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	20210326	V1.0

Report Information

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

Copyright Notice

Table of Contents

1. Introduction

The effective test time of this report is from From March 24, 2021 to March 26, 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.

The scope of this smart contract security audit does not include external contract calls, new attack methods that may appear in the future, and code after contract upgrades or tampering. (With the development of the project, the smart contract may add a new pool , New functional modules, new external contract calls, etc.), does not include front-end security and server security.

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：

Report query address link:

Target information of the YouSwap audit:

Target information	
Token name	YouSwap
Contract address	YouSwapFactory： https://ropsten.etherscan.io/address/0x2794bF982476C95755604bad8Bf7280f53f71786#code

	Repurchase: https://ropsten.etherscan.io/address/0xcc9BbCaB68c22CfCb5cCf1C461138aD4C79f0cD3#code YouSwapRouter: https://ropsten.etherscan.io/address/0x121E6174575F776c9A58a92F475b8c1a77eF63DB#code
Code type	Ethereum smart contract code
Code language	solidity

Contract documents and hash:

Contract documents	MD5
Repurchase.sol	F0E13EE37ED279D27066E2B0693B06A9
IERC20.sol	C0B512FC8612A4A480AEE4CB1FDD89DD
IYouSwapCallee.sol	6D323A9D289D0C7DD213D7D77807A965
IYouSwapERC20.sol	C7AE5CFE6C9E6A90D467A15BB9E9ABC4
IYouSwapFactory.sol	CD95507B6C2EEF2433C93AB7C2378A4F
IYouSwapPair.sol	614E14D8F21E2CF9FB94BCE92526F206
MathV1.sol	683C9332744A5B7400B22F5606FE1D0A
UQ112x112.sol	A9E748B2299564A1FAA22BE501135F2D
YouSwapERC20.sol	A6BA582C125C11FD5C5A53A40319D96B
YouSwapFactory.sol	44C25F80F444CA8B5A192FD9F156DA13
YouSwapPair.sol	9C4C56C05C09AED821EBC9B8AB59CC83

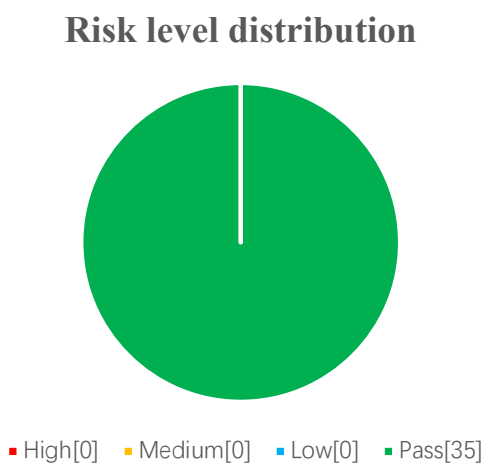
YouSwapRouter.sol	8A63C64773DF9F784FBF4D6268858F24

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	35



2.2 Audit Result

Result of audit			
Audit Target	**Audit**	**Status**	**Audit Description**
Business security testing	**Caller add, delete, modify and check logic design**	Pass	After testing, there is no such safety vulnerability.
	Emergency withdrawal logic design	Pass	After testing, there is no such safety vulnerability.
	Repurchase and destruction logic design	Pass	After testing, there is no such safety vulnerability.
	Create trading pair logic design	Pass	After testing, there is no such safety vulnerability.
	Transaction fee related logic design	Pass	After testing, there is no such safety vulnerability.
	Mint function logic design	Pass	After testing, there is no such safety vulnerability.
	Burn function logic design	Pass	After testing, there is no such safety vulnerability.
	Swap asset transaction logic design	Pass	After testing, there is no such safety vulnerability.
Basic code vulnerability detection	**Compiler version security**	Pass	After testing, there is no such safety vulnerability.
	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. Caller add, delete, modify and check logic design【PASS】

Audit the logic design of adding, deleting, modifying and checking caller to check whether the parameter verification, function caller permission verification, and related logic design are reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
function addCaller(address _newCaller) public onlyOwner returns (bool) {
        require(_newCaller != address(0), "NewCaller is the zero address");
        return EnumerableSet.add(_caller, _newCaller);
    }

    function delCaller(address _delCaller) public onlyOwner returns (bool) {
        require(_delCaller != address(0), "DelCaller is the zero address");
        return EnumerableSet.remove(_caller, _delCaller);
    }

    function getCallerLength() public view returns (uint256) {
        return EnumerableSet.length(_caller);
    }

    function isCaller(address _call) public view returns (bool) {
        return EnumerableSet.contains(_caller, _call);
    }

    function getCaller(uint256 _index) public view returns (address){
        require(_index <= getCallerLength() - 1, "index out of bounds");
        return EnumerableSet.at(_caller, _index);
    }
```

Recommendation： nothing.

3.2. Emergency withdrawal logic design【PASS】

Audit the logic design of the emergency withdrawal, check whether the parameters are verified, the function caller authority verification, and whether the related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
    function setEmergencyAddress(address _newAddress) public onlyOwner {
        require(_newAddress != address(0), "Is zero address");
        emergencyAddress = _newAddress;
    }
    function emergencyWithdraw(address _token) public onlyOwner {
        require(IERC20(_token).balanceOf(address(this)) > 0, "Insufficient contract balance");
        IERC20(_token).transfer(emergencyAddress,
IERC20(_token).balanceOf(address(this)));
```

Recommendation： nothing.

3.3. Repurchase and destruction logic design【PASS】

Audit the logic design of repurchase destruction, check whether the parameters are verified, function caller authority verification, and related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
    function swap() external onlyCaller returns (uint256 amountOut){
        require(IERC20(USDT).balanceOf(address(this)) >= amountIn, "Insufficient contract
balance");
        (uint256 reserve0, uint256 reserve1,) = IYouSwapPair(YOU_USDT).getReserves();
        uint256 amountInWithFee = amountIn.mul(997);
        amountOut = amountIn.mul(997).mul(reserve0) /
reserve1.mul(1000).add(amountInWithFee);
        _safeTransfer(USDT, YOU_USDT, amountIn);
        IYouSwapPair(YOU_USDT).swap(amountOut, 0, destroyAddress, new bytes(0));
```

```
}
```

Recommendation： nothing.

3.4. Create trading pair logic design【PASS】

Audit the logic design of the creation transaction, check whether the parameter verification, function caller authority verification, and related logic design are reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

```
function createPair(address tokenA, address tokenB) external returns (address pair) {
        require(tokenA != tokenB, 'YouSwap: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'YouSwap: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'YouSwap: PAIR_EXISTS'); // single check is sufficient
        bytes memory bytecode = type(YouSwapPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        IYouSwapPair(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);
```

Recommendation： nothing.

3.5. Transaction fee related logic design【PASS】

Audit the logic design related to transaction fees, check whether the parameters are verified, function caller authority verification, and whether the related logic design is reasonable.

Audit analysis: The relevant logic design is reasonable and correct.

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

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

function setFeeToRate(uint256 _rate) external {
    require(msg.sender == feeToSetter, 'YouSwap: FORBIDDEN');
    require(_rate > 0, "YouSwap: FEE_TO_RATE_OVERFLOW");
    feeToRate = _rate.sub(1);
}
```

Recommendation： nothing.

3.6. Mint function logic design【PASS】

Audit the logic design of mint's increased liquidity token function to check whether the parameters are verified, the function caller authority verification, and the related logic design is reasonable

Audit analysis: The relevant logic design is reasonable and correct.

```
// 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, 'YouSwap: 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);
    }
```

Recommendation： nothing.

3.7. Burn function logic design【PASS】

Audit the logic design of burn liquidity destruction function to check whether the parameters are verified, function caller authority verification, and related logic design is reasonable

Audit analysis: The relevant logic design is reasonable and correct.

```
    // 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, 'YouSwap:
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);
    }
```

Recommendation： nothing.

3.8. Swap asset transaction logic design【PASS】

Audit the logic design of swap asset transaction function, check whether the parameter verification, function caller authority verification, and related logic design are reasonable

Audit analysis: The relevant logic design is reasonable and correct.

```
// 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, 'YouSwap: INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'YouSwap: 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, 'YouSwap: 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) IYouSwapCallee(to).YouSwapV2Call(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, 'YouSwap:
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), 'YouSwap: K');
        }

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

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：

Repurchase.sol

```
/**
  *Submitted for verification at Etherscan.io on 2021-03-17
*/
// File: localhost/contracts/interfaces/IYouSwapPair.sol
pragma solidity >=0.5.0;

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

        function name() external pure returns (string memory);
        function symbol() external pure returns (string memory);
        function decimals() external pure returns (uint8);
        function totalSupply() external view returns (uint);
        function balanceOf(address owner) external view returns (uint);
        function allowance(address owner, address spender) external view returns (uint);

        function approve(address spender, uint value) external returns (bool);
        function transfer(address to, uint value) external returns (bool);
        function transferFrom(address from, address to, uint value) external returns (bool);

        function DOMAIN_SEPARATOR() external view returns (bytes32);
        function PERMIT_TYPEHASH() external pure returns (bytes32);
        function nonces(address owner) external view returns (uint);

        function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r,
bytes32 s) external;
```

```
    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);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32
blockTimestampLast);
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
}

// File: localhost/contracts/libraries/EnumerableSet.sol
```

```
pragma solidity =0.5.16;


/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
 * types.
 *
 * Sets have the following properties:
 *
 * - Elements are added, removed, and checked for existence in constant time
 * (O(1)).
 * - Elements are enumerated in O(n). No guarantees are made on the ordering.
 *
 * ```
 * contract Example {
 *         // Add the library methods
 *         using EnumerableSet for EnumerableSet.AddressSet;
 *
 *         // Declare a set state variable
 *         EnumerableSet.AddressSet private mySet;
 * }
 * ```
 *
 * As of v3.3.0, sets of type `bytes32` (`Bytes32Set`), `address` (`AddressSet`)
 * and `uint256` (`UintSet`) are supported.
 */
library EnumerableSet {
        // To implement this library for multiple types with as little code
        // repetition as possible, we write it in terms of a generic Set type with
        // bytes32 values.
        // The Set implementation uses private functions, and user-facing
        // implementations (such as AddressSet) are just wrappers around the
        // underlying Set.
```

```
// This means that we can only create new EnumerableSets for types that fit
// in bytes32.

struct Set {
    // Storage of set values
    bytes32[] _values;

    // Position of the value in the `values` array, plus 1 because index 0
    // means a value is not in the set.
    mapping (bytes32 => uint256) _indexes;
}

/**
 * @dev Add a value to a set. O(1).
 *
 * Returns true if the value was added to the set, that is if it was not
 * already present.
 */
function _add(Set storage set, bytes32 value) private returns (bool) {
    if (!_contains(set, value)) {
        set._values.push(value);
        // The value is stored at length-1, but we add 1 to all indexes
        // and use 0 as a sentinel value
        set._indexes[value] = set._values.length;
        return true;
    } else {
        return false;
    }
}

/**
 * @dev Removes a value from a set. O(1).
 *
```

```
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function _remove(Set storage set, bytes32 value) private returns (bool) {
        // We read and store the value's index to prevent multiple reads from the same storage slot
        uint256 valueIndex = set._indexes[value];

        if (valueIndex != 0) { // Equivalent to contains(set, value)
            // To delete an element from the _values array in O(1), we swap the element to delete with the last one in
            // the array, and then remove the last element (sometimes called as 'swap and pop').
            // This modifies the order of the array, as noted in {at}.

            uint256 toDeleteIndex = valueIndex - 1;
            uint256 lastIndex = set._values.length - 1;

            // When the value to delete is the last one, the swap operation is unnecessary. However, since this occurs
            // so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' statement.

            bytes32 lastvalue = set._values[lastIndex];

            // Move the last value to the index where the value to delete is
            set._values[toDeleteIndex] = lastvalue;
            // Update the index for the moved value
            set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

            // Delete the slot where the moved value was stored
            set._values.pop();

            // Delete the index for the deleted slot
            delete set._indexes[value];
```

```
            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function _contains(Set storage set, bytes32 value) private view returns (bool) {
        return set._indexes[value] != 0;
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function _length(Set storage set) private view returns (uint256) {
        return set._values.length;
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function _at(Set storage set, uint256 index) private view returns (bytes32) {
        require(set._values.length > index, "EnumerableSet: index out of bounds");
        return set._values[index];
```

```
}

// Bytes32Set

struct Bytes32Set {
    Set _inner;
}

/**
 * @dev Add a value to a set. O(1).
 *
 * Returns true if the value was added to the set, that is if it was not
 * already present.
 */
function add(Bytes32Set storage set, bytes32 value) internal returns (bool) {
    return _add(set._inner, value);
}

/**
 * @dev Removes a value from a set. O(1).
 *
 * Returns true if the value was removed from the set, that is if it was
 * present.
 */
function remove(Bytes32Set storage set, bytes32 value) internal returns (bool) {
    return _remove(set._inner, value);
}

/**
 * @dev Returns true if the value is in the set. O(1).
 */
function contains(Bytes32Set storage set, bytes32 value) internal view returns (bool) {
    return _contains(set._inner, value);
```

```
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(Bytes32Set storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(Bytes32Set storage set, uint256 index) internal view returns (bytes32) {
        return _at(set._inner, index);
    }

    // AddressSet

    struct AddressSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
```

```
     * already present.
     */
    function add(AddressSet storage set, address value) internal returns (bool) {
        return _add(set._inner, bytes32(uint256(value)));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(AddressSet storage set, address value) internal returns (bool) {
        return _remove(set._inner, bytes32(uint256(value)));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(AddressSet storage set, address value) internal view returns (bool) {
        return _contains(set._inner, bytes32(uint256(value)));
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(AddressSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
```

```
* Note that there are no guarantees on the ordering of values inside the
* array, and it may change when more values are added or removed.
*
* Requirements:
*
* - `index` must be strictly less than {length}.
*/
function at(AddressSet storage set, uint256 index) internal view returns (address) {
        return address(uint256(_at(set._inner, index)));
}



// UintSet

struct UintSet {
        Set _inner;
}

/**
 * @dev Add a value to a set. O(1).
 *
 * Returns true if the value was added to the set, that is if it was not
 * already present.
 */
function add(UintSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
}

/**
 * @dev Removes a value from a set. O(1).
 *
 * Returns true if the value was removed from the set, that is if it was
 * present.
```

```
     */
    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(UintSet storage set, uint256 value) internal view returns (bool) {
        return _contains(set._inner, bytes32(value));
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
        return uint256(_at(set._inner, index));
    }
}
```

```
// File: localhost/contracts/libraries/MathV1.sol

pragma solidity >=0.5.0;

// a library for performing various math operations

library Math {
    function min(uint x, uint y) internal pure returns (uint z) {
        z = x < y ? x : y;
    }

    // babylonian method (https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_method)
    function sqrt(uint y) internal pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }
}

// a library for performing overflow-safe math, courtesy of DappHub (https://github.com/dapphub/ds-math)
library SafeMath {
    function add(uint x, uint y) internal pure returns (uint z) {
        require((z = x + y) >= x, 'ds-math-add-overflow');
    }
```

```
    function sub(uint x, uint y) internal pure returns (uint z) {
        require((z = x - y) <= x, 'ds-math-sub-underflow');
    }

    function mul(uint x, uint y) internal pure returns (uint z) {
        require(y == 0 || (z = x * y) / y == x, 'ds-math-mul-overflow');
    }
}
// File: localhost/contracts/interfaces/IERC20.sol

//SPDX-License-Identifier: SimPL-2.0
pragma solidity >=0.5.0;

interface IERC20 {
    /**
     * @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);
}
// File: localhost/contracts/token/Repurchase.sol

pragma solidity =0.5.16;

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

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
```

```
     */
    function transferOwnership(address newOwner) public    onlyOwner {
        require(newOwner != address(0), "YouSwap:
NEW_OWNER_IS_THE_ZERO_ADDRESS");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}


contract Repurchase is Ownable{
    using SafeMath for uint256;

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _caller;

    bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)')));
    address public constant USDT = 0xFA8B1212119197eC88Fc768AF1b04aD0519Ad994;
    address public constant YOU = 0x1093EcdBACa4168F136f3BEfd17a261bfbbeDdA3;
    address public constant YOU_USDT = 0x94339BbdB9550a5758Da5EC65C547520EC819520;
    address public constant destroyAddress = 0xF971ec570538874F33cbc35C5156e9C3bC8CeF66;
    address public emergencyAddress;
    uint256 public amountIn;

    constructor (uint256 _amount, address _emergencyAddress) public {
        require(_amount > 0, "Amount must be greater than zero");
        require(_emergencyAddress != address(0), "Is zero address");
        amountIn = _amount;
        emergencyAddress = _emergencyAddress;
    }

    function setAmountIn(uint256 _newIn) public onlyOwner {
        amountIn = _newIn;
```

```
    }

    function setEmergencyAddress(address _newAddress) public onlyOwner {
        require(_newAddress != address(0), "Is zero address");
        emergencyAddress = _newAddress;
    }

    function addCaller(address _newCaller) public onlyOwner returns (bool) {
        require(_newCaller != address(0), "NewCaller is the zero address");
        return EnumerableSet.add(_caller, _newCaller);
    }

    function delCaller(address _delCaller) public onlyOwner returns (bool) {
        require(_delCaller != address(0), "DelCaller is the zero address");
        return EnumerableSet.remove(_caller, _delCaller);
    }

    function getCallerLength() public view returns (uint256) {
        return EnumerableSet.length(_caller);
    }

    function isCaller(address _call) public view returns (bool) {
        return EnumerableSet.contains(_caller, _call);
    }

    function getCaller(uint256 _index) public view returns (address){
        require(_index <= getCallerLength() - 1, "index out of bounds");
        return EnumerableSet.at(_caller, _index);
    }

    function swap() external onlyCaller returns (uint256 amountOut){
        require(IERC20(USDT).balanceOf(address(this)) >= amountIn, "Insufficient contract
balance");
```

```
        (uint256 reserve0, uint256 reserve1,) = IYouSwapPair(YOU_USDT).getReserves();
        uint256 amountInWithFee = amountIn.mul(997);
        amountOut = amountIn.mul(997).mul(reserve0) /
reserve1.mul(1000).add(amountInWithFee);
        _safeTransfer(USDT, YOU_USDT, amountIn);
        IYouSwapPair(YOU_USDT).swap(amountOut, 0, destroyAddress, new bytes(0));
    }

    modifier onlyCaller() {
        require(isCaller(msg.sender), "Not the caller");
        _;
    }

    function emergencyWithdraw(address _token) public onlyOwner {
        require(IERC20(_token).balanceOf(address(this)) > 0, "Insufficient contract balance");
        IERC20(_token).transfer(emergencyAddress, IERC20(_token).balanceOf(address(this)));
    }

    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))), 'YouSwap:
TRANSFER_FAILED');
    }
}
```

IERC20.sol

```
//SPDX-License-Identifier: SimPL-2.0
pragma solidity >=0.5.0;

interface IERC20 {
    /**
     * @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);
}
```

IYouSwapCallee.sol

```
pragma solidity >=0.5.0;

interface IYouSwapCallee {
    function YouSwapV2Call(address sender, uint amount0, uint amount1, bytes calldata data)
external;
}
```

IYouSwapERC20

```
pragma solidity >=0.5.0;

interface IYouSwapERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r,
bytes32 s) external;
}
```

IYouSwapFactory.sol

```
pragma solidity >=0.5.0;

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

    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);

    function createPair(address tokenA, address tokenB) external returns (address pair);

    function setFeeTo(address) external;
    function setFeeToSetter(address) external;

    function setFeeToRate(uint256) external;
    function feeToRate() external view returns (uint256);
}
```

IYouSwapPair.sol

```
pragma solidity >=0.5.0;

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

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
```

```
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r,
bytes32 s) external;

    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);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32
blockTimestampLast);
```

```
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
}
```

MathV1.sol

```
pragma solidity >=0.5.0;

// a library for performing various math operations

library Math {
    function min(uint x, uint y) internal pure returns (uint z) {
        z = x < y ? x : y;
    }

    // babylonian method
(https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_method)
    function sqrt(uint y) internal pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
```

```
            } else if (y != 0) {
                z = 1;
            }
        }
}

// a library for performing overflow-safe math, courtesy of DappHub (https://github.com/dapphub/ds-math)
library SafeMath {
        function add(uint x, uint y) internal pure returns (uint z) {
            require((z = x + y) >= x, 'ds-math-add-overflow');
        }

        function sub(uint x, uint y) internal pure returns (uint z) {
            require((z = x - y) <= x, 'ds-math-sub-underflow');
        }

        function mul(uint x, uint y) internal pure returns (uint z) {
            require(y == 0 || (z = x * y) / y == x, 'ds-math-mul-overflow');
        }
}
```

UQ112x112.sol

```
pragma solidity =0.5.16;

// a library for handling binary fixed point numbers (https://en.wikipedia.org/wiki/Q_(number_format))

// range: [0, 2**112 - 1]
// resolution: 1 / 2**112

library UQ112x112 {
        uint224 constant Q112 = 2**112;

        // encode a uint112 as a UQ112x112
```

```
    function encode(uint112 y) internal pure returns (uint224 z) {
        z = uint224(y) * Q112; // never overflows
    }

    // divide a UQ112x112 by a uint112, returning a UQ112x112
    function uqdiv(uint224 x, uint112 y) internal pure returns (uint224 z) {
        z = x / uint224(y);
    }
}
```

YouSwapERC20.sol

```
pragma solidity =0.5.16;

import './interfaces/IYouSwapERC20.sol';
import './libraries/MathV1.sol';

contract YouSwapERC20 is IYouSwapERC20 {
    using SafeMath for uint;

    string public constant name = 'YouSwap LP Token';
    string public constant symbol = 'ULP';
    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, 'YouSwap: 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, 'YouSwap:
INVALID_SIGNATURE');
            _approve(owner, spender, value);
        }
}
```

YouSwapFactory,sol

```
pragma solidity =0.5.16;

import './interfaces/IYouSwapFactory.sol';
import './YouSwapPair.sol';
import './libraries/MathV1.sol';

contract YouSwapFactory is IYouSwapFactory {
    using SafeMath  for uint;

    address public feeTo;
    address public feeToSetter;
    uint256 public feeToRate;
    bytes32 public initCodeHash;

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

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

    constructor(address _feeToSetter) public {
```

```
        feeToSetter = _feeToSetter;
        initCodeHash = keccak256(abi.encodePacked(type(YouSwapPair).creationCode));
    }

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

    function createPair(address tokenA, address tokenB) external returns (address pair) {
        require(tokenA != tokenB, 'YouSwap: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'YouSwap: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'YouSwap: PAIR_EXISTS'); // single check
is sufficient
        bytes memory bytecode = type(YouSwapPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        IYouSwapPair(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 {
        require(msg.sender == feeToSetter, 'YouSwap: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'YouSwap: FORBIDDEN');
```

```
        feeToSetter = _feeToSetter;
    }

    function setFeeToRate(uint256 _rate) external {
        require(msg.sender == feeToSetter, 'YouSwap: FORBIDDEN');
        require(_rate > 0, "YouSwap: FEE_TO_RATE_OVERFLOW");
        feeToRate = _rate.sub(1);
    }
}
```

YouSwapPair.sol

```
pragma solidity =0.5.16;

import './interfaces/IYouSwapPair.sol';
import './YouSwapERC20.sol';
import './libraries/MathV1.sol';
import './libraries/UQ112x112.sol';
import './interfaces/IERC20.sol';
import './interfaces/IYouSwapFactory.sol';
import './interfaces/IYouSwapCallee.sol';

contract YouSwapPair is IYouSwapPair, YouSwapERC20 {
    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, 'YouSwap: 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))), 'YouSwap:
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, 'YouSwap: 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), 'YouSwap: 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 = IYouSwapFactory(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(IYouSwapFactory(factory).feeToRate()).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, 'YouSwap: 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, 'YouSwap:
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, 'YouSwap:
INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'YouSwap:
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, 'YouSwap: 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) IYouSwapCallee(to).YouSwapV2Call(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, 'YouSwap: 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), 'YouSwap: 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);
        }
}
```

YouSwapRouter.sol

```
/**
 *Submitted for verification at Etherscan.io on 2021-03-13
*/

// File: localhost/contracts/interfaces/ISwapMining.sol

pragma solidity >=0.5.0;

interface ISwapMining {
    function swap(address account, address input, address output, uint256 amount) external returns
(bool);
}

// File: localhost/contracts/interfaces/IWETH.sol

pragma solidity >=0.5.0;

interface IWETH {
    function deposit() external payable;
    function transfer(address to, uint value) external returns (bool);
    function withdraw(uint) external;
}

// File: localhost/contracts/interfaces/IERC20.sol

pragma solidity >=0.5.0;

interface IERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);
```

```
    function name() external view returns (string memory);
    function symbol() external view returns (string memory);
    function decimals() external view returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);
}

// File: localhost/contracts/libraries/SafeMath.sol

pragma solidity =0.6.6;

// a library for performing overflow-safe math, courtesy of DappHub (https://github.com/dapphub/ds-math)

library SafeMath {
    function add(uint x, uint y) internal pure returns (uint z) {
        require((z = x + y) >= x, 'ds-math-add-overflow');
    }

    function sub(uint x, uint y) internal pure returns (uint z) {
        require((z = x - y) <= x, 'ds-math-sub-underflow');
    }

    function mul(uint x, uint y) internal pure returns (uint z) {
        require(y == 0 || (z = x * y) / y == x, 'ds-math-mul-overflow');
    }
}
```

```
// File: localhost/contracts/interfaces/IYouSwapPair.sol

pragma solidity >=0.5.0;

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

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r,
bytes32 s) external;

    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);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32
blockTimestampLast);
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
}

// File: localhost/contracts/libraries/YouSwapLibrary.sol

pragma solidity >=0.5.0;



library YouSwapLibrary {
    using SafeMath for uint;
```

```
    // returns sorted token addresses, used to handle return values from pairs sorted in this order
    function sortTokens(address tokenA, address tokenB) internal pure returns (address token0,
address token1) {
        require(tokenA != tokenB, 'YouSwapLibrary: IDENTICAL_ADDRESSES');
        (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'YouSwapLibrary: ZERO_ADDRESS');
    }

    // calculates the CREATE2 address for a pair without making any external calls
    function pairFor(address factory, address tokenA, address tokenB) internal pure returns (address
pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        pair = address(uint(keccak256(abi.encodePacked(
                hex'ff',
                factory,
                keccak256(abi.encodePacked(token0, token1)),
                hex'8919347964f406fcc7e9a98fd3e05e8ba3e0270039e1e056121a8bffd0f2789e' //
init code hash
            ))));
    }

    // fetches and sorts the reserves for a pair
    function getReserves(address factory, address tokenA, address tokenB) internal view returns (uint
reserveA, uint reserveB) {
        (address token0,) = sortTokens(tokenA, tokenB);
        (uint reserve0, uint reserve1,) = IYouSwapPair(pairFor(factory, tokenA,
tokenB)).getReserves();
        (reserveA, reserveB) = tokenA == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
    }

    // given some amount of an asset and pair reserves, returns an equivalent amount of the other asset
    function quote(uint amountA, uint reserveA, uint reserveB) internal pure returns (uint amountB) {
        require(amountA > 0, 'YouSwapLibrary: INSUFFICIENT_AMOUNT');
```

```
        require(reserveA > 0 && reserveB > 0, 'YouSwapLibrary: INSUFFICIENT_LIQUIDITY');
        amountB = amountA.mul(reserveB) / reserveA;
    }

    // given an input amount of an asset and pair reserves, returns the maximum output amount of the other asset
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) internal pure returns (uint amountOut) {
        require(amountIn > 0, 'YouSwapLibrary: INSUFFICIENT_INPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'YouSwapLibrary: INSUFFICIENT_LIQUIDITY');
        uint amountInWithFee = amountIn.mul(997);
        uint numerator = amountInWithFee.mul(reserveOut);
        uint denominator = reserveIn.mul(1000).add(amountInWithFee);
        amountOut = numerator / denominator;
    }

    // given an output amount of an asset and pair reserves, returns a required input amount of the other asset
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) internal pure returns (uint amountIn) {
        require(amountOut > 0, 'YouSwapLibrary: INSUFFICIENT_OUTPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'YouSwapLibrary: INSUFFICIENT_LIQUIDITY');
        uint numerator = reserveIn.mul(amountOut).mul(1000);
        uint denominator = reserveOut.sub(amountOut).mul(997);
        amountIn = (numerator / denominator).add(1);
    }

    // performs chained getAmountOut calculations on any number of pairs
    function getAmountsOut(address factory, uint amountIn, address[] memory path) internal view returns (uint[] memory amounts) {
        require(path.length >= 2, 'YouSwapLibrary: INVALID_PATH');
```

```
            amounts = new uint[](path.length);
            amounts[0] = amountIn;
            for (uint i; i < path.length - 1; i++) {
                (uint reserveIn, uint reserveOut) = getReserves(factory, path[i], path[i + 1]);
                amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut);
            }
        }

        // performs chained getAmountIn calculations on any number of pairs
        function getAmountsIn(address factory, uint amountOut, address[] memory path) internal view
returns (uint[] memory amounts) {
            require(path.length >= 2, 'YouSwapLibrary: INVALID_PATH');
            amounts = new uint[](path.length);
            amounts[amounts.length - 1] = amountOut;
            for (uint i = path.length - 1; i > 0; i--) {
                (uint reserveIn, uint reserveOut) = getReserves(factory, path[i - 1], path[i]);
                amounts[i - 1] = getAmountIn(amounts[i], reserveIn, reserveOut);
            }
        }
}

// File: localhost/contracts/interfaces/IYouSwapRouter.sol

pragma solidity >=0.6.2;

interface IYouSwapRouter {
        function factory() external pure returns (address);
        function WETH() external pure returns (address);
        function swapMining() external pure returns (address);

        function addLiquidity(
            address tokenA,
            address tokenB,
```

```
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint 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 returns (uint amountA, uint amountB);
    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountToken, uint amountETH);
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
```

function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
external
payable
returns (uint[] memory amounts);
function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to, uint deadline)
external
returns (uint[] memory amounts);
function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
external
returns (uint[] memory amounts);
function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
external
payable
returns (uint[] memory amounts);

function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint amountOut);
function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint amountIn);
function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memory amounts);
function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memory amounts);

function removeLiquidityETHSupportingFeeOnTransferTokens(
address token,
uint liquidity,
uint amountTokenMin,

```
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountETH);
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountETH);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
```

```
    ) external;
}

// File: localhost/contracts/libraries/Ownable.sol

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

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
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);
}

/**
 * @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;
    }
}
// File: localhost/contracts/libraries/TransferHelper.sol

// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.6.0;

// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return
true/false
library TransferHelper {
    function safeApprove(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to,
value));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            'TransferHelper::safeApprove: approve failed'
        );
    }

    function safeTransfer(
        address token,
        address to,
        uint256 value
```

```
    ) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to,
value));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            'TransferHelper::safeTransfer: transfer failed'
        );
    }

    function safeTransferFrom(
        address token,
        address from,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from,
to, value));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            'TransferHelper::transferFrom: transferFrom failed'
        );
    }

    function safeTransferETH(address to, uint256 value) internal {
        (bool success, ) = to.call{value: value}(new bytes(0));
        require(success, 'TransferHelper::safeTransferETH: ETH transfer failed');
    }
}

// File: localhost/contracts/interfaces/IYouSwapFactory.sol
```

```
pragma solidity >=0.5.0;

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

    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);

    function createPair(address tokenA, address tokenB) external returns (address pair);

    function setFeeTo(address) external;
    function setFeeToSetter(address) external;

    function setFeeToRate(uint256) external;
    function feeToRate() external view returns (uint256);
}

// File: localhost/contracts/YouSwapRouter.sol

pragma solidity =0.6.6;
```

```
contract YouSwapRouter is IYouSwapRouter, Ownable {
    using SafeMath for uint;

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

    modifier ensure(uint deadline) {
        require(deadline >= block.timestamp, 'YouSwapRouter: 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
    }

    function setSwapMining(address _swapMininng) public onlyOwner {
        swapMining = _swapMininng;
    }

    // **** 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 (IYouSwapFactory(factory).getPair(tokenA, tokenB) == address(0)) {
            IYouSwapFactory(factory).createPair(tokenA, tokenB);
        }
        (uint reserveA, uint reserveB) = YouSwapLibrary.getReserves(factory, tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = YouSwapLibrary.quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'YouSwapRouter:
INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = YouSwapLibrary.quote(amountBDesired, reserveB,
reserveA);
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'YouSwapRouter:
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 = YouSwapLibrary.pairFor(factory, tokenA, tokenB);
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IYouSwapPair(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 = YouSwapLibrary.pairFor(factory, token, WETH);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWETH(WETH).deposit{value: amountETH}();
        assert(IWETH(WETH).transfer(pair, amountETH));
        liquidity = IYouSwapPair(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 = YouSwapLibrary.pairFor(factory, tokenA, tokenB);
        IYouSwapPair(pair).transferFrom(msg.sender, pair, liquidity); // send liquidity to pair
        (uint amount0, uint amount1) = IYouSwapPair(pair).burn(to);
        (address token0,) = YouSwapLibrary.sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
        require(amountA >= amountAMin, 'YouSwapRouter: INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'YouSwapRouter: 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 = YouSwapLibrary.pairFor(factory, tokenA, tokenB);
        uint value = approveMax ? uint(-1) : liquidity;
        IYouSwapPair(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 = YouSwapLibrary.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        IYouSwapPair(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 = YouSwapLibrary.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        IYouSwapPair(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,) = YouSwapLibrary.sortTokens(input, output);
            uint amountOut = amounts[i + 1];
            if (swapMining != address(0)) {
                ISwapMining(swapMining).swap(msg.sender, input, output, amountOut);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) :
(amountOut, uint(0));
            address to = i < path.length - 2 ? YouSwapLibrary.pairFor(factory, output, path[i + 2]) :
_to;
            IYouSwapPair(YouSwapLibrary.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 = YouSwapLibrary.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'YouSwapRouter:
INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, YouSwapLibrary.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 = YouSwapLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'YouSwapRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, YouSwapLibrary.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, 'YouSwapRouter: INVALID_PATH');
        amounts = YouSwapLibrary.getAmountsOut(factory, msg.value, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'YouSwapRouter:
INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).deposit{value: amounts[0]}();
        assert(IWETH(WETH).transfer(YouSwapLibrary.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, 'YouSwapRouter: INVALID_PATH');
        amounts = YouSwapLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'YouSwapRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, YouSwapLibrary.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, 'YouSwapRouter: INVALID_PATH');
        amounts = YouSwapLibrary.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'YouSwapRouter:
INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, YouSwapLibrary.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, 'YouSwapRouter: INVALID_PATH');
        amounts = YouSwapLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= msg.value, 'YouSwapRouter: EXCESSIVE_INPUT_AMOUNT');
        IWETH(WETH).deposit{value: amounts[0]}();
```

```
            assert(IWETH(WETH).transfer(YouSwapLibrary.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,) = YouSwapLibrary.sortTokens(input, output);
                IYouSwapPair pair = IYouSwapPair(YouSwapLibrary.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 = YouSwapLibrary.getAmountOut(amountInput, reserveInput,
reserveOutput);
                }
                if (swapMining != address(0)) {
                    ISwapMining(swapMining).swap(msg.sender, input, output, amountOutput);
                }
                (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOutput) :
(amountOutput, uint(0));
                address to = i < path.length - 2 ? YouSwapLibrary.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, YouSwapLibrary.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,
            'YouSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
        external
        virtual
        override
        payable
        ensure(deadline)
    {
        require(path[0] == WETH, 'YouSwapRouter: INVALID_PATH');
```

```
        uint amountIn = msg.value;
        IWETH(WETH).deposit{value: amountIn}();
        assert(IWETH(WETH).transfer(YouSwapLibrary.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,
            'YouSwapRouter: 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, 'YouSwapRouter: INVALID_PATH');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, YouSwapLibrary.pairFor(factory, path[0], path[1]), amountIn
        );
        _swapSupportingFeeOnTransferTokens(path, address(this));
        uint amountOut = IERC20(WETH).balanceOf(address(this));
        require(amountOut >= amountOutMin, 'YouSwapRouter:
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 YouSwapLibrary.quote(amountA, reserveA, reserveB);
    }

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

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

    function getAmountsOut(uint amountIn, address[] memory path)
        public
        view
        virtual
        override
```

```
        returns (uint[] memory amounts)
    {
        return YouSwapLibrary.getAmountsOut(factory, amountIn, path);
    }

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

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