1LimitOrder is ITitanSwapV1LimitOrder {

    using SafeMath for uint;
    address public  depositAccount;
    address public immutable router;
    address public immutable  WETH;
     address public immutable factory;
    uint public balance;
    uint public userBalance;
    mapping (uint => Order) private depositOrders;
    // to deposit order count
    uint public orderCount;
    // total order count
    uint public orderIds;
    // eth fee,defualt 0.01 eth
    uint public ethFee = 10000000000000000;

    constructor(address _router,address _depositAccount,address _WETH,address
_factory,uint _ethFee) public{
        router = _router;
        depositAccount = _depositAccount;
        WETH = _WETH;
        factory = _factory;
        ethFee = _ethFee;
    }


    struct Order {
        bool exist;
        address pair;
        address payable user; // 用户地址
        address sellToken;
        // uint direct; // 0 或 1,默认根据pair的token地址升序排,0- token0, token1 1- token1 
token0
        uint amountIn;
        uint amountOut;
        uint ethValue;

    }

     function setDepositAccount(address _depositAccount)  external override{
        require(msg.sender == depositAccount, 'TitanSwapV1: FORBIDDEN');
        depositAccount = _depositAccount;
    }


    function depositExactTokenForTokenOrder(address sellToken,address pair,uint 
amountIn,uint amountOut) external override payable {
        // call swap method cost fee.
        uint fee = ethFee;
        require(msg.value >= fee,"TitanSwapV1 : no fee enough");
        orderIds = orderIds.add(1);
        uint _orderId = orderIds;
        // need transfer eth fee. need msg.sender send approve trx first.
        TransferHelper.safeTransferFrom(sellToken,msg.sender,address(this),amountIn);

        depositOrders[_orderId] = 
Order(true,pair,msg.sender,sellToken,amountIn,amountOut,msg.value);
        emit Deposit(_orderId,pair,msg.sender,amountIn,amountOut,msg.value);
        orderCount = orderCount.add(1);
        balance = balance.add(msg.value);
        userBalance = userBalance.add(msg.value);
    }

     function depositExactEthForTokenOrder(address pair,uint amountIn,uint amountOut) 
external override payable {
        uint fee = ethFee;
        uint calFee = msg.value.sub(amountIn);
        require(calFee >= fee,"TitanSwapV1 : no fee enough");

        orderIds = orderIds.add(1);
        uint _orderId = orderIds;

        depositOrders[_orderId] = 
Order(true,pair,msg.sender,address(0),amountIn,amountOut,msg.value);
        emit Deposit(_orderId,pair,msg.sender,amountIn,amountOut,msg.value);
        orderCount = orderCount.add(1);
        balance = balance.add(msg.value);
        userBalance = userBalance.add(msg.value);
     }

      function depositExactTokenForEth(address sellToken,address pair,uint amountIn,uint 
amountOut) external override payable {
        uint fee = ethFee;
        require(msg.value >= fee,"TitanSwapV1 : no fee enough");
        orderIds = orderIds.add(1);
        uint _orderId = orderIds;

         // need transfer eth fee. need msg.sender send approve trx first.
        TransferHelper.safeTransferFrom(sellToken,msg.sender,address(this),amountIn);
        depositOrders[_orderId] = 
Order(true,pair,msg.sender,sellToken,amountIn,amountOut,msg.value);
        emit Deposit(_orderId,pair,msg.sender,amountIn,amountOut,msg.value);
        orderCount = orderCount.add(1);
        balance = balance.add(msg.value);
        userBalance = userBalance.add(msg.value);
      }
    function cancelTokenOrder(uint orderId) external override {
        Order memory order = depositOrders[orderId];
        require(order.exist,"order not exist.");
        require(msg.sender == order.user,"no auth to cancel.");

        // revert eth
        TransferHelper.safeTransferETH(order.user,order.ethValue);

        if(order.sellToken != address(0)) {
            // revert token
            TransferHelper.safeTransfer(order.sellToken,order.user,order.amountIn);
        }

        userBalance = userBalance.sub(order.ethValue);
        balance = balance.sub(order.ethValue);

        delete(depositOrders[orderId]);
        orderCount = orderCount.sub(1);
    }

    function queryOrder(uint orderId) external override view 
returns(address,address,uint,uint,uint) {
        Order memory order = depositOrders[orderId];
        return (order.pair,order.user,order.amountIn,order.amountOut,order.ethValue);
    }

    function existOrder(uint orderId) external override view returns(bool) {
        return depositOrders[orderId].exist;
    }

     function executeExactTokenForTokenOrder(
        uint orderId,
        address[] calldata path,
        uint deadline
   ) external override {
       require(msg.sender == depositAccount, 'TitanSwapV1 executeOrder: FORBIDDEN');

       Order memory order = depositOrders[orderId];
       require(order.exist,"order not exist!");
       // approve to router
       TransferHelper.safeApprove(path[0],router,order.amountIn);


       delete(depositOrders[orderId]);
       orderCount = orderCount.sub(1);
       userBalance = userBalance.sub(order.ethValue);


ITitanSwapV1Router01(router).swapExactTokensForTokens(order.amountIn,order.amountOut,p
ath,order.user,deadline);
    }

     // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, address[] memory path, address _to) internal 
virtual {
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = UniswapV2Library.sortTokens(input, output);
            uint amountOut = amounts[i + 1];
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) : 
(amountOut, uint(0));
            address to = i < path.length - 2 ? UniswapV2Library.pairFor(factory, output, 
path[i + 2]) : _to;
            IUniswapV2Pair(UniswapV2Library.pairFor(factory, input, output)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }


    function executeExactETHForTokenOrder(uint orderId, address[] calldata path, uint 
deadline) external override payable {
         require(deadline >= block.timestamp, 'UniswapV2Router: EXPIRED');
        require(msg.sender == depositAccount, 'TitanSwapV1 executeOrder: FORBIDDEN');
        require(msg.value > 0, 'TitanSwapV1 executeOrder: NO ETH');
        Order memory order = depositOrders[orderId];
        require(order.exist,"order not exist!");
        delete(depositOrders[orderId]);
        orderCount = orderCount.sub(1);
        userBalance = userBalance.sub(order.ethValue);
        // call with msg.value = amountIn
        require(path[0] == WETH, 'UniswapV2Router: INVALID_PATH');
        uint[]  memory amounts = UniswapV2Library.getAmountsOut(factory, msg.value, path);
        require(amounts[amounts.length - 1] >= order.amountOut, 'UniswapV2Router: 
INSUFFICIENT_OUTPUT_AMOUNT');

        IWETH(WETH).deposit{value: msg.value}();
         assert(IWETH(WETH).transfer(order.pair, amounts[0]));
        _swap(amounts, path, order.user);

    }


    function executeExactTokenForETHOrder(uint orderId, address[] calldata path, uint 
deadline) external override {
         require(msg.sender == depositAccount, 'TitanSwapV1 executeOrder: FORBIDDEN');

        Order memory order = depositOrders[orderId];
        require(order.exist,"order not exist!");
        // approve to router
        TransferHelper.safeApprove(path[0],router,order.amountIn);
        delete(depositOrders[orderId]);
        orderCount = orderCount.sub(1);
        userBalance = userBalance.sub(order.ethValue);

ITitanSwapV1Router01(router).swapExactTokensForETH(order.amountIn,order.amountOut,path
,order.user,deadline);
    }


    function withdrawFee(address payable to) external override {
        require(msg.sender == depositAccount, 'TitanSwapV1 : FORBIDDEN');
        uint amount = balance.sub(userBalance);
        require(amount > 0,'TitanSwapV1 : amount = 0');
        TransferHelper.safeTransferETH(to,amount);
        balance = balance.sub(amount);
    }

    function setEthFee(uint _ethFee) external override {
        require(msg.sender == depositAccount, 'TitanSwapV1 : FORBIDDEN');
        require(_ethFee >= 10000000,'TitanSwapV1: fee wrong');
        ethFee = _ethFee;
    }


}

// contracts/TitanSwapV1Pair.sol
pragma solidity =0.6.12;

import "./interfaces/ITitanSwapV1Factory.sol";
import "./interfaces/ITitanSwapV1Callee.sol";
import "./interfaces/IERC20.sol";
import "./libraries/SafeMath.sol";
import "./libraries/Math.sol";
import "./libraries/UQ112x112.sol";
import "./TitanSwapV1ERC20.sol";

contract TitanSwapV1Pair is TitanSwapV1ERC20 {
    using SafeMath  for uint;
    using UQ112x112 for uint224;

    uint public constant MINIMUM_LIQUIDITY = 10**3;
    bytes4 private constant SELECTOR = 
bytes4(keccak256(bytes('transfer(address,uint256)')));

    address public factory;
    address public token0;
    address public token1;

    uint112 private reserve0;           // uses single storage slot, accessible via 
getReserves
```

```
    uint112 private reserve1;           // uses single storage slot, accessible via
getReserves
    uint32  private blockTimestampLast; // uses single storage slot, accessible via
getReserves

    uint public price0CumulativeLast;
    uint public price1CumulativeLast;
    uint public kLast; // reserve0 * reserve1, as of immediately after the most recent
liquidity event

    uint private unlocked = 1;
    modifier lock() {
        require(unlocked == 1, 'TitanSwapV1: LOCKED');
        unlocked = 0;
        _;
        unlocked = 1;
    }

    function getReserves() public view returns (uint112 _reserve0, uint112 _reserve1, uint32
_blockTimestampLast) {
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }

    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to,
value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TitanSwapV1:
TRANSFER_FAILED');
    }

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    constructor() public {
        factory = msg.sender;
    }

    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external {
        require(msg.sender == factory, 'TitanSwapV1: FORBIDDEN'); // sufficient check
        token0 = _token0;
        token1 = _token1;
    }

    // update reserves and, on the first call per block, price accumulators
    function _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1)
private {
        require(balance0 <= uint112(-1) && balance1 <= uint112(-1), 'TitanSwapV1:
OVERFLOW');
        uint32 blockTimestamp = uint32(block.timestamp % 2**32);
        uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is desired
        if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
            // * never overflows, and + overflow is desired
            price0CumulativeLast += uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) *
timeElapsed;
            price1CumulativeLast += uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) *
timeElapsed;
        }
        reserve0 = uint112(balance0);
        reserve1 = uint112(balance1);
        blockTimestampLast = blockTimestamp;
        emit Sync(reserve0, reserve1);
    }

    // if fee is on, mint liquidity equivalent to 1/6th of the growth in sqrt(k)
    function _mintFee(uint112 _reserve0, uint112 _reserve1) private returns (bool feeOn)
{
        address feeTo = ITitanSwapV1Factory(factory).feeTo();
        feeOn = feeTo != address(0);
        uint _kLast = kLast; // gas savings
        if (feeOn) {
            if (_kLast != 0) {
                uint rootK = Math.sqrt(uint(_reserve0).mul(_reserve1));
                uint rootKLast = Math.sqrt(_kLast);
                if (rootK > rootKLast) {
                    uint numerator = totalSupply.mul(rootK.sub(rootKLast));
                    uint denominator = rootK.mul(5).add(rootKLast);
                    uint liquidity = numerator / denominator;
                    if (liquidity > 0) _mint(feeTo, liquidity);
                }
            }
        } else if (_kLast != 0) {
            kLast = 0;
        }
    }

    // this low-level function should be called from a contract which performs important
safety checks
    function mint(address to) external lock returns (uint liquidity) {
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        uint balance0 = IERC20(token0).balanceOf(address(this));
        uint balance1 = IERC20(token1).balanceOf(address(this));
        uint amount0 = balance0.sub(_reserve0);
        uint amount1 = balance1.sub(_reserve1);

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply; // gas savings, must be defined here since
totalSupply can update in _mintFee
        if (_totalSupply == 0) {
            liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
            _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first
MINIMUM_LIQUIDITY tokens
        } else {
            liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0,
amount1.mul(_totalSupply) / _reserve1);
        }
        require(liquidity > 0, 'TitanSwapV1: INSUFFICIENT_LIQUIDITY_MINTED');
        _mint(to, liquidity);

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are
up-to-date
        emit Mint(msg.sender, amount0, amount1);
    }

    // this low-level function should be called from a contract which performs important
safety checks
    function burn(address to) external lock returns (uint amount0, uint amount1) {
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        address _token0 = token0;                                // gas savings
        address _token1 = token1;                                // gas savings
        uint balance0 = IERC20(_token0).balanceOf(address(this));
        uint balance1 = IERC20(_token1).balanceOf(address(this));
        uint liquidity = balanceOf[address(this)];

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply; // gas savings, must be defined here since
totalSupply can update in _mintFee
        amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata
distribution
        amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata
distribution
        require(amount0 > 0 && amount1 > 0, 'TitanSwapV1: INSUFFICIENT_LIQUIDITY_BURNED');
        _burn(address(this), liquidity);
        _safeTransfer(_token0, to, amount0);
        _safeTransfer(_token1, to, amount1);
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are
up-to-date
```

```
        emit Burn(msg.sender, amount0, amount1, to);
    }

    // this low-level function should be called from a contract which performs important
safety checks
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external
lock {
        require(amount0Out > 0 || amount1Out > 0, 'TitanSwapV1:
INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'TitanSwapV1:
INSUFFICIENT_LIQUIDITY');

        uint balance0;
        uint balance1;
        { // scope for _token{0,1}, avoids stack too deep errors
            address _token0 = token0;
            address _token1 = token1;
            require(to != _token0 && to != _token1, 'TitanSwapV1: INVALID_TO');
            if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically
transfer tokens
            if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically
transfer tokens
            if (data.length > 0) ITitanSwapV1Callee(to).titanSwapV1Call(msg.sender,
amount0Out, amount1Out, data);
            balance0 = IERC20(_token0).balanceOf(address(this));
            balance1 = IERC20(_token1).balanceOf(address(this));
        }
        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 -
amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 -
amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'TitanSwapV1: INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
            uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
            uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
            require(balance0Adjusted.mul(balance1Adjusted) >=
uint(_reserve0).mul(_reserve1).mul(1000**2), 'TitanSwapV1: K');
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }

    // force balances to match reserves
    function skim(address to) external lock {
        address _token0 = token0; // gas savings
        address _token1 = token1; // gas savings
        _safeTransfer(_token0, to,
IERC20(_token0).balanceOf(address(this)).sub(reserve0));
        _safeTransfer(_token1, to,
IERC20(_token1).balanceOf(address(this)).sub(reserve1));
    }

    // force reserves to match balances
    function sync() external lock {
        _update(IERC20(token0).balanceOf(address(this)),
IERC20(token1).balanceOf(address(this)), reserve0, reserve1);
    }
}

// contracts/TitanSwapV1Router.sol
pragma solidity =0.6.12;

import "./interfaces/ITitanSwapV1Router01.sol";
import "./interfaces/ITitanSwapV1Factory.sol";
import "./interfaces/ITitanSwapV1Pair.sol";
import "./interfaces/IWETH.sol";
import "./interfaces/IERC20.sol";
import "./libraries/SafeMath.sol";
import "./libraries/TitanSwapV1Library.sol";
import "./libraries/TransferHelper.sol";

contract TitanSwapV1Router is ITitanSwapV1Router01 {
    using SafeMath for uint;

    address public immutable override factory;
    address public immutable override WETH;

    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'TitanSwapV1Router: EXPIRED');
        _;
    }

    constructor(address _factory, address _WETH) public {
        factory = _factory;
        WETH = _WETH;
    }

    receive() external payable {
        assert(msg.sender == WETH); // only accept ETH via fallback from the WETH contract
    }

    // **** ADD LIQUIDITY ****
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin
    ) internal virtual returns (uint amountA, uint amountB) {
        // create the pair if it doesn't exist yet
        if (ITitanSwapV1Factory(factory).getPair(tokenA, tokenB) == address(0)) {
            ITitanSwapV1Factory(factory).createPair(tokenA, tokenB);
        }
        (uint reserveA, uint reserveB) = TitanSwapV1Library.getReserves(factory, tokenA,
tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = TitanSwapV1Library.quote(amountADesired, reserveA,
reserveB);
        if (amountBOptimal <= amountBDesired) {
            require(amountBOptimal >= amountBMin, 'TitanSwapV1Router:
INSUFFICIENT_B_AMOUNT');
            (amountA, amountB) = (amountADesired, amountBOptimal);
        } else {
            uint amountAOptimal = TitanSwapV1Library.quote(amountBDesired, reserveB,
reserveA);
            assert(amountAOptimal <= amountADesired);
            require(amountAOptimal >= amountAMin, 'TitanSwapV1Router:
INSUFFICIENT_A_AMOUNT');
            (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint amountA, uint amountB, uint
liquidity) {
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired,
amountAMin, amountBMin);
        address pair = TitanSwapV1Library.pairFor(factory, tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = ITitanSwapV1Pair(pair).mint(to);
    }

    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline) returns (uint amountToken, uint
amountETH, uint liquidity) {
        (amountToken, amountETH) = _addLiquidity(
        token,
        WETH,
        amountTokenDesired,
        msg.value,
        amountTokenMin,
        amountETHMin
        );
        address pair = TitanSwapV1Library.pairFor(factory, token, WETH);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWETH(WETH).deposit{value: amountETH}();
        assert(IWETH(WETH).transfer(pair, amountETH));
        liquidity = ITitanSwapV1Pair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountETH) TransferHelper.safeTransferETH(msg.sender, msg.value -
amountETH);
    }

    // **** REMOVE LIQUIDITY ****
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountA, uint amountB) {
        address pair = TitanSwapV1Library.pairFor(factory, tokenA, tokenB);
        ITitanSwapV1Pair(pair).transferFrom(msg.sender, pair, liquidity); // send
liquidity to pair
        (uint amount0, uint amount1) = ITitanSwapV1Pair(pair).burn(to);
        (address token0,) = TitanSwapV1Library.sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
        require(amountA >= amountAMin, 'TitanSwapV1Router: INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'TitanSwapV1Router: INSUFFICIENT_B_AMOUNT');
    }

    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountToken, uint amountETH)
{
        (amountToken, amountETH) = removeLiquidity(
        token,
        WETH,
        liquidity,
        amountTokenMin,
        amountETHMin,
        address(this),
        deadline
        );
        TransferHelper.safeTransfer(token, to, amountToken);
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }

    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountA, uint amountB) {
        address pair = TitanSwapV1Library.pairFor(factory, tokenA, tokenB);
        uint value = approveMax ? uint(-1) : liquidity;
        ITitanSwapV1Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountA, amountB) = removeLiquidity(tokenA, tokenB, liquidity, amountAMin,
amountBMin, to, deadline);
    }

    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountToken, uint amountETH) {
        address pair = TitanSwapV1Library.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        ITitanSwapV1Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountToken, amountETH) = removeLiquidityETH(token, liquidity, amountTokenMin,
amountETHMin, to, deadline);
    }

    // **** REMOVE LIQUIDITY (supporting fee-on-transfer tokens) ****
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountETH) {
        (, amountETH) = removeLiquidity(
        token,
        WETH,
        liquidity,
        amountTokenMin,
        amountETHMin,
        address(this),
        deadline
        );
        TransferHelper.safeTransfer(token, to, IERC20(token).balanceOf(address(this)));
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }

    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountETH) {
        address pair = TitanSwapV1Library.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        ITitanSwapV1Pair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        amountETH = removeLiquidityETHSupportingFeeOnTransferTokens(
            token, liquidity, amountTokenMin, amountETHMin, to, deadline);
    }

    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, address[] memory path, address _to) internal
virtual {
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = TitanSwapV1Library.sortTokens(input, output);
            uint amountOut = amounts[i + 1];
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) :
(amountOut, uint(0));
            address to = i < path.length - 2 ? TitanSwapV1Library.pairFor(factory, output,
path[i + 2]) : _to;
            ITitanSwapV1Pair(TitanSwapV1Library.pairFor(factory, input, output)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = TitanSwapV1Library.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'TitanSwapV1Router:
INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, TitanSwapV1Library.pairFor(factory, path[0], path[1]),
amounts[0]);
        _swap(amounts, path, to);
    }

    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = TitanSwapV1Library.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'TitanSwapV1Router: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, TitanSwapV1Library.pairFor(factory, path[0], path[1]),
amounts[0]);
            _swap(amounts, path, to);
    }

    function swapExactETHForTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline) returns (uint[] memory amounts){
        require(path[0] == WETH, 'TitanSwapV1Router: INVALID_PATH');
        amounts = TitanSwapV1Library.getAmountsOut(factory, msg.value, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'TitanSwapV1Router:
INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).deposit{value: amounts[0]}();
        assert(IWETH(WETH).transfer(TitanSwapV1Library.pairFor(factory, path[0],
path[1]), amounts[0]));
        _swap(amounts, path, to);
    }

    function swapTokensForExactETH(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts){
        require(path[path.length - 1] == WETH, 'TitanSwapV1Router: INVALID_PATH');
        amounts = TitanSwapV1Library.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'TitanSwapV1Router: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, TitanSwapV1Library.pairFor(factory, path[0], path[1]),
amounts[0]);
        _swap(amounts, path, address(this));
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }

    function swapExactTokensForETH(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts){
        require(path[path.length - 1] == WETH, 'TitanSwapV1Router: INVALID_PATH');
        amounts = TitanSwapV1Library.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'TitanSwapV1Router:
INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, TitanSwapV1Library.pairFor(factory, path[0], path[1]),
amounts[0]
        );
        _swap(amounts, path, address(this));
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }

    function swapETHForExactTokens(
        uint amountOut,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline) returns (uint[] memory amounts){
        require(path[0] == WETH, 'TitanSwapV1Router: INVALID_PATH');
        amounts = TitanSwapV1Library.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= msg.value, 'TitanSwapV1Router: EXCESSIVE_INPUT_AMOUNT');
        IWETH(WETH).deposit{value: amounts[0]}();
        assert(IWETH(WETH).transfer(TitanSwapV1Library.pairFor(factory, path[0],
path[1]), amounts[0]));
        _swap(amounts, path, to);
        // refund dust eth, if any
        if (msg.value > amounts[0]) TransferHelper.safeTransferETH(msg.sender, msg.value
- amounts[0]);
    }

    // **** SWAP (supporting fee-on-transfer tokens) ****
    // requires the initial amount to have already been sent to the first pair
    function _swapSupportingFeeOnTransferTokens(address[] memory path, address _to)
internal virtual {
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = TitanSwapV1Library.sortTokens(input, output);
            ITitanSwapV1Pair pair = ITitanSwapV1Pair(TitanSwapV1Library.pairFor(factory,
input, output));
            uint amountInput;
            uint amountOutput;
            { // scope to avoid stack too deep errors
                (uint reserve0, uint reserve1,) = pair.getReserves();
                (uint reserveInput, uint reserveOutput) = input == token0 ? (reserve0,
reserve1) : (reserve1, reserve0);
                amountInput = IERC20(input).balanceOf(address(pair)).sub(reserveInput);
                amountOutput = TitanSwapV1Library.getAmountOut(amountInput, reserveInput,
reserveOutput);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOutput) :
(amountOutput, uint(0));
            address to = i < path.length - 2 ? TitanSwapV1Library.pairFor(factory, output,
path[i + 2]) : _to;
            pair.swap(amount0Out, amount1Out, to, new bytes(0));
        }
    }

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) {
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, TitanSwapV1Library.pairFor(factory, path[0], path[1]),
amountIn
        );
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >=
amountOutMin,
            'TitanSwapV1Router: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }

    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline){
        require(path[0] == WETH, 'TitanSwapV1Router: INVALID_PATH');
        uint amountIn = msg.value;
        IWETH(WETH).deposit{value: amountIn}();
        assert(IWETH(WETH).transfer(TitanSwapV1Library.pairFor(factory, path[0],
path[1]), amountIn));
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >=
amountOutMin,
            'TitanSwapV1Router: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }

    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline){
        require(path[path.length - 1] == WETH, 'TitanSwapV1Router: INVALID_PATH');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, TitanSwapV1Library.pairFor(factory, path[0], path[1]),
amountIn
        );
        _swapSupportingFeeOnTransferTokens(path, address(this));
        uint amountOut = IERC20(WETH).balanceOf(address(this));
        require(amountOut >= amountOutMin, 'TitanSwapV1Router:
INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).withdraw(amountOut);
        TransferHelper.safeTransferETH(to, amountOut);
    }

    // **** LIBRARY FUNCTIONS ****
    function quote(uint amountA, uint reserveA, uint reserveB) public pure virtual override
returns (uint amountB) {
        return TitanSwapV1Library.quote(amountA, reserveA, reserveB);
    }

    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) public pure
virtual override returns (uint amountOut){
        return TitanSwapV1Library.getAmountOut(amountIn, reserveIn, reserveOut);
    }

    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) public pure
virtual override returns (uint amountIn){
        return TitanSwapV1Library.getAmountIn(amountOut, reserveIn, reserveOut);
    }

    function getAmountsOut(uint amountIn, address[] memory path) public view virtual
override returns (uint[] memory amounts){
        return TitanSwapV1Library.getAmountsOut(factory, amountIn, path);
    }

    function getAmountsIn(uint amountOut, address[] memory path) public view virtual
override returns (uint[] memory amounts){
        return TitanSwapV1Library.getAmountsIn(factory, amountOut, path);
    }
}
```

# 5. Appendix B: vulnerability risk rating criteria

| Smart contract vulnerability rating standard | |
|---|---|
| **Vulnerability rating** | Vulnerability rating description |
| **High risk vulnerability** | The loophole which can directly cause the contract or the user's fund loss, such as the value overflow loophole which can cause the value of the substitute currency to zero, the false recharge loophole that can cause the exchange to lose the substitute coin, can cause the contract account to lose the ETH or the reentry loophole of the substitute currency, and so on; It can cause the loss of ownership rights of token contract, such as: the key function access control defect or call injection leads to the key function access control bypassing, and the loophole that the token contract can not work properly. Such as: a denial-of-service vulnerability due to sending ETHs to a malicious address, and a denial-of-service vulnerability due to gas depletion. |
| **Middle risk vulnerability** | High risk vulnerabilities that need specific addresses to trigger, such as numerical overflow vulnerabilities that can be triggered by the owner of a token contract, access control defects of non-critical functions, and logical design defects that do not result in direct capital losses, etc. |
| **Low risk vulnerability** | A vulnerability that is difficult to trigger, or that will harm a limited number after triggering, such as a numerical overflow that requires a large number of ETH or tokens to trigger, and a vulnerability that the attacker cannot directly profit from after triggering a numerical overflow. Rely on risks by specifying the order of transactions triggered by a high gas. |

# 6. Appendix C：Introduction of test tool

## 6.1. Manticore

Manticore is a symbolic execution tool for analysis of binaries and smart contracts. It discovers inputs that crash programs via memory safety violations. Manticore records an instruction-level trace of execution for each generated input and exposes programmatic access to its analysis engine via a Python API.

## 6.2. Oyente

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

## 6.3. securify.sh

Securify can verify common security issues with smart contracts, such as transactional out-of-order and lack of input validation. It analyzes all possible execution paths of the program while fully automated. In addition, Securify has a specific language for specifying vulnerabilities. Securify can keep an eye on current security and other reliability issues.

## 6.4. Echidna

Echidna is a Haskell library designed for fuzzing EVM code.

## 6.5. MAIAN

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

## 6.6. ethersplay

Ethersplay is an EVM disassembler that contains related analysis tools.

## 6.7. ida-evm

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

## 6.8. Remix-ide

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

## 6.9. Knownsec Penetration Tester Special Toolkit

Knownsec penetration tester special tool kit, developed and collected by Knownsec penetration testing engineers, includes batch automatic testing tools dedicated to testers, self-developed tools, scripts, or utility tools.