Garfield Finance Swap

Smart Contract Audit

Garfield Finance Swap Audit Summary

Project name : Garfield Finance Swap Contract

Project address: None

Code URL : https://github.com/GarfieldLab/garfieldswap-core

Commit : e04f8fc3653ff2edb8abfbacf06ae2530db78d20

Project target : Garfield Finance Swap Contract Audit

Blockchain : Huobi ECO Chain（Heco）

Test result : PASSED

Audit Info

Audit NO : 0X202105090006

Audit Team : Armors Labs

Audit Proofreading: https://armors.io/#project-cases

Garfield Finance Swap Audit

The Garfield Finance Swap team asked us to review and audit their Garfield Finance Swap contract. We looked at the code and now publish our results.

Here is our assessment and recommendations, in order of importance.

Document information

Name	Auditor	Version	Date
Garfield Finance Swap Audit	Rock, Sophia, Rushairer, Rico, David, Alice	1.0.0	2021-05-09

Audit results

Note that as of the date of publishing, the above review reflects the current understanding of known security patterns as they relate to the Garfield Finance Swap contract. The above should not be construed as investment advice.

Based on the widely recognized security status of the current underlying blockchain and smart contract, this audit report is valid for 3 months from the date of output.

(Statement: Armors Labs reports only on facts that have occurred or existed before this report is issued and assumes corresponding responsibilities. Armors Labs is not able to determine the security of its smart contracts and is not responsible for any subsequent or existing facts after this report is issued. The security audit analysis and other content of this report are only based on the documents and information provided by the information provider to Armors Labs at the time of issuance of this report (" information provided " for short). Armors Labs postulates that the information provided is not missing, tampered, deleted or hidden. If the information provided is missing, tampered,

deleted, hidden or reflected in a way that is not consistent with the actual situation, Armors Labs shall not be responsible for the losses and adverse effects caused.)

Audited target file

file	md5
./SwapFactory.sol	6f46c8de2ecf753ee11378b03cbb8c5a
./SwapRouter.sol	6bcd8c8b0ff27aa9f174d23e535031da

Vulnerability analysis

Vulnerability distribution

vulnerability level	number
Critical severity	0
High severity	0
Medium severity	0
Low severity	0

Summary of audit results

Vulnerability	status
Re-Entrancy	safe
Arithmetic Over/Under Flows	safe
Unexpected Blockchain Currency	safe
Delegatecall	safe
Default Visibilities	safe
Entropy Illusion	safe
External Contract Referencing	safe
Short Address/Parameter Attack	safe
Unchecked CALL Return Values	safe
Race Conditions / Front Running	safe
Denial Of Service (DOS)	safe
Block Timestamp Manipulation	safe
Constructors with Care	safe
Unintialised Storage Pointers	safe
Floating Points and Numerical Precision	safe

Vulnerability	status
tx.origin Authentication	safe
Permission restrictions	safe

Contract file

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

interface ISwapFactory {
    event PairCreated(
        address indexed token0,
        address indexed token1,
        address pair,
        uint256
    );

    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(uint256) external view returns (address pair);

    function allPairsLength() external view returns (uint256);

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

    function setFeeTo(address) external;

    function setFeeToSetter(address) external;

    function pairCodeHash() external pure returns (bytes32);
}

interface ISwapPair {
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
    event Transfer(address indexed from, address indexed to, uint256 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 (uint256);

    function balanceOf(address owner) external view returns (uint256);

    function allowance(address owner, address spender)
```

```
        external
        view
        returns (uint256);

    function approve(address spender, uint256 value) external returns (bool);

    function transfer(address to, uint256 value) external returns (bool);

    function transferFrom(
        address from,
        address to,
        uint256 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 (uint256);

    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;

    event Mint(address indexed sender, uint256 amount0, uint256 amount1);
    event Burn(
        address indexed sender,
        uint256 amount0,
        uint256 amount1,
        address indexed to
    );
    event Swap(
        address indexed sender,
        uint256 amount0In,
        uint256 amount1In,
        uint256 amount0Out,
        uint256 amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint256);

    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 (uint256);
```

```
    function price1CumulativeLast() external view returns (uint256);

    function kLast() external view returns (uint256);

    function mint(address to) external returns (uint256 liquidity);

    function burn(address to)
        external
        returns (uint256 amount0, uint256 amount1);

    function swap(
        uint256 amount0Out,
        uint256 amount1Out,
        address to,
        bytes calldata data
    ) external;

    function skim(address to) external;

    function sync() external;

    function fee() external view returns (uint8);

    function feeTo() external view returns (address);

    function getFeeTo() external view returns (address);

    function creator() external view returns (address);

    function birthday() external view returns (uint256);

    function rootKmul() external view returns (uint8);

    function initialize(address, address) external;

    function setFeeTo(address) external;

    function setrootKmul(uint8) external;

    function setFee(uint8) external;
}

interface IERC20 {
    function totalSupply() external view returns (uint256);

    function balanceOf(address account) external view returns (uint256);

    function transfer(address recipient, uint256 amount)
        external
        returns (bool);

    function allowance(address owner, address spender)
        external
        view
        returns (uint256);

    function approve(address spender, uint256 amount) external returns (bool);

    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external returns (bool);

    event Transfer(address indexed from, address indexed to, uint256 value);
```

```
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );

    function name() external pure returns (string memory);

    function symbol() external pure returns (string memory);

    function decimals() external pure returns (uint8);
}

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    function sub(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }
        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");
        return c;
    }

    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }

    function div(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        uint256 c = a / b;
        return c;
    }

    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    function mod(
        uint256 a,
        uint256 b,
```

```
        string memory errorMessage
    ) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}

interface ISwapCallee {
    function swapCall(
        address sender,
        uint256 amount0,
        uint256 amount1,
        bytes calldata data
    ) external;
}

interface IERC20Swap {
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
    event Transfer(address indexed from, address indexed to, uint256 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 (uint256);

    function balanceOf(address owner) external view returns (uint256);

    function allowance(address owner, address spender)
        external
        view
        returns (uint256);

    function approve(address spender, uint256 value) external returns (bool);

    function transfer(address to, uint256 value) external returns (bool);

    function transferFrom(
        address from,
        address to,
        uint256 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 (uint256);

    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;
}
```

```
contract SwapERC20 is IERC20Swap {
    using SafeMath for uint256;

    /// @notice token名称
    string public constant override name = "Garfield Swap LP Token";
    /// @notice token缩写
    string public override symbol = "GLP";
    /// @notice 精度
    uint8 public constant override decimals = 18;
    /// @notice 总量
    uint256 public override totalSupply;
    /// @notice 余额映射
    mapping(address => uint256) public override balanceOf;
    /// @notice 批准映射
    mapping(address => mapping(address => uint256)) public override allowance;

    /// @notice 域分割
    bytes32 public override DOMAIN_SEPARATOR;
    /// @notice The EIP-712 typehash for the contract's domain
    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)"
    bytes32 public constant override PERMIT_TYPEHASH =
        0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;

    /// @notice nonces映射
    mapping(address => uint256) public override nonces;

    /// @notice 批准事件
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
    /// @notice 发送事件
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev 构造函数
     */
    constructor() public {
        // 链ID
        uint256 chainId;
        // 获取链ID
        // solium-disable-next-line
        assembly {
            chainId := chainid()
        }
        //EIP712Domain
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256(
                    "EIP712Domain(string name,string version,uint256 chainId,address verifyingContrac
                ),
                keccak256(bytes(name)),
                keccak256(bytes("1")),
                chainId,
                address(this)
            )
        );
    }

    function _mint(address to, uint256 value) internal {
        totalSupply = totalSupply.add(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(address(0), to, value);
    }
```

```
    function _burn(address from, uint256 value) internal {
        balanceOf[from] = balanceOf[from].sub(value);
        totalSupply = totalSupply.sub(value);
        emit Transfer(from, address(0), value);
    }

    function _approve(
        address owner,
        address spender,
        uint256 value
    ) private {
        allowance[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    function _transfer(
        address from,
        address to,
        uint256 value
    ) private {
        balanceOf[from] = balanceOf[from].sub(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(from, to, value);
    }

    function approve(address spender, uint256 value)
        external
        override
        returns (bool)
    {
        _approve(msg.sender, spender, value);
        return true;
    }

    function transfer(address to, uint256 value)
        external
        override
        returns (bool)
    {
        _transfer(msg.sender, to, value);
        return true;
    }

    function transferFrom(
        address from,
        address to,
        uint256 value
    ) external override returns (bool) {
        if (allowance[from][msg.sender] != uint256(-1)) {
            allowance[from][msg.sender] = allowance[from][msg.sender].sub(
                value
            );
        }
        _transfer(from, to, value);
        return true;
    }

    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external override {
```

```
        require(deadline >= block.timestamp, "Swap: 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,
            "Swap: INVALID_SIGNATURE"
        );
        _approve(owner, spender, value);
    }
}

library Math {
    function min(uint256 x, uint256 y) internal pure returns (uint256 z) {
        z = x < y ? x : y;
    }

    // babylonian method (https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_
    function sqrt(uint256 y) internal pure returns (uint256 z) {
        if (y > 3) {
            z = y;
            uint256 x = y / 2 + 1;
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }
}

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);
    }
}

/**
 * @title 交易钩子合约接口
 */
interface ISwapHook {
    function swapHook(
```

```
        address sender,
        uint256 amount0Out,
        uint256 amount1Out,
        address to
    ) external;
}

/**
 * @title Swap 配对合约
 */
contract SwapPair is SwapERC20 {
    using SafeMath for uint256;
    using UQ112x112 for uint224;

    /// @notice 最小流动性 = 1000
    uint256 public constant MINIMUM_LIQUIDITY = 10**3;
    /// @dev transfer的selector编码
    bytes4 private constant SELECTOR =
        bytes4(keccak256(bytes("transfer(address,uint256)")));

    /// @notice 工厂合约地址
    address public factory;
    /// @dev token地址数组
    address public token0;
    address public token1;

    /// @notice 收税地址
    address public feeTo; // feeTo
    /// @notice 配对创建者
    address public creator; // creator
    /// @notice 交易钩子合约地址
    address public swapHookAddress;

    /// @dev token储备量
    uint112 private reserve0; // uses single storage slot, accessible via getReserves
    uint112 private reserve1; // uses single storage slot, accessible via getReserves
    /// @dev 更新储备量的最后时间戳
    uint32 private blockTimestampLast; // uses single storage slot, accessible via getReserves

    /// @notice 配对合约创建时间
    uint256 public birthday;
    /// @dev token价格最后累计
    uint256 public price0CumulativeLast;
    uint256 public price1CumulativeLast;
    /// @notice 在最近一次流动性事件之后的K值,储备量0*储备量1，自最近一次流动性事件发生后
    uint256 public kLast; // reserve0 * reserve1, as of immediately after the most recent liquidity e
    /// @notice 流动性相当于sqrt（k）增长的1/6
    uint8 public rootKmul = 5; // mint liquidity equivalent to 1/6th of the growth in sqrt(k)
    /// @notice 手续费占比0.3%
    uint8 public fee = 3; // 0.3%

    /// @dev 防止重入开关
    uint256 private unlocked = 1;

    /**
     * @dev 事件:铸造
     * @param sender 发送者
     * @param amount0 输入金额0
     * @param amount1 输入金额1
     */
    event Mint(address indexed sender, uint256 amount0, uint256 amount1);
    /**
     * @dev 事件:销毁
     * @param sender 发送者
     * @param amount0 输入金额0
     * @param amount1 输入金额1
```

```
     * @param to to地址
     */
    event Burn(
        address indexed sender,
        uint256 amount0,
        uint256 amount1,
        address indexed to
    );
    /**
     * @dev 事件:交换
     * @param sender 发送者
     * @param amount0In 输入金额0
     * @param amount1In 输入金额1
     * @param amount0Out 输出金额0
     * @param amount1Out 输出金额1
     * @param to to地址
     */
    event Swap(
        address indexed sender,
        uint256 amount0In,
        uint256 amount1In,
        uint256 amount0Out,
        uint256 amount1Out,
        address indexed to
    );
    /**
     * @dev 事件:同步
     * @param reserve0 储备量0
     * @param reserve1 储备量1
     */
    event Sync(uint112 reserve0, uint112 reserve1);

    /**
     * @dev 事件:收税地址更新
     * @param feeTo 收税地址
     */
    event FeeToUpdated(address indexed feeTo);

    /**
     * @dev 事件:K值乘数更新
     * @param rootKmul K值乘数
     */
    event RootKmulUpdated(uint8 rootKmul);

    /**
     * @dev 事件:交易钩子合约地址更新
     * @param swapHookAddress 交易钩子合约地址
     */
    event SwapHookUpdated(address swapHookAddress);

    /**
     * @dev 事件:收税比例更新
     * @param fee 收税比例
     */
    event FeeUpdated(uint8 fee);

    /**
     * @dev 修饰符:锁定运行防止重入
     */
    modifier lock() {
        require(unlocked == 1, "Swap: LOCKED");
        unlocked = 0;
        _;
        unlocked = 1;
    }
```

```
    /**
     * @dev 修饰符:确认必须为工厂合约的FeeToSetter地址
     */
    modifier onlyFeeToSetter() {
        // 确认必须为工厂合约的FeeToSetter地址
        require(
            msg.sender == ISwapFactory(factory).feeToSetter(),
            "Swap: FORBIDDEN"
        );
        _;
    }

    /**
     * @dev 获取储备
     * @return _reserve0 储备量0
     * @return _reserve1 储备量1
     * @return _blockTimestampLast 时间戳
     */
    function getReserves()
        public
        view
        returns (
            uint112 _reserve0,
            uint112 _reserve1,
            uint32 _blockTimestampLast
        )
    {
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }

    /**
     * @dev 私有安全发送
     * @param token token
     * @param to    to地址
     * @param value 数额
     */
    function _safeTransfer(
        address token,
        address to,
        uint256 value
    ) private {
        (bool success, bytes memory data) =
            token.call(abi.encodeWithSelector(SELECTOR, to, value));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            "UniswapV2: TRANSFER_FAILED"
        );
    }

    /**
     * @dev 构造函数
     */
    constructor() public {
        //factory地址为合约布署者
        factory = msg.sender;
        birthday = block.timestamp;
    }

    /**
     * @dev 初始化方法,部署时由工厂调用一次
     * @param _token0 token0
     * @param _token1 token1
     */
    function initialize(address _token0, address _token1) external {
```

```
        require(msg.sender == factory, "Swap: FORBIDDEN"); // sufficient check
        token0 = _token0;
        token1 = _token1;
        symbol = string(
            abi.encodePacked(
                "GLP:",
                IERC20Swap(_token0).symbol(),
                "-",
                IERC20Swap(_token1).symbol()
            )
        );
    }

    /**
     * @dev 设置收税地址
     * @param _feeTo 收税地址
     */
    function setFeeTo(address _feeTo) external onlyFeeToSetter {
        feeTo = _feeTo;
        emit FeeToUpdated(_feeTo);
    }

    /**
     * @dev 设置K值乘数
     * @param _rootKmul K值乘数
     */
    function setRootKmul(uint8 _rootKmul) external onlyFeeToSetter {
        rootKmul = _rootKmul;
        emit RootKmulUpdated(_rootKmul);
    }

    /**
     * @dev 设置收税比例
     * @param _fee 收税比例
     */
    function setFee(uint8 _fee) external onlyFeeToSetter {
        fee = _fee;
        emit FeeUpdated(fee);
    }

    /**
     * @dev 设置交易钩子合约地址
     * @param _swapHookAddress 交易钩子合约地址
     */
    function setSwapHook(address _swapHookAddress) external onlyFeeToSetter {
        swapHookAddress = _swapHookAddress;
        emit SwapHookUpdated(swapHookAddress);
    }

    /**
     * @dev 获取收税地址
     * @return 收税地址
     */
    function getFeeTo() public view returns (address) {
        // 如果feeTo地址不为0地址,以feeTo地址为准
        if (feeTo != address(0)) {
            return feeTo;
            // 否则如果配对合约创建30天之后,以工程合约的feeTo地址为准
        } else if (block.timestamp.sub(birthday) > 30 days) {
            return ISwapFactory(factory).feeTo();
            // 否者feeTo地址为配对合约创建者
        } else {
            return creator;
        }
    }
```

```
    /**
     * @dev 更新储量，并在每个区块的第一次调用时更新价格累加器
     * @param balance0 余额0
     * @param balance1  余额1
     * @param _reserve0 储备0
     * @param _reserve1 储备1
     */
    function _update(
        uint256 balance0,
        uint256 balance1,
        uint112 _reserve0,
        uint112 _reserve1
    ) private {
        //确认余额0和余额1小于等于最大的uint112
        require(
            balance0 <= uint112(-1) && balance1 <= uint112(-1),
            "Swap: OVERFLOW"
        );
        //区块时间戳,将时间戳转换为uint32
        uint32 blockTimestamp = uint32(block.timestamp % 2**32);
        //计算时间流逝
        uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is desired
        //如果时间流逝>0 并且 储备量0,1不等于0
        if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
            // * never overflows, and + overflow is desired
            //价格0最后累计 += 储备量1 * 2**112 / 储备量0 * 时间流逝
            price0CumulativeLast +=
                uint256(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) *
                timeElapsed;
            //价格1最后累计 += 储备量0 * 2**112 / 储备量1 * 时间流逝
            price1CumulativeLast +=
                uint256(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) *
                timeElapsed;
        }
        //余额0,1放入储备量0,1
        reserve0 = uint112(balance0);
        reserve1 = uint112(balance1);
        //更新最后时间戳
        blockTimestampLast = blockTimestamp;
        //触发同步事件
        emit Sync(reserve0, reserve1);
    }

    /**
     * @dev 默认情况下铸造流动性相当于1/6的增长sqrt（k）
     * @param _reserve0 储备0
     * @param _reserve1 储备1
     */
    function _mintFee(uint112 _reserve0, uint112 _reserve1)
        private
    {
        //定义k值
        uint256 _kLast = kLast; // gas savings
        //如果k值不等于0
        if (_kLast != 0 && rootKmul > 0) {
            //计算(_reserve0*_reserve1)的平方根
            uint256 rootK = Math.sqrt(uint256(_reserve0).mul(_reserve1));
            //计算k值的平方根
            uint256 rootKLast = Math.sqrt(_kLast);
            //如果rootK>rootKLast
            if (rootK > rootKLast) {
                //分子 = erc20总量 * (rootK - rootKLast)
                uint256 numerator = totalSupply.mul(rootK.sub(rootKLast));
                //分母 = rootK * 5 + rootKLast
                uint256 denominator = rootK.mul(rootKmul).add(rootKLast);
                //流动性 = 分子 / 分母
```

```
                uint256 liquidity = numerator / denominator;
                // 如果流动性 > 0 将流动性铸造给feeTo地址
                if (liquidity > 0) _mint(getFeeTo(), liquidity);
            }
        }
    }

    /**
     * @dev 铸造方法
     * @param to to地址
     * @return liquidity 流动性数量
     * @notice 应该从执行重要安全检查的合同中调用此低级功能
     */
    function mint(address to) external lock returns (uint256 liquidity) {
        //获取`储备量0`,`储备量1`
        (uint112 _reserve0, uint112 _reserve1, ) = getReserves(); // gas savings
        //获取当前合约在token0合约内的余额
        uint256 balance0 = IERC20(token0).balanceOf(address(this));
        //获取当前合约在token1合约内的余额
        uint256 balance1 = IERC20(token1).balanceOf(address(this));
        //amount0 = 余额0
        uint256 amount0 = balance0.sub(_reserve0);
        //amount1 = 余额1
        uint256 amount1 = balance1.sub(_reserve1);

        //调用铸造费方法
        _mintFee(_reserve0, _reserve1);
        //获取totalSupply,必须在此处定义，因为totalSupply可以在mintFee中更新
        uint256 _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply ca
        //如果_totalSupply等于0,首次创建配对
        if (_totalSupply == 0) {
            //流动性 = (数量0 * 数量1)的平方根 - 最小流动性1000
            liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
            //在总量为0的初始状态,永久锁定最低流动性
            _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY tok

            // 设置配对创建者为to地址
            creator = to; // set creator
        } else {
            //流动性 = 最小值 (amount0 * _totalSupply / _reserve0) 和 (amount1 * _totalSupply / _reserv
            liquidity = Math.min(
                amount0.mul(_totalSupply) / _reserve0,
                amount1.mul(_totalSupply) / _reserve1
            );
        }
        //确认流动性 > 0
        require(liquidity > 0, "Swap: INSUFFICIENT_LIQUIDITY_MINTED");
        //铸造流动性给to地址
        _mint(to, liquidity);

        //更新储备量
        _update(balance0, balance1, _reserve0, _reserve1);
        //k值 = 储备0 * 储备1,储备量中包含虚流动性,所以要再减去
        kLast = uint256(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
        //触发铸造事件
        emit Mint(msg.sender, amount0, amount1);
    }

    /**
     * @dev 销毁方法
     * @param to to地址
     * @return amount0
     * @return amount1
     * @notice 应该从执行重要安全检查的合同中调用此低级功能
     */
    function burn(address to)
```

```
        external
        lock
        returns (uint256 amount0, uint256 amount1)
    {
        //获取`储备量0`,`储备量1`
        (uint112 _reserve0, uint112 _reserve1, ) = getReserves();
        //获取当前合约在token0合约内的余额
        address _token0 = token0;
        uint256 balance0 = IERC20(_token0).balanceOf(address(this));
        //获取当前合约在token1合约内的余额
        address _token1 = token1;
        uint256 balance1 = IERC20(_token1).balanceOf(address(this));
        //从当前合约的balanceOf映射中获取当前合约自身的流动性数量,移除流动性的之前先将lptoken发送到配对合约
        uint256 liquidity = balanceOf[address(this)];

        //调用铸造费方法
        _mintFee(_reserve0, _reserve1);
        //获取totalSupply,必须在此处定义，因为totalSupply可以在mintFee中更新
        uint256 _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply ca
        //amount0 = 流动性数量 * 余额0 / totalSupply   使用余额确保按比例分配
        amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribu
        //amount1 = 流动性数量 * 余额1 / totalSupply   使用余额确保按比例分配
        amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribu
        //确认amount0和amount1都大于0
        require(
            amount0 > 0 && amount1 > 0,
            "Swap: INSUFFICIENT_LIQUIDITY_BURNED"
        );
        //销毁当前合约内的流动性数量
        _burn(address(this), liquidity);
        //将amount0数量的_token0发送给to地址
        _safeTransfer(_token0, to, amount0);
        //将amount1数量的_token1发送给to地址
        _safeTransfer(_token1, to, amount1);
        //更新balance0
        balance0 = IERC20(_token0).balanceOf(address(this));
        //更新balance1
        balance1 = IERC20(_token1).balanceOf(address(this));

        //更新储备量
        _update(balance0, balance1, _reserve0, _reserve1);
        //k值 = 储备0 * 储备1,储备量中包含虚流动性,所以要再减去
        kLast = uint256(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
        //触发销毁事件
        emit Burn(msg.sender, amount0, amount1, to);
    }

    /**
     * @dev 交换方法
     * @param amount0Out 输出数额0
     * @param amount1Out 输出数额1
     * @param to    to地址
     * @param data  用于回调的数据
     * @notice 应该从执行重要安全检查的合同中调用此低级功能
     */
    function swap(
        uint256 amount0Out,
        uint256 amount1Out,
        address to,
        bytes calldata data
    ) external lock {
        //确认amount0Out和amount1Out都大于0
        require(
            amount0Out > 0 || amount1Out > 0,
            "Swap: INSUFFICIENT_OUTPUT_AMOUNT"
        );
```

```
        //获取`储备量0`,`储备量1`
        (uint112 _reserve0, uint112 _reserve1, ) = getReserves(); // gas savings
        //确认`输出数量0,1` < `储备量0,1`
        require(
            amount0Out < _reserve0 && amount1Out < _reserve1,
            "Swap: INSUFFICIENT_LIQUIDITY"
        );

        //初始化变量
        uint256 balance0;
        uint256 balance1;
        {
            //标记_token{0,1}的作用域，避免堆栈太深的错误
            // scope for _token{0,1}, avoids stack too deep errors
            address _token0 = token0;
            address _token1 = token1;
            //确认to地址不等于_token0和_token1
            require(to != _token0 && to != _token1, "Swap: INVALID_TO");
            //如果`输出数量0` > 0 安全发送`输出数量0`的token0到to地址
            if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer to
            //如果`输出数量1` > 0 安全发送`输出数量1`的token1到to地址
            if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer to
            //如果data的长度大于0 调用to地址的接口,闪电贷!
            if (data.length > 0)
                ISwapCallee(to).swapCall(
                    msg.sender,
                    amount0Out,
                    amount1Out,
                    data
                );
            //调用交易钩子
            if (swapHookAddress != address(0))
                ISwapHook(swapHookAddress).swapHook(
                    msg.sender,
                    amount0Out,
                    amount1Out,
                    to
                );
            //获取当前合约在token0合约内的余额
            balance0 = IERC20(_token0).balanceOf(address(this));
            //获取当前合约在token1合约内的余额
            balance1 = IERC20(_token1).balanceOf(address(this));
        }
        //如果 余额0 > 储备0 - amount0Out 则 amount0In = 余额0 - (储备0 - amount0Out) 否则 amount0In = 0
        uint256 amount0In =
            balance0 > _reserve0 - amount0Out
                ? balance0 - (_reserve0 - amount0Out)
                : 0;
        //如果 余额1 > 储备1 - amount1Out 则 amount1In = 余额1 - (储备1 - amount1Out) 否则 amount1In = 0
        uint256 amount1In =
            balance1 > _reserve1 - amount1Out
                ? balance1 - (_reserve1 - amount1Out)
                : 0;
        //确认`输入数量0||1`大于0
        require(
            amount0In > 0 || amount1In > 0,
            "Swap: INSUFFICIENT_INPUT_AMOUNT"
        );
        {
            //标记reserve{0,1}的作用域，避免堆栈太深的错误
            // scope for reserve{0,1}Adjusted, avoids stack too deep errors
            //调整后的余额0 = 余额0 * 1000 - (amount0In * fee)
            uint256 balance0Adjusted =
                balance0.mul(1000).sub(amount0In.mul(fee));
            //调整后的余额1 = 余额1 * 1000 - (amount1In * fee)
            uint256 balance1Adjusted =
```

```
                balance1.mul(1000).sub(amount1In.mul(fee));
            //确认balance0Adjusted * balance1Adjusted >= 储备0 * 储备1 * 1000000
            require(
                balance0Adjusted.mul(balance1Adjusted) >=
                    uint256(_reserve0).mul(_reserve1).mul(1000**2),
                "Swap: K"
            );
        }

        //更新储备量
        _update(balance0, balance1, _reserve0, _reserve1);
        //触发交换事件
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }

    /**
     * @dev 强制平衡以匹配储备
     * @param to to地址
     */
    function skim(address to) external lock {
        //将当前合约在`token0,1`的余额-`储备量0,1`安全发送到to地址
        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)
        );
    }

    /**
     * @dev 强制准备金与余额匹配
     */
    function sync() external lock {
        _update(
            IERC20(token0).balanceOf(address(this)),
            IERC20(token1).balanceOf(address(this)),
            reserve0,
            reserve1
        );
    }
}

/**
 * @title Swap工厂合约
 */
contract SwapFactory is ISwapFactory {
    /// @notice 收税地址
    address public override feeTo;
    /// @notice 收税权限控制地址,应为治理地址
    address public override feeToSetter;
    /// @notice 配对映射,地址=>(地址=>地址)
    mapping(address => mapping(address => address)) public override getPair;
    /// @notice 所有配对数组
    address[] public override allPairs;

    /**
     * @dev 事件:创建配对
     * @param token0 token0
     * @param token1 token1
     * @param pair 配对地址
```

```
     */
    event PairCreated(
        address indexed token0,
        address indexed token1,
        address pair,
        uint256
    );

    /**
     * @dev 构造函数
     */
    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
    }

    /**
     * @dev 查询配对数组长度方法
     */
    function allPairsLength() external view override returns (uint256) {
        return allPairs.length;
    }

    /**
     * @dev  配对合约源代码Bytecode的hash值(用作前端计算配对合约地址)
     */
    function pairCodeHash() external pure override returns (bytes32) {
        return keccak256(type(SwapPair).creationCode);
    }

    /**
     * @dev 创建配对
     * @param tokenA TokenA
     * @param tokenB TokenB
     * @return pair 配对地址
     * @notice 应该从路由合约调用配对寻找工厂合约来调用,否则通过路由合约找不到配对合约
     */
    function createPair(address tokenA, address tokenB)
        external
        override
        returns (address pair)
    {
        //确认tokenA不等于tokenB
        require(tokenA != tokenB, "Swap: IDENTICAL_ADDRESSES");
        //将tokenA和tokenB进行大小排序,确保tokenA小于tokenB
        (address token0, address token1) =
            tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        //确认token0不等于0地址
        require(token0 != address(0), "Swap: ZERO_ADDRESS");
        //确认配对映射中不存在token0=>token1
        require(getPair[token0][token1] == address(0), "Swap: PAIR_EXISTS"); // single check is suffi
        //给bytecode变量赋值"SwapPair"合约的创建字节码
        bytes memory bytecode = type(SwapPair).creationCode;
        //将token0和token1打包后创建哈希
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        //内联汇编
        //solium-disable-next-line
        assembly {
            //通过create2方法布署合约,并且加盐,返回地址到pair变量
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        //调用pair地址的合约中的"initialize"方法,传入变量token0,token1
        SwapPair(pair).initialize(token0, token1);
        //配对映射中设置token0=>token1=pair
        getPair[token0][token1] = pair;
        //配对映射中设置token1=>token0=pair
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
```

```
        //配对数组中推入pair地址
        allPairs.push(pair);
        //触发配对成功事件
        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    /**
     * @dev 修饰符:确认必须为工厂合约的FeeToSetter地址
     */
    modifier onlyFeeToSetter() {
        // 确认必须为工厂合约的FeeToSetter地址
        require(msg.sender == feeToSetter, "Swap: FORBIDDEN");
        _;
    }

    /**
     * @dev 设置收税地址
     * @param _feeTo 收税地址
     */
    function setFeeTo(address _feeTo) external override onlyFeeToSetter {
        feeTo = _feeTo;
    }

    /**
     * @dev 设置收税权限控制
     * @param _feeToSetter 收税权限控制
     */
    function setFeeToSetter(address _feeToSetter)
        external
        override
        onlyFeeToSetter
    {
        feeToSetter = _feeToSetter;
    }
}

// SPDX-License-Identifier: MIT
pragma solidity ^0.6.0;

interface IERC20 {
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
    event Transfer(address indexed from, address indexed to, uint256 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 (uint256);

    function balanceOf(address owner) external view returns (uint256);

    function allowance(address owner, address spender)
        external
        view
        returns (uint256);

    function approve(address spender, uint256 value) external returns (bool);

    function transfer(address to, uint256 value) external returns (bool);
```

```
    function transferFrom(
        address from,
        address to,
        uint256 value
    ) external returns (bool);
}

library SafeMath {
    function add(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x + y) >= x, "ds-math-add-overflow");
    }

    function sub(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require((z = x - y) <= x, "ds-math-sub-underflow");
    }

    function mul(uint256 x, uint256 y) internal pure returns (uint256 z) {
        require(y == 0 || (z = x * y) / y == x, "ds-math-mul-overflow");
    }
}

interface ISwapPair {
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
    event Transfer(address indexed from, address indexed to, uint256 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 (uint256);

    function balanceOf(address owner) external view returns (uint256);

    function allowance(address owner, address spender)
        external
        view
        returns (uint256);

    function approve(address spender, uint256 value) external returns (bool);

    function transfer(address to, uint256 value) external returns (bool);

    function transferFrom(
        address from,
        address to,
        uint256 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 (uint256);

    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
```

```
        bytes32 r,
        bytes32 s
    ) external;

    event Mint(address indexed sender, uint256 amount0, uint256 amount1);
    event Burn(
        address indexed sender,
        uint256 amount0,
        uint256 amount1,
        address indexed to
    );
    event Swap(
        address indexed sender,
        uint256 amount0In,
        uint256 amount1In,
        uint256 amount0Out,
        uint256 amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint256);

    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 (uint256);

    function price1CumulativeLast() external view returns (uint256);

    function kLast() external view returns (uint256);

    function mint(address to) external returns (uint256 liquidity);

    function burn(address to)
        external
        returns (uint256 amount0, uint256 amount1);

    function swap(
        uint256 amount0Out,
        uint256 amount1Out,
        address to,
        bytes calldata data
    ) external;

    function skim(address to) external;

    function sync() external;

    function fee() external view returns (uint8);

    function feeTo() external view returns (address);

    function getFeeTo() external view returns (address);
```

```
    function creator() external view returns (address);

    function birthday() external view returns (uint256);

    function rootKmul() external view returns (uint8);

    function initialize(address, address) external;

    function setFeeTo(address) external;

    function setrootKmul(uint8) external;

    function setFee(uint8) external;
}

interface ISwapFactory {
    event PairCreated(
        address indexed token0,
        address indexed token1,
        address pair,
        uint256
    );

    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(uint256) external view returns (address pair);

    function allPairsLength() external view returns (uint256);

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

    function setFeeTo(address) external;

    function setFeeToSetter(address) external;

    function pairCodeHash() external pure returns (bytes32);
}

/**
 * @title Swap库合约
 */
library SwapLibrary {
    using SafeMath for uint256;

    /**
     * @dev 排序token地址
     * @notice 返回排序的令牌地址，用于处理按此顺序排序的对中的返回值
     * @param tokenA TokenA
     * @param tokenB TokenB
     * @return token0  Token0
     * @return token1  Token1
     */
    function sortTokens(address tokenA, address tokenB)
        internal
        pure
        returns (address token0, address token1)
```

```
    {
        //确认tokenA不等于tokenB
        require(tokenA != tokenB, "SwapLibrary: IDENTICAL_ADDRESSES");
        //排序token地址
        (token0, token1) = tokenA < tokenB
            ? (tokenA, tokenB)
            : (tokenB, tokenA);
        //确认token地址不等于0地址
        require(token0 != address(0), "SwapLibrary: ZERO_ADDRESS");
    }

    /**
     * @dev 获取pair合约地址
     * @notice 计算一对的CREATE2地址，而无需进行任何外部调用
     * @param factory 工厂地址
     * @param tokenA TokenA
     * @param tokenB TokenB
     * @return pair  pair合约地址
     */
    function pairFor(
        address factory,
        address tokenA,
        address tokenB
    ) internal pure returns (address pair) {
        //排序token地址
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        // 获取pairCodeHash
        bytes32 pairCodeHash = ISwapFactory(factory).pairCodeHash();
        //根据排序的token地址计算create2的pair地址
        pair = address(
            uint256(
                keccak256(
                    abi.encodePacked(
                        hex"ff",
                        factory,
                        keccak256(abi.encodePacked(token0, token1)),
                        pairCodeHash // init code hash
                    )
                )
            )
        );
    }

    /**
     * @dev 获取储备量
     * @notice 提取并排序一对的储备金
     * @param factory 工厂地址
     * @param tokenA TokenA
     * @param tokenB TokenB
     * @return reserveA  储备量A
     * @return reserveB  储备量B
     * @return fee  手续费
     */
    function getReserves(
        address factory,
        address tokenA,
        address tokenB
    )
        internal
        view
        returns (
            uint256 reserveA,
            uint256 reserveB,
            uint8 fee
        )
    {
```

```
        //排序token地址
        (address token0, ) = sortTokens(tokenA, tokenB);
        //通过排序后的token地址和工厂合约地址获取到pair合约地址,并从pair合约中获取储备量0,1
        (uint256 reserve0, uint256 reserve1, ) =
            ISwapPair(pairFor(factory, tokenA, tokenB)).getReserves();
        //根据输入的token顺序返回储备量
        (reserveA, reserveB) = tokenA == token0
            ? (reserve0, reserve1)
            : (reserve1, reserve0);
        //获取配对合约中设置的手续费比例
        fee = ISwapPair(pairFor(factory, tokenA, tokenB)).fee();
    }

    /**
     * @dev 对价计算
     * @notice 给定一定数量的资产和货币对储备金，则返回等值的其他资产
     * @param amountA 数额A
     * @param reserveA 储备量A
     * @param reserveB 储备量B
     * @return amountB  数额B
     */
    function quote(
        uint256 amountA,
        uint256 reserveA,
        uint256 reserveB
    ) internal pure returns (uint256 amountB) {
        //确认数额A>0
        require(amountA > 0, "SwapLibrary: INSUFFICIENT_AMOUNT");
        //确认储备量A,B大于0
        require(
            reserveA > 0 && reserveB > 0,
            "SwapLibrary: INSUFFICIENT_LIQUIDITY"
        );
        //数额B = 数额A * 储备量B / 储备量A
        amountB = amountA.mul(reserveB) / reserveA;
    }

    /**
     * @dev 获取单个输出数额
     * @notice 给定一项资产的输入量和配对的储备，返回另一项资产的最大输出量
     * @param amountIn 输入数额
     * @param reserveIn 储备量In
     * @param reserveOut 储备量Out
     * @param fee 手续费比例
     * @return amountOut  输出数额
     */
    function getAmountOut(
        uint256 amountIn,
        uint256 reserveIn,
        uint256 reserveOut,
        uint8 fee
    ) internal pure returns (uint256 amountOut) {
        //确认输入数额大于0
        require(amountIn > 0, "SwapLibrary: INSUFFICIENT_INPUT_AMOUNT");
        //确认储备量In和储备量Out大于0
        require(
            reserveIn > 0 && reserveOut > 0,
            "SwapLibrary: INSUFFICIENT_LIQUIDITY"
        );
        //税后输入数额 = 输入数额 * (1000-fee)
        uint256 amountInWithFee = amountIn.mul(1000 - fee);
        //分子 = 税后输入数额 * 储备量Out
        uint256 numerator = amountInWithFee.mul(reserveOut);
        //分母 = 储备量In * 1000 + 税后输入数额
        uint256 denominator = reserveIn.mul(1000).add(amountInWithFee);
        //输出数额 = 分子 / 分母
```

```
        amountOut = numerator / denominator;
    }

    /**
     * @dev 获取单个输出数额
     * @notice 给定一项资产的输出量和对储备，返回其他资产的所需输入量
     * @param amountOut 输出数额
     * @param reserveIn 储备量In
     * @param reserveOut 储备量Out
     * @param fee 手续费比例
     * @return amountIn  输入数额
     */
    function getAmountIn(
        uint256 amountOut,
        uint256 reserveIn,
        uint256 reserveOut,
        uint8 fee
    ) internal pure returns (uint256 amountIn) {
        //确认输出数额大于0
        require(amountOut > 0, "SwapLibrary: INSUFFICIENT_OUTPUT_AMOUNT");
        //确认储备量In和储备量Out大于0
        require(
            reserveIn > 0 && reserveOut > 0,
            "SwapLibrary: INSUFFICIENT_LIQUIDITY"
        );
        //分子 = 储备量In * 储备量Out * 1000
        uint256 numerator = reserveIn.mul(amountOut).mul(1000);
        //分母 = 储备量Out - 输出数额 * (1000-fee)
        uint256 denominator = reserveOut.sub(amountOut).mul(1000 - fee);
        //输入数额 = (分子 / 分母) + 1
        amountIn = (numerator / denominator).add(1);
    }

    /**
     * @dev 获取输出数额
     * @notice 对任意数量的对执行链接的getAmountOut计算
     * @param factory 工厂合约地址
     * @param amountIn 输入数额
     * @param path 路径数组
     * @return amounts  数额数组
     */
    function getAmountsOut(
        address factory,
        uint256 amountIn,
        address[] memory path
    ) internal view returns (uint256[] memory amounts) {
        //确认路径数组长度大于2
        require(path.length >= 2, "SwapLibrary: INVALID_PATH");
        //初始化数额数组
        amounts = new uint256[](path.length);
        //数额数组[0] = 输入数额
        amounts[0] = amountIn;
        //遍历路径数组,path长度-1
        for (uint256 i; i < path.length - 1; i++) {
            //(储备量In,储备量Out,手续费比例) = 获取储备(当前路径地址,下一个路径地址)
            (uint256 reserveIn, uint256 reserveOut, uint8 fee) =
                getReserves(factory, path[i], path[i + 1]);
            //下一个数额 = 获取输出数额(当前数额,储备量In,储备量Out)
            amounts[i + 1] = getAmountOut(
                amounts[i],
                reserveIn,
                reserveOut,
                fee
            );
        }
    }
```

```
    /**
     * @dev 获取输出数额
     * @notice 对任意数量的对执行链接的getAmountIn计算
     * @param factory 工厂合约地址
     * @param amountOut 输出数额
     * @param path 路径数组
     * @return amounts  数额数组
     */
    function getAmountsIn(
        address factory,
        uint256 amountOut,
        address[] memory path
    ) internal view returns (uint256[] memory amounts) {
        //确认路径数组长度大于2
        require(path.length >= 2, "SwapLibrary: INVALID_PATH");
        //初始化数额数组
        amounts = new uint256[](path.length);
        //数额数组最后一个元素 = 输出数额
        amounts[amounts.length - 1] = amountOut;
        //从倒数第二个元素倒叙遍历路径数组
        for (uint256 i = path.length - 1; i > 0; i--) {
            //(储备量In,储备量Out,手续费比例) = 获取储备(上一个路径地址,当前路径地址)
            (uint256 reserveIn, uint256 reserveOut, uint8 fee) =
                getReserves(factory, path[i - 1], path[i]);
            //上一个数额 = 获取输入数额(当前数额,储备量In,储备量Out)
            amounts[i - 1] = getAmountIn(
                amounts[i],
                reserveIn,
                reserveOut,
                fee
            );
        }
    }
}

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: 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: 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: TRANSFER_FROM_FAILED"
        );
    }

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

interface ISwapRouter {
    function factory() external view returns (address);

    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        external
        returns (
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        );

    function addLiquidityETH(
        address token,
        uint256 amountTokenDesired,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    )
        external
        payable
        returns (
            uint256 amountToken,
            uint256 amountETH,
            uint256 liquidity
        );

    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
```

```
    ) external returns (uint256 amountA, uint256 amountB);

    function removeLiquidityETH(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountToken, uint256 amountETH);

    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountA, uint256 amountB);

    function removeLiquidityETHWithPermit(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountToken, uint256 amountETH);

    function swapExactTokensForTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);

    function swapTokensForExactTokens(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);

    function swapExactETHForTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable returns (uint256[] memory amounts);

    function swapTokensForExactETH(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
```

```
        uint256 deadline
    ) external returns (uint256[] memory amounts);

    function swapExactTokensForETH(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external returns (uint256[] memory amounts);

    function swapETHForExactTokens(
        uint256 amountOut,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable returns (uint256[] memory amounts);

    function quote(
        uint256 amountA,
        uint256 reserveA,
        uint256 reserveB
    ) external pure returns (uint256 amountB);

    function getAmountOut(
        uint256 amountIn,
        uint256 reserveIn,
        uint256 reserveOut,
        uint8 fee
    ) external pure returns (uint256 amountOut);

    function getAmountIn(
        uint256 amountOut,
        uint256 reserveIn,
        uint256 reserveOut,
        uint8 fee
    ) external pure returns (uint256 amountIn);

    function getAmountsOut(uint256 amountIn, address[] calldata path)
        external
        view
        returns (uint256[] memory amounts);

    function getAmountsIn(uint256 amountOut, address[] calldata path)
        external
        view
        returns (uint256[] memory amounts);

    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) external returns (uint256 amountETH);

    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
```

```
        bytes32 r,
        bytes32 s
    ) external returns (uint256 amountETH);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;

    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable;

    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external;
}

interface IERC20Swap {
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );
    event Transfer(address indexed from, address indexed to, uint256 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 (uint256);

    function balanceOf(address owner) external view returns (uint256);

    function allowance(address owner, address spender)
        external
        view
        returns (uint256);

    function approve(address spender, uint256 value) external returns (bool);

    function transfer(address to, uint256 value) external returns (bool);

    function transferFrom(
        address from,
        address to,
        uint256 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 (uint256);
```

```
    function permit(
        address owner,
        address spender,
        uint256 value,
        uint256 deadline,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external;
}

interface IWETH {
    function deposit() external payable;

    function transfer(address to, uint256 value) external returns (bool);

    function withdraw(uint256) external;
}

/**
 * @title Swap 路由合约
 */
contract SwapRouter is ISwapRouter {
    using SafeMath for uint256;

    /// @notice 布署时定义的常量factory地址和WETH地址
    address public immutable override factory;
    address public immutable override WETH;

    /**
     * @dev 修饰符:确保最后期限大于当前时间
     */
    modifier ensure(uint256 deadline) {
        require(deadline >= block.timestamp, "SwapRouter: EXPIRED");
        _;
    }

    /**
     * @dev 构造函数
     * @param _factory 工厂合约地址
     * @param _WETH WETH合约地址
     */
    constructor(address _factory, address _WETH) public {
        factory = _factory;
        WETH = _WETH;
    }

    /**
     * @dev 收款方法
     */
    receive() external payable {
        //断言调用者为WETH合约地址
        assert(msg.sender == WETH); // only accept ETH via fallback from the WETH contract
    }

    // **** 添加流动性 ****
    /**
     * @dev 添加流动性的私有方法
     * @param tokenA tokenA地址
     * @param tokenB tokenB地址
     * @param amountADesired 期望数量A
     * @param amountBDesired 期望数量B
     * @param amountAMin 最小数量A
     * @param amountBMin 最小数量B
     * @return amountA   数量A
```

```
     * @return amountB   数量B
     */
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin
    ) internal virtual returns (uint256 amountA, uint256 amountB) {
        //如果工厂合约不存在,则创建配对
        if (ISwapFactory(factory).getPair(tokenA, tokenB) == address(0)) {
            ISwapFactory(factory).createPair(tokenA, tokenB);
        }
        //获取不含虚流动性的储备量reserve{A,B}
        (uint256 reserveA, uint256 reserveB, ) =
            SwapLibrary.getReserves(factory, tokenA, tokenB);
        //如果储备reserve{A,B}==0
        if (reserveA == 0 && reserveB == 0) {
            //数量amount{A,B} = 期望数量A,B
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            //最优数量B = 期望数量A * 储备B / 储备A
            uint256 amountBOptimal =
                SwapLibrary.quote(amountADesired, reserveA, reserveB);
            //如果最优数量B <= 期望数量B
            if (amountBOptimal <= amountBDesired) {
                //确认最优数量B >= 最小数量B
                require(
                    amountBOptimal >= amountBMin,
                    "SwapRouter: INSUFFICIENT_B_AMOUNT"
                );
                //数量amount{A,B} = 期望数量A, 最优数量B
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                //最优数量A = 期望数量A * 储备A / 储备B
                uint256 amountAOptimal =
                    SwapLibrary.quote(amountBDesired, reserveB, reserveA);
                //断言最优数量A <= 期望数量A
                assert(amountAOptimal <= amountADesired);
                //确认最优数量A >= 最小数量A
                require(
                    amountAOptimal >= amountAMin,
                    "SwapRouter: INSUFFICIENT_A_AMOUNT"
                );
                //数量amount{A,B} = 最优数量A, 期望数量B
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }

    /**
     * @dev 添加流动性方法*
     * @param tokenA tokenA地址
     * @param tokenB tokenB地址
     * @param amountADesired 期望数量A
     * @param amountBDesired 期望数量B
     * @param amountAMin 最小数量A
     * @param amountBMin 最小数量B
     * @param to to地址
     * @param deadline 最后期限
     * @return amountA   数量A
     * @return amountB   数量B
     * @return liquidity   流动性数量
     */
    function addLiquidity(
```

```
        address tokenA,
        address tokenB,
        uint256 amountADesired,
        uint256 amountBDesired,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (
            uint256 amountA,
            uint256 amountB,
            uint256 liquidity
        )
    {
        //添加流动性,获取数量A,数量B
        (amountA, amountB) = _addLiquidity(
            tokenA,
            tokenB,
            amountADesired,
            amountBDesired,
            amountAMin,
            amountBMin
        );
        //根据TokenA,TokenB地址,获取`pair合约`地址
        address pair = SwapLibrary.pairFor(factory, tokenA, tokenB);
        //将数量为amountA的tokenA从msg.sender账户中安全发送到pair合约地址
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        //将数量为amountB的tokenB从msg.sender账户中安全发送到pair合约地址
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        //流动性数量 = pair合约的铸造方法铸造给to地址的返回值
        liquidity = ISwapPair(pair).mint(to);
    }

    /**
     * @dev 添加ETH流动性方法*
     * @param token token地址
     * @param amountTokenDesired Token期望数量
     * @param amountTokenMin Token最小数量
     * @param amountETHMin ETH最小数量
     * @param to to地址
     * @param deadline 最后期限
     * @return amountToken   Token数量
     * @return amountETH   ETH数量
     * @return liquidity   流动性数量
     */
    function addLiquidityETH(
        address token,
        uint256 amountTokenDesired,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    )
        external
        payable
        virtual
        override
        ensure(deadline)
        returns (
            uint256 amountToken,
            uint256 amountETH,
```

```
            uint256 liquidity
        )
    {
        //添加流动性,获取Token数量,ETH数量
        (amountToken, amountETH) = _addLiquidity(
            token,
            WETH,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountETHMin
        );
        //根据Token,WETH地址,获取`pair合约`地址
        address pair = SwapLibrary.pairFor(factory, token, WETH);
        //将`Token数量`的token从msg.sender账户中安全发送到`pair合约`地址
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        //向`ETH合约`存款`ETH数量`的主币
        IWETH(WETH).deposit{value: amountETH}();
        //将`ETH数量`的`ETH`token发送到`pair合约`地址
        assert(IWETH(WETH).transfer(pair, amountETH));
        //流动性数量 = pair合约的铸造方法铸造给`to地址`的返回值
        liquidity = ISwapPair(pair).mint(to);
        //如果`收到的主币数量`>`ETH数量` 则返还`收到的主币数量`-`ETH数量`
        if (msg.value > amountETH)
            TransferHelper.safeTransferETH(msg.sender, msg.value - amountETH);
    }

    // **** 移除流动性 ****
    /**
     * @dev 移除流动性*
     * @param tokenA tokenA地址
     * @param tokenB tokenB地址
     * @param liquidity 流动性数量
     * @param amountAMin 最小数量A
     * @param amountBMin 最小数量B
     * @param to to地址
     * @param deadline 最后期限
     * @return amountA   数量A
     * @return amountB   数量B
     */
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline
    )
        public
        virtual
        override
        ensure(deadline)
        returns (uint256 amountA, uint256 amountB)
    {
        //计算TokenA,TokenB的CREATE2地址，而无需进行任何外部调用
        address pair = SwapLibrary.pairFor(factory, tokenA, tokenB);
        //将流动性数量从用户发送到pair地址(需提前批准)
        ISwapPair(pair).transferFrom(msg.sender, pair, liquidity);
        //pair合约销毁流动性数量,并将数值0,1的token发送到to地址
        (uint256 amount0, uint256 amount1) = ISwapPair(pair).burn(to);
        //排序tokenA,tokenB
        (address token0, ) = SwapLibrary.sortTokens(tokenA, tokenB);
        //按排序后的token顺序返回数值AB
        (amountA, amountB) = tokenA == token0
            ? (amount0, amount1)
```

```
            : (amount1, amount0);
        //确保数值AB大于最小值AB
        require(amountA >= amountAMin, "SwapRouter: INSUFFICIENT_A_AMOUNT");
        require(amountB >= amountBMin, "SwapRouter: INSUFFICIENT_B_AMOUNT");
    }

    /**
     * @dev 移除ETH流动性*
     * @param token token地址
     * @param liquidity 流动性数量
     * @param amountTokenMin token最小数量
     * @param amountETHMin ETH最小数量
     * @param to to地址
     * @param deadline 最后期限
     * @return amountToken   token数量
     * @return amountETH   ETH数量
     */
    function removeLiquidityETH(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    )
        public
        virtual
        override
        ensure(deadline)
        returns (uint256 amountToken, uint256 amountETH)
    {
        //(token数量,ETH数量) = 移除流动性(token地址,WETH地址,流动性数量,token最小数量,ETH最小数量,当前合约地
        (amountToken, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        //将token数量的token发送到to地址
        TransferHelper.safeTransfer(token, to, amountToken);
        //从WETH取款ETH数量的主币
        IWETH(WETH).withdraw(amountETH);
        //将ETH数量的ETH发送到to地址
        TransferHelper.safeTransferETH(to, amountETH);
    }

    /**
     * @dev 带签名移除流动性*
     * @param tokenA tokenA地址
     * @param tokenB tokenB地址
     * @param liquidity 流动性数量
     * @param amountAMin 最小数量A
     * @param amountBMin 最小数量B
     * @param to to地址
     * @param deadline 最后期限
     * @param approveMax 全部批准
     * @param v v
     * @param r r
     * @param s s
     * @return amountA   数量A
     * @return amountB   数量B
     */
    function removeLiquidityWithPermit(
```

```
        address tokenA,
        address tokenB,
        uint256 liquidity,
        uint256 amountAMin,
        uint256 amountBMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external virtual override returns (uint256 amountA, uint256 amountB) {
        //计算TokenA,TokenB的CREATE2地址，而无需进行任何外部调用
        address pair = SwapLibrary.pairFor(factory, tokenA, tokenB);
        //如果全部批准,value值等于最大uint256,否则等于流动性
        uint256 value = approveMax ? uint256(-1) : liquidity;
        //调用pair合约的许可方法(调用账户,当前合约地址,数值,最后期限,v,r,s)
        ISwapPair(pair).permit(
            msg.sender,
            address(this),
            value,
            deadline,
            v,
            r,
            s
        );
        //(数量A,数量B) = 移除流动性(tokenA地址,tokenB地址,流动性数量,最小数量A,最小数量B,to地址,最后期限)
        (amountA, amountB) = removeLiquidity(
            tokenA,
            tokenB,
            liquidity,
            amountAMin,
            amountBMin,
            to,
            deadline
        );
    }

    /**
     * @dev 带签名移除ETH流动性*
     * @param token token地址
     * @param liquidity 流动性数量
     * @param amountTokenMin token最小数量
     * @param amountETHMin ETH最小数量
     * @param to to地址
     * @param deadline 最后期限
     * @param approveMax 全部批准
     * @param v v
     * @param r r
     * @param s s
     * @return amountToken   token数量
     * @return amountETH   ETH数量
     */
    function removeLiquidityETHWithPermit(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    )
        external
```

```
        virtual
        override
        returns (uint256 amountToken, uint256 amountETH)
    {
        //计算Token,WETH的CREATE2地址，而无需进行任何外部调用
        address pair = SwapLibrary.pairFor(factory, token, WETH);
        //如果全部批准,value值等于最大uint256,否则等于流动性
        uint256 value = approveMax ? uint256(-1) : liquidity;
        //调用pair合约的许可方法(调用账户,当前合约地址,数值,最后期限,v,r,s)
        ISwapPair(pair).permit(
            msg.sender,
            address(this),
            value,
            deadline,
            v,
            r,
            s
        );
        //(token数量,ETH数量) = 移除ETH流动性(token地址,流动性数量,token最小数量,ETH最小数量,to地址,最后期限)
        (amountToken, amountETH) = removeLiquidityETH(
            token,
            liquidity,
            amountTokenMin,
            amountETHMin,
            to,
            deadline
        );
    }

    /**
     * @dev 移除流动性支持Token收转帐税*
     * @param token token地址
     * @param liquidity 流动性数量
     * @param amountTokenMin token最小数量
     * @param amountETHMin ETH最小数量
     * @param to to地址
     * @param deadline 最后期限
     * @return amountETH   ETH数量
     */
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline
    ) public virtual override ensure(deadline) returns (uint256 amountETH) {
        //(,ETH数量) = 移除流动性(token地址,WETH地址,流动性数量,token最小数量,ETH最小数量,当前合约地址,最后期限)
        (, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        //将当前合约中的token数量的token发送到to地址
        TransferHelper.safeTransfer(
            token,
            to,
            IERC20(token).balanceOf(address(this))
        );
        //从WETH取款ETH数量的主币
        IWETH(WETH).withdraw(amountETH);
        //将ETH数量的ETH发送到to地址
```

```
        TransferHelper.safeTransferETH(to, amountETH);
    }

    /**
     * @dev 带签名移除流动性,支持Token收转帐税*
     * @param token token地址
     * @param liquidity 流动性数量
     * @param liquidity 流动性数量
     * @param amountTokenMin token最小数量
     * @param amountETHMin ETH最小数量
     * @param to to地址
     * @param deadline 最后期限
     * @param approveMax 全部批准
     * @param v v
     * @param r r
     * @param s s
     * @return amountETH   ETH数量
     */
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint256 liquidity,
        uint256 amountTokenMin,
        uint256 amountETHMin,
        address to,
        uint256 deadline,
        bool approveMax,
        uint8 v,
        bytes32 r,
        bytes32 s
    ) external virtual override returns (uint256 amountETH) {
        //计算Token,WETH的CREATE2地址，而无需进行任何外部调用
        address pair = SwapLibrary.pairFor(factory, token, WETH);
        //如果全部批准,value值等于最大uint256,否则等于流动性
        uint256 value = approveMax ? uint256(-1) : liquidity;
        //调用pair合约的许可方法(调用账户,当前合约地址,数值,最后期限,v,r,s)
        ISwapPair(pair).permit(
            msg.sender,
            address(this),
            value,
            deadline,
            v,
            r,
            s
        );
        //(,ETH数量) = 移除流动性支持Token收转帐税(token地址,流动性数量,Token最小数量,ETH最小数量,to地址,最后
        amountETH = removeLiquidityETHSupportingFeeOnTransferTokens(
            token,
            liquidity,
            amountTokenMin,
            amountETHMin,
            to,
            deadline
        );
    }

    // **** 交换 ****
    /**
     * @dev 私有交换*
     * @notice 要求初始金额已经发送到第一对
     * @param amounts 数额数组
     * @param path 路径数组
     * @param _to to地址
     */
    function _swap(
        uint256[] memory amounts,
        address[] memory path,
```

```
        address _to
    ) internal virtual {
        //遍历路径数组
        for (uint256 i; i < path.length - 1; i++) {
            //(输入地址,输出地址) = (当前地址,下一个地址)
            (address input, address output) = (path[i], path[i + 1]);
            //token0 = 排序(输入地址,输出地址)
            (address token0, ) = SwapLibrary.sortTokens(input, output);
            //输出数量 = 数额数组下一个数额
            uint256 amountOut = amounts[i + 1];
            //(输出数额0,输出数额1) = 输入地址==token0 ? (0,输出数额) : (输出数额,0)
            (uint256 amount0Out, uint256 amount1Out) =
                input == token0
                    ? (uint256(0), amountOut)
                    : (amountOut, uint256(0));
            //to地址 = i<路径长度-2 ? (输出地址,路径下下个地址)的pair合约地址 : to地址
            address to =
                i < path.length - 2
                    ? SwapLibrary.pairFor(factory, output, path[i + 2])
                    : _to;
            //调用(输入地址,输出地址)的pair合约地址的交换方法(输出数额0,输出数额1,to地址,0x00)
            ISwapPair(SwapLibrary.pairFor(factory, input, output)).swap(
                amount0Out,
                amount1Out,
                to,
                new bytes(0)
            );
        }
    }

    /**
     * @dev 根据精确的token交换尽量多的token*
     * @param amountIn 精确输入数额
     * @param amountOutMin 最小输出数额
     * @param path 路径数组
     * @param to to地址
     * @param deadline 最后期限
     * @return amounts  数额数组
     */
    function swapExactTokensForTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        //数额数组 ≈ 遍历路径数组(
        //      (输入数额 * (1000-fee) * 储备量Out) /
        //      (储备量In * 1000 + 输入数额 * (1000-fee)))
        amounts = SwapLibrary.getAmountsOut(factory, amountIn, path);
        //确认数额数组最后一个元素>=最小输出数额
        require(
            amounts[amounts.length - 1] >= amountOutMin,
            "SwapRouter: INSUFFICIENT_OUTPUT_AMOUNT"
        );
        //将数量为数额数组[0]的路径[0]的token从调用者账户发送到路径0,1的pair合约
        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            SwapLibrary.pairFor(factory, path[0], path[1]),
```

```
            amounts[0]
        );
        //私有交换(数额数组,路径数组,to地址)
        _swap(amounts, path, to);
    }

    /**
     * @dev 使用尽量少的token交换精确的token*
     * @param amountOut 精确输出数额
     * @param amountInMax 最大输入数额
     * @param path 路径数组
     * @param to to地址
     * @param deadline 最后期限
     * @return amounts  数额数组
     */
    function swapTokensForExactTokens(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        //数额数组 ≈ 遍历路径数组(
        //      (储备量In * 储备量Out * 1000) /
        //      (储备量Out - 输出数额 * (1000-fee)) + 1)
        amounts = SwapLibrary.getAmountsIn(factory, amountOut, path);
        //确认数额数组第一个元素<=最大输入数额
        require(
            amounts[0] <= amountInMax,
            "SwapRouter: EXCESSIVE_INPUT_AMOUNT"
        );
        //将数量为数额数组[0]的路径[0]的token从调用者账户发送到路径0,1的pair合约
        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            SwapLibrary.pairFor(factory, path[0], path[1]),
            amounts[0]
        );
        //私有交换(数额数组,路径数组,to地址)
        _swap(amounts, path, to);
    }

    /**
     * @dev 根据精确的ETH交换尽量多的token*
     * @param amountOutMin 最小输出数额
     * @param path 路径数组
     * @param to to地址
     * @param deadline 最后期限
     * @return amounts  数额数组
     */
    function swapExactETHForTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        payable
        virtual
        override
```

```
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        //确认路径第一个地址为WETH
        require(path[0] == WETH, "SwapRouter: INVALID_PATH");
        //数额数组 ≈ 遍历路径数组(
        //      (msg.value * (1000-fee) * 储备量Out) /
        //      (储备量In * 1000 + msg.value * (1000-fee)))
        amounts = SwapLibrary.getAmountsOut(factory, msg.value, path);
        //确认数额数组最后一个元素>=最小输出数额
        require(
            amounts[amounts.length - 1] >= amountOutMin,
            "SwapRouter: INSUFFICIENT_OUTPUT_AMOUNT"
        );
        //将数额数组[0]的数额存款ETH到ETH合约
        IWETH(WETH).deposit{value: amounts[0]}();
        //断言将数额数组[0]的数额的ETH发送到路径(0,1)的pair合约地址
        assert(
            IWETH(WETH).transfer(
                SwapLibrary.pairFor(factory, path[0], path[1]),
                amounts[0]
            )
        );
        //私有交换(数额数组,路径数组,to地址)
        _swap(amounts, path, to);
    }

    /**
     * @dev 使用尽量少的token交换精确的ETH*
     * @param amountOut 精确输出数额
     * @param amountInMax 最大输入数额
     * @param path 路径数组
     * @param to to地址
     * @param deadline 最后期限
     * @return amounts  数额数组
     */
    function swapTokensForExactETH(
        uint256 amountOut,
        uint256 amountInMax,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        //确认路径最后一个地址为WETH
        require(path[path.length - 1] == WETH, "SwapRouter: INVALID_PATH");
        //数额数组 ≈ 遍历路径数组(
        //      (储备量In * 储备量Out * 1000) /
        //      (储备量Out - 输出数额 * (1000-fee)) + 1)
        amounts = SwapLibrary.getAmountsIn(factory, amountOut, path);
        //确认数额数组第一个元素<=最大输入数额
        require(
            amounts[0] <= amountInMax,
            "SwapRouter: EXCESSIVE_INPUT_AMOUNT"
        );
        //将数量为数额数组[0]的路径[0]的token从调用者账户发送到路径0,1的pair合约
        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            SwapLibrary.pairFor(factory, path[0], path[1]),
            amounts[0]
```

```
        );
        //私有交换(数额数组,路径数组,当前合约地址)
        _swap(amounts, path, address(this));
        //从ETH合约提款数额数组最后一个数值的ETH
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        //将数额数组最后一个数值的ETH发送到to地址
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }

    /**
     * @dev 根据精确的token交换尽量多的ETH*
     * @param amountIn 精确输入数额
     * @param amountOutMin 最小输出数额
     * @param path 路径数组
     * @param to to地址
     * @param deadline 最后期限
     * @return amounts  数额数组
     */
    function swapExactTokensForETH(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        virtual
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        //确认路径最后一个地址为WETH
        require(path[path.length - 1] == WETH, "SwapRouter: INVALID_PATH");
        //数额数组 ≈ 遍历路径数组(
        //      (输入数额 * (1000-fee) * 储备量Out) /
        //      (储备量In * 1000 + 输入数额 * (1000-fee)))
        amounts = SwapLibrary.getAmountsOut(factory, amountIn, path);
        //确认数额数组最后一个元素>=最小输出数额
        require(
            amounts[amounts.length - 1] >= amountOutMin,
            "SwapRouter: INSUFFICIENT_OUTPUT_AMOUNT"
        );
        //将数量为数额数组[0]的路径[0]的token从调用者账户发送到路径0,1的pair合约
        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            SwapLibrary.pairFor(factory, path[0], path[1]),
            amounts[0]
        );
        //私有交换(数额数组,路径数组,当前合约地址)
        _swap(amounts, path, address(this));
        //从WETH合约提款数额数组最后一个数值的ETH
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        //将数额数组最后一个数值的ETH发送到to地址
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }

    /**
     * @dev 使用尽量少的ETH交换精确的token*
     * @param amountOut 精确输出数额
     * @param path 路径数组
     * @param to to地址
     * @param deadline 最后期限
     * @return amounts  数额数组
     */
    function swapETHForExactTokens(
```

```
        uint256 amountOut,
        address[] calldata path,
        address to,
        uint256 deadline
    )
        external
        payable
        virtual
        override
        ensure(deadline)
        returns (uint256[] memory amounts)
    {
        //确认路径第一个地址为WETH
        require(path[0] == WETH, "SwapRouter: INVALID_PATH");
        //数额数组 = 遍历路径数组(
        //      (储备量In * 储备量Out * 1000) /
        //      (储备量Out - 输出数额 * (1000-fee)) + 1)
        amounts = SwapLibrary.getAmountsIn(factory, amountOut, path);
        //确认数额数组第一个元素<=msg.value
        require(amounts[0] <= msg.value, "SwapRouter: EXCESSIVE_INPUT_AMOUNT");
        //将数额数组[0]的数额存款ETH到WETH合约
        IWETH(WETH).deposit{value: amounts[0]}();
        //断言将数额数组[0]的数额的WETH发送到路径(0,1)的pair合约地址
        assert(
            IWETH(WETH).transfer(
                SwapLibrary.pairFor(factory, path[0], path[1]),
                amounts[0]
            )
        );
        //私有交换(数额数组,路径数组,to地址)
        _swap(amounts, path, to);
        //如果`收到的主币数量`>`数额数组[0]` 则返还`收到的主币数量`-`数额数组[0]`
        if (msg.value > amounts[0])
            TransferHelper.safeTransferETH(msg.sender, msg.value - amounts[0]);
    }

    // **** 交换 (支持收取转帐税的Token) ****
    // requires the initial amount to have already been sent to the first pair
    /**
     * @dev 私有交换支持Token收转帐税*
     * @param path 路径数组
     * @param _to to地址
     */
    function _swapSupportingFeeOnTransferTokens(
        address[] memory path,
        address _to
    ) internal virtual {
        //遍历路径数组
        for (uint256 i; i < path.length - 1; i++) {
            //(输入地址,输出地址) = (当前地址,下一个地址)
            (address input, address output) = (path[i], path[i + 1]);
            // 根据输入地址,输出地址找到配对合约
            ISwapPair pair =
                ISwapPair(SwapLibrary.pairFor(factory, input, output));
            //token0 = 排序(输入地址,输出地址)
            (address token0, ) = SwapLibrary.sortTokens(input, output);
            // 定义一些数额变量
            uint256 amountInput;
            uint256 amountOutput;
            {
                //避免堆栈太深的错误
                //获取配对的交易手续费
                uint8 fee = pair.fee();
                //获取配对合约的储备量0,储备量1
                (uint256 reserve0, uint256 reserve1, ) = pair.getReserves();
                // 排序输入储备量和输出储备量
```

```
                (uint256 reserveInput, uint256 reserveOutput) =
                    input == token0
                        ? (reserve0, reserve1)
                        : (reserve1, reserve0);
                // 储备量0,1,配对合约中的余额-储备量
                amountInput = IERC20(input).balanceOf(address(pair)).sub(
                    reserveInput
                );
                //根据输入数额,输入储备量,输出储备量,交易手续费计算输出数额
                amountOutput = SwapLibrary.getAmountOut(
                    amountInput,
                    reserveInput,
                    reserveOutput,
                    fee
                );
            }
            // // 排序输出数额0,输出数额1
            (uint256 amount0Out, uint256 amount1Out) =
                input == token0
                    ? (uint256(0), amountOutput)
                    : (amountOutput, uint256(0));
            //to地址 = i<路径长度-2 ? (输出地址,路径下下个地址)的pair合约地址 : to地址
            address to =
                i < path.length - 2
                    ? SwapLibrary.pairFor(factory, output, path[i + 2])
                    : _to;
            //调用pair合约的交换方法(输出数额0,输出数额1,to地址,0x00)
            pair.swap(amount0Out, amount1Out, to, new bytes(0));
        }
    }

    /**
     * @dev 根据精确的token交换尽量多的token,支持Token收转帐税*
     * @param amountIn 精确输入数额
     * @param amountOutMin 最小输出数额
     * @param path 路径数组
     * @param to to地址
     * @param deadline 最后期限
     */
    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external virtual override ensure(deadline) {
        //将数量为数额数组[0]的路径[0]的token从调用者账户发送到路径0,1的pair合约
        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            SwapLibrary.pairFor(factory, path[0], path[1]),
            amountIn
        );
        // 记录to地址在地址路径最后一个token中的余额
        uint256 balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        // 调用私有交换支持Token收转帐税方法
        _swapSupportingFeeOnTransferTokens(path, to);
        // 确认to地址收到的地址路径中最后一个token数量大于最小输出数量
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >=
                amountOutMin,
            "SwapRouter: INSUFFICIENT_OUTPUT_AMOUNT"
        );
    }

    /**
```

```
     * @dev 根据精确的ETH交换尽量多的token,支持Token收转帐税*
     * @param amountOutMin 最小输出数额
     * @param path 路径数组
     * @param to to地址
     * @param deadline 最后期限
     */
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external payable virtual override ensure(deadline) {
        //确认路径第一个地址为WETH
        require(path[0] == WETH, "SwapRouter: INVALID_PATH");
        //输入数量=合约收到的主币数量
        uint256 amountIn = msg.value;
        //向WETH合约存款ETH
        IWETH(WETH).deposit{value: amountIn}();
        //断言将WETH发送到了地址路径0,1组成的配对合约中
        assert(
            IWETH(WETH).transfer(
                SwapLibrary.pairFor(factory, path[0], path[1]),
                amountIn
            )
        );
        // 记录to地址在地址路径最后一个token中的余额
        uint256 balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        // 调用私有交换支持Token收转帐税方法
        _swapSupportingFeeOnTransferTokens(path, to);
        // 确认to地址收到的地址路径中最后一个token数量大于最小输出数量
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >=
                amountOutMin,
            "SwapRouter: INSUFFICIENT_OUTPUT_AMOUNT"
        );
    }

    /**
     * @dev 根据精确的token交换尽量多的ETH,支持Token收转帐税*
     * @param amountIn 精确输入数额
     * @param amountOutMin 最小输出数额
     * @param path 路径数组
     * @param to to地址
     * @param deadline 最后期限
     */
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint256 amountIn,
        uint256 amountOutMin,
        address[] calldata path,
        address to,
        uint256 deadline
    ) external virtual override ensure(deadline) {
        //确认路径最后一个地址为WETH
        require(path[path.length - 1] == WETH, "SwapRouter: INVALID_PATH");
        //将地址路径0的Token发送到地址路径0,1组成的配对合约
        TransferHelper.safeTransferFrom(
            path[0],
            msg.sender,
            SwapLibrary.pairFor(factory, path[0], path[1]),
            amountIn
        );
        //调用私有交换支持Token收转帐税方法
        _swapSupportingFeeOnTransferTokens(path, address(this));
        //输出金额=当前合约收到的WETH数量
        uint256 amountOut = IERC20(WETH).balanceOf(address(this));
        //确认输出金额大于最小输出数额
```

```
        require(
            amountOut >= amountOutMin,
            "SwapRouter: INSUFFICIENT_OUTPUT_AMOUNT"
        );
        //向WETH合约取款
        IWETH(WETH).withdraw(amountOut);
        //将ETH发送到to地址
        TransferHelper.safeTransferETH(to, amountOut);
    }

    // **** LIBRARY FUNCTIONS ****
    function quote(
        uint256 amountA,
        uint256 reserveA,
        uint256 reserveB
    ) public pure virtual override returns (uint256 amountB) {
        return SwapLibrary.quote(amountA, reserveA, reserveB);
    }

    function getAmountOut(
        uint256 amountIn,
        uint256 reserveIn,
        uint256 reserveOut,
        uint8 fee
    ) public pure virtual override returns (uint256 amountOut) {
        return SwapLibrary.getAmountOut(amountIn, reserveIn, reserveOut, fee);
    }

    function getAmountIn(
        uint256 amountOut,
        uint256 reserveIn,
        uint256 reserveOut,
        uint8 fee
    ) public pure virtual override returns (uint256 amountIn) {
        return SwapLibrary.getAmountIn(amountOut, reserveIn, reserveOut, fee);
    }

    function getAmountsOut(uint256 amountIn, address[] memory path)
        public
        view
        virtual
        override
        returns (uint256[] memory amounts)
    {
        return SwapLibrary.getAmountsOut(factory, amountIn, path);
    }

    function getAmountsIn(uint256 amountOut, address[] memory path)
        public
        view
        virtual
        override
        returns (uint256[] memory amounts)
    {
        return SwapLibrary.getAmountsIn(factory, amountOut, path);
    }
}
```


Analysis of audit results

Re-Entrancy

- **Description:**
  One of the features of smart contracts is the ability to call and utilise code of other external contracts. Contracts also typically handle Blockchain Currency, and as such often send Blockchain Currency to various external user addresses. The operation of calling external contracts, or sending Blockchain Currency to an address, requires the contract to submit an external call. These external calls can be hijacked by attackers whereby they force the contract to execute further code (i.e. through a fallback function) , including calls back into itself. Thus the code execution "re-enters" the contract. Attacks of this kind were used in the infamous DAO hack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Arithmetic Over/Under Flows

- **Description:**
  The Virtual Machine (EVM) specifies fixed-size data types for integers. This means that an integer variable, only has a certain range of numbers it can represent. A uint8 for example, can only store numbers in the range [0,255]. Trying to store 256 into a uint8 will result in 0. If care is not taken, variables in Solidity can be exploited if user input is unchecked and calculations are performed which result in numbers that lie outside the range of the data type that stores them.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unexpected Blockchain Currency

- **Description:**
  Typically when Blockchain Currency is sent to a contract, it must execute either the fallback function, or another function described in the contract. There are two exceptions to this, where Blockchain Currency can exist in a contract without having executed any code. Contracts which rely on code execution for every Blockchain Currency sent to the contract can be vulnerable to attacks where Blockchain Currency is forcibly sent to a contract.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Delegatecall

- **Description:**
  The CALL and DELEGATECALL opcodes are useful in allowing developers to modularise their code. Standard external message calls to contracts are handled by the CALL opcode whereby code is run in the context of the external contract/function. The DELEGATECALL opcode is identical to the standard message call, except that the code executed at the targeted address is run in the context of the calling contract along with the fact that

msg.sender and msg.value remain unchanged. This feature enables the implementation of libraries whereby developers can create reusable code for future contracts.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:** no.

Default Visibilities

- **Description:**
  Functions in Solidity have visibility specifiers which dictate how functions are allowed to be called. The visibility determines whBlockchain Currency a function can be called externally by users, by other derived contracts, only internally or only externally. There are four visibility specifiers, which are described in detail in the Solidity Docs. Functions default to public allowing users to call them externally. Incorrect use of visibility specifiers can lead to some devestating vulernabilities in smart contracts as will be discussed in this section.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Entropy Illusion

- **Description:**
  All transactions on the blockchain are deterministic state transition operations. Meaning that every transaction modifies the global state of the ecosystem and it does so in a calculable way with no uncertainty. This ultimately means that inside the blockchain ecosystem there is no source of entropy or randomness. There is no rand() function in Solidity. Achieving decentralised entropy (randomness) is a well established problem and many ideas have been proposed to address this (see for example, RandDAO or using a chain of Hashes as described by Vitalik in this post).
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

External Contract Referencing

- **Description:**
  One of the benefits of the global computer is the ability to re-use code and interact with contracts already deployed on the network. As a result, a large number of contracts reference external contracts and in general operation use external message calls to interact with these contracts. These external message calls can mask malicious actors intentions in some non-obvious ways, which we will discuss.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unsolved TODO comments

- **Description:**
  Check for Unsolved TODO comments
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Short Address/Parameter Attack

- **Description:**
  This attack is not specifically performed on Solidity contracts themselves but on third party applications that may interact with them. I add this attack for completeness and to be aware of how parameters can be manipulated in contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unchecked CALL Return Values

- **Description:**
  There a number of ways of performing external calls in solidity. Sending Blockchain Currency to external accounts is commonly performed via the transfer() method. However, the send() function can also be used and, for more versatile external calls, the CALL opcode can be directly employed in solidity. The call() and send() functions return a boolean indicating if the call succeeded or failed. Thus these functions have a simple caveat, in that the transaction that executes these functions will not revert if the external call (intialised by call() or send()) fails, rather the call() or send() will simply return false. A common pitfall arises when the return value is not checked, rather the developer expects a revert to occur.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Race Conditions / Front Running

- **Description:**
  The combination of external calls to other contracts and the multi-user nature of the underlying blockchain gives rise to a variety of potential Solidity pitfalls whereby users race code execution to obtain unexpected states. Re-Entrancy is one example of such a race condition. In this section we will talk more generally about different kinds

of race conditions that can occur on the blockchain. There is a variety of good posts on this subject, a few are: Wiki - Safety, DASP - Front-Running and the Consensus - Smart Contract Best Practices.

- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Denial Of Service (DOS)

- **Description:**
  This category is very broad, but fundamentally consists of attacks where users can leave the contract inoperable for a small period of time, or in some cases, permanently. This can trap Blockchain Currency in these contracts forever, as was the case with the Second Parity MultiSig hack
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Block Timestamp Manipulation

- **Description:**
  Block timestamps have historically been used for a variety of applications, such as entropy for random numbers (see the Entropy Illusion section for further details), locking funds for periods of time and various state-changing conditional statements that are time-dependent. Miner's have the ability to adjust timestamps slightly which can prove to be quite dangerous if block timestamps are used incorrectly in smart contracts.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Constructors with Care

- **Description:**
  Constructors are special functions which often perform critical, privileged tasks when initialising contracts. Before solidity v0.4.22 constructors were defined as functions that had the same name as the contract that contained them. Thus, when a contract name gets changed in development, if the constructor name isn't changed, it becomes a normal, callable function. As you can imagine, this can (and has) lead to some interesting contract hacks.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Unintialised Storage Pointers

- **Description:**
  The EVM stores data either as storage or as memory. Understanding exactly how this is done and the default types for local variables of functions is highly recommended when developing contracts. This is because it is possible to produce vulnerable contracts by inappropriately intialising variables.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Floating Points and Numerical Precision

- **Description:**
  As of this writing (Solidity v0.4.24), fixed point or floating point numbers are not supported. This means that floating point representations must be made with the integer types in Solidity. This can lead to errors/vulnerabilities if not implemented correctly.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

tx.origin Authentication

- **Description:**
  Solidity has a global variable, tx.origin which 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 smart contracts leaves the contract vulnerable to a phishing-like attack.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

Permission restrictions

- **Description:**
  Contract managers who can control liquidity or pledge pools, etc., or impose unreasonable restrictions on other users.
- **Detection results:**

```
PASSED!
```

- **Security suggestion:**
  no.

armors.io

contact@armors.io