知道创宇

密级 ★ | 项目组公开

智能合约审计报告

安全状态

安全

★ ★ ★ ★ ★

主测人： 知道创宇区块链安全研究团队

版本说明

修订内容	时间	修订者	版本号
编写文档	20210402	知道创宇区块链安全研究团队	V1.0

文档信息

文档名称	文档版本	报告编号	保密级别
LongBit 智能合约审计报告	V1.0		项目组公开

声明

创宇仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，创宇无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向创宇提供的文件和资料。创宇假设:已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，创宇对由此而导致的损失和不利影响不承担任何责任。

目录

1. 综述

本次报告有效测试时间是从 2021 年 4 月 1 日开始到 2021 年 4 月 2 日结束，在此期间针对 **LongBit 智能合约的交易所代码**的安全性和规范性进行审计并以此作为报告统计依据。

本次智能合约安全审计的范围，不包括外部合约调用，不包含未来可能出现的新型攻击方式，不包含合约升级或篡改后的代码（随着项目方的发展，智能合约可能会增加新的 pool、新的功能模块，新的外部合约调用等），不包含前端安全与服务器安全。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三章节）进行了全面的分析，综合评定为**通过**。

本次智能合约安全审计结果： 通过

由于本次测试过程在非生产环境下进行，所有代码均为最新备份，测试过程均与相关接口人进行沟通，并在操作风险可控的情况下进行相关测试操作，以规避测试过程中的生产运营风险、代码安全风险。

本次审计的报告信息：

报告编号：

报告查询地址链接:

本次审计的目标信息：

条目	描述
代码类型	DeFi 协议代码、BSC 智能合约代码
代码语言	Solidity

合约文件及哈希：

合约文件	MD5
LongBitAudit.sol	18d88a4f6a243e1824139c7209e88d1a

LongBitController.sol	3af6d72dc91fbeb03bb3fa68c60f6375
Migrations.sol	ca8d6ca8a6edf34f149a5095a8b074c9
Bakery.sol	45b6e09e4b39d04acbb8a3174d1851d5
BakeryBase.sol	567fea6ae3e86c4d36fa0c2eeac80d39
BakeryBNB.sol	01c052f5435a0d7a9b13dc0f4857a889
BakeryLibrary.sol	7e8a9777a6f27510e0aa8cb19671dbc4
IBakeryFactory.sol	28445778782beb783e2f091de8bceebe
IBakeryPair.sol	b9a444dd43fb232d81a006acd86af701
Jul.sol	c87f9fff45eb0aa2c73ea3d308e8617f
JulBase.sol	ecc753b1f26018657cee5a542cd93972
JulBNB.sol	a61f3d6d238e224967843335fdeca699
JulLibrary.sol	520b569e75eb3e1e125b617ad9f7c221
IJulFactory.sol	0c996f820d4d323118abcad41c7793dd
IJulPair.sol	53043cd2992caf20c50271102be31957
Long.sol	267ff9b1a1495b19820d0f3a891b2070
LongBase.sol	82ee620b2609b3ea97ab886485aa4b5a
LongBNB.sol	dd2e40f04b6412ef5f91acc11d5b7882
LongLibrary.sol	6db4c01b0d09fcda9a0f109bb4f355b3
ILongFactory.sol	6686d26514d23f91408984cd8311e720
ILongPair.sol	e3134c2a7a7c9617d2d863b35f9f5863

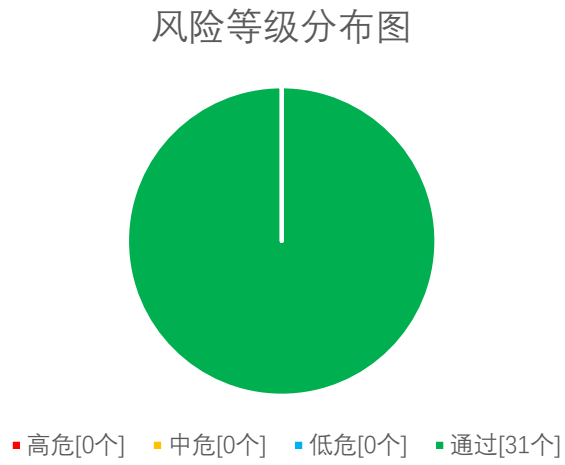
Pancake.sol	ecc1c7a3527fb28b3956430475b1cbfe
PancakeBase.sol	7a1e0ba60d617bc4ab2b12ee2436811d
PancakeBNB.sol	e2f89a6c39d3d0168ead62dcbd9c02f2
PancakeLibrary.sol	f7fb155f57c0316caaee0e43a922e99d
IPancakeFactory.sol	d53bb6e5b33ff9f6d563f5101db62344
IPancakePair.sol	e18bd2e7f0462cfb58f071b90fc7e470
ExchangeBase.sol	82a54aa4a02a9736d39f13fceba70b05
IExchange.sol	1fba8718169604507cd5827b8e0cd18a
ILongBitController.sol	b68028b89d101501641c382245fe2092
IWBNB.sol	df3d70ed8c1b1ea4c90fc789d2b52f27
ArrayLib.sol	71303f01539d7edb3124b31dd7a200dc
ERC20Lib.sol	44f5d78a11e52858ca7e854a15a31207
FlagLib.sol	b105b86b0cd01b846570088ec161074c

2. 代码漏洞分析

2.1 漏洞等级分布

本次漏洞风险按等级统计：

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	0	31



2.2 审计结果汇总说明

审计结果			
审计项目	审计内容	状态	描述
业务安全性检测	预估兑换量	通过	经检测，不存在安全问题。
	代币兑换	通过	经检测，不存在安全问题。
	增添与设置交易所	通过	经检测，不存在安全问题。
	交易兑换框架	通过	经检测，不存在安全问题。
代码基本漏洞检测	编译器版本安全	通过	经检测，不存在该安全问题。
	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。
	require/assert 的合理使用	通过	经检测，不存在该安全问题。
	fallback 函数安全	通过	经检测，不存在该安全问题。
	tx.orgin 身份验证	通过	经检测，不存在该安全问题。
	owner 权限控制	通过	经检测，不存在该安全问题。
	gas 消耗检测	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	增发代币漏洞	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。
	错误使用随机数检测	通过	经检测，不存在该安全问题。
	不安全的接口使用	通过	经检测，不存在该安全问题。
	变量覆盖	通过	经检测，不存在该安全问题。

	未初始化的存储指针	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	交易顺序依赖检测	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	通过	经检测，不存在该安全问题。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	假充值漏洞检测	通过	经检测，不存在该安全问题。
	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	重排攻击检测	通过	经检测，不存在该安全问题。

3. 业务安全性检测

3.1. 预估兑换量【通过】

审计分析：预估兑换量功能由 LongBitAudit.sol 和 LongBitController.sol 合约文件中的 getExpectedReturn、getExpectedReturnWithGas、getExpectedReturnWithGasMulti 等函数共同实现，用于用户在兑换前查询预估兑换量。LongBitAudit.sol 合约中的查询最终都会通过调用控制器合约的 getExpectedReturnWithGasa 函数获取数据。

```
// File: LongBitAudit.sol
/*
 * 获取估算兑换量
 */
function getExpectedReturn(
    IERC20 fromToken,
    IERC20 destToken,
    uint256 amount,
    uint256 parts,
    uint256 flags
)
    public
    view
    returns(
        uint256 returnAmount,
        uint256[] memory distribution
    )
{
    (returnAmount, , distribution) = getExpectedReturnWithGas(
        fromToken,
        destToken,
```

```
            amount,
            parts,
            flags,
            0
        );
}

/*
 * 获取估算兑换量 + 估算 gas 使用量
 */
function getExpectedReturnWithGas(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags,
        uint256 destTokenEthPriceTimesGasPrice
)
        public
        view
        returns(
            uint256 returnAmount,
            uint256 estimateGasAmount,
            uint256[] memory distribution
        )
{
        return controller.getExpectedReturnWithGas(
            fromToken,
            destToken,
            amount,
            parts,
            flags,
            destTokenEthPriceTimesGasPrice
```

```
    );
}

// File: LongBitController.sol
/*
 * 获取可兑换量 + 预估 gas + 交易所权重
 */
function getExpectedReturnWithGas(//knownsec// 查询预估可获取量
    IERC20 fromToken,
    IERC20 destToken,
    uint256 amount,
    uint256 parts,
    uint256 flags,
    uint256 destTokenEthPriceTimesGasPrice
)
    public
    view
    override
    returns(
        uint256 returnAmount,
        uint256 estimateGasAmount,
        uint256[] memory distribution
    )
{
    if (fromToken == destToken) {
        return (amount, 0, new uint256[](DEXES_COUNT));
    }

    bool invert = flags.check(FLAG_DISABLE_ALL_SPLIT_SOURCES);
    int256[][] memory matrix = new int256[][](DEXES_COUNT);
    uint256[] memory gases = new uint256[](DEXES_COUNT);

    (returnAmount, estimateGasAmount, distribution) = _getAllReserves(
```

```
            Args({
                fromToken: fromToken,
                destToken: destToken,
                amount: amount,
                parts: parts,
                flags: flags,
                destTokenEthPriceTimesGasPrice: destTokenEthPriceTimesGasPrice,
                matrix: matrix,
                gases: gases,
                invert: invert
            })
        );
    }

    function _getAllReserves(Args memory args)//knownsec// 查询储量
        internal
        view
        returns(
            uint256 returnAmount,
            uint256 estimateGasAmount,
            uint256[] memory distribution
        )
    {

        bool atLeastOnePositive = false;
        for (uint i = 0; i < DEXES_COUNT; i++) {
            uint256[] memory rets;
            IExchange exchange = IExchange(exchanges[i]);
            (rets, args.gases[i]) = (args.invert != exchange.checkFlag(args.flags))?_calculateNoReturn(args.parts):exchange.calculate(args.fromToken, args.destToken, args.amount, args.parts, args.flags);

            // Prepend zero and sub gas
```

```
            int256 gas = int256(args.gases[i].mul(args.destTokenEthPriceTimesGasPrice).div(1e18));
            args.matrix[i] = new int256[](args.parts + 1);
            for (uint j = 0; j < rets.length; j++) {
                args.matrix[i][j + 1] = int256(rets[j]) - gas;
                atLeastOnePositive = atLeastOnePositive || (args.matrix[i][j + 1] > 0);
            }
        }

        if (!atLeastOnePositive) {
            for (uint i = 0; i < DEXES_COUNT; i++) {
                for (uint j = 1; j < args.parts + 1; j++) {
                    if (args.matrix[i][j] == 0) {
                        args.matrix[i][j] = VERY_NEGATIVE_VALUE;
                    }
                }
            }
        }

        (returnAmount, estimateGasAmount, distribution) = _getReturnAndGasByDistribution(args);
    }
```

安全建议：无。

3.2. 代币兑换【通过】

审计分析：代币兑换功能主要由 LongBitAudit.sol 和 LongBitController.sol 合约文件中的 swap 类函数，如 swap、_swap、swapWithReferral、swapMulti、swapWithReferralMulti 等函数联合实现，用于用户通过多种方式兑换指定代币。其中 LongBitAudit.sol 合约中提供的兑换功能最终会调用控制器合约的 swapMulti 函数实现兑换。

```
// File:LongBitAudit.sol
/*
 * 多 token 兑换 + 推荐人功能
 */
function swapWithReferralMulti(
    IERC20[] memory tokens,
    uint256 amount,
    uint256 minReturn,
    uint256[] memory distribution,
    uint256[] memory flags,
    address referral,
    uint256 feePercent
) public payable returns(uint256 returnAmount) {
    require(tokens.length >= 2 && amount > 0, "LongBit: swap makes no sense");
    require(flags.length == tokens.length - 1, "LongBit: flags array length is invalid");
    require((msg.value != 0) == tokens.first().isBNB(), "LongBit: msg.value should be used only for BNB swap");
    require(feePercent <= 0.03e18, "LongBit: feePercent out of range");
    require(msg.sender != referral, "LongBit:   sender and referral cannot be the same");

    Balances memory beforeBalances = _getFirstAndLastBalances(tokens, true);

    // Transfer From
    if (amount == uint256(-1)) {
        amount = Math.min(
            tokens.first().balanceOf(msg.sender),
            tokens.first().allowance(msg.sender, address(this))
        );
    }
    tokens.first().universalTransferFromSenderToThis(amount);//knownsec// 转入 token0
    uint256 confirmed = tokens.first().universalBalanceOf(address(this)).sub(beforeBalances.ofFromToken);//knownsec// 实际额
```

```
        // Swap
        tokens.first().universalApprove(address(controller), confirmed);
        controller.swapMulti{value:tokens.first().isBNB() ? confirmed : 0}(
            tokens,
            confirmed,
            minReturn,
            distribution,
            flags
        );

        Balances memory afterBalances = _getFirstAndLastBalances(tokens, false);

        // Return
        returnAmount = afterBalances.ofDestToken.sub(beforeBalances.ofDestToken);//knownsec//兑换实际额
        require(returnAmount >= minReturn, "LongBit: actual return amount is less than minReturn");
        //手续费
        returnAmount = _mintFee(tokens, returnAmount);
        //转回
        tokens.last().universalTransfer(msg.sender, returnAmount);

        emit Swapped(
            msg.sender,
            tokens.first(),
            tokens.last(),
            amount,
            returnAmount,
            minReturn,
            distribution,
            flags,
            referral,
```

```
            feePercent
        );

        // Return remainder
        if (afterBalances.ofFromToken > beforeBalances.ofFromToken) {//knownsec// 转回剩余fromToken
            tokens.first().universalTransfer(msg.sender, afterBalances.ofFromToken.sub(beforeBalances.ofFromToken));
        }
    }

    // File:LongBitController.sol
    /*
     * 根据权重兑换
     */
    function swapMulti(
        IERC20[] memory tokens,
        uint256 amount,
        uint256 minReturn,
        uint256[] memory distribution,
        uint256[] memory flags
    ) public payable override returns(uint256 returnAmount) {
        tokens[0].universalTransferFrom(msg.sender, address(this), amount);//knownsec// 转入

        returnAmount = tokens[0].universalBalanceOf(address(this));
        for (uint i = 1; i < tokens.length; i++) {
            if (tokens[i - 1] == tokens[i]) {
                continue;
            }

            uint256[] memory dist = new uint256[](distribution.length);
            for (uint j = 0; j < distribution.length; j++) {
                dist[j] = (distribution[j] >> (8 * (i - 1))) & 0xFF;
```

```
            }

            _swap(
                tokens[i - 1],
                tokens[i],
                returnAmount,
                dist,
                flags[i - 1]
            );
            returnAmount = tokens[i].universalBalanceOf(address(this));
        }
        require(returnAmount >= minReturn, "Gswap: actual return amount is less than minReturn");
        tokens[tokens.length - 1].universalTransfer(msg.sender, returnAmount);//knownsec// 转出
}


function _swap(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256[] memory distribution,
        uint256 flags
) internal returns(uint256 returnAmount) {
        require(distribution.length <= exchanges.length, "Gswap: Distribution array should not exceed reserves array size");

        uint256 parts = 0;
        uint256 lastNonZeroIndex = 0;
        for (uint i = 0; i < distribution.length; i++) {
            if (distribution[i] > 0) {
                parts = parts.add(distribution[i]);
                lastNonZeroIndex = i;
            }
```

```
        }

        if (parts == 0) {
            if (fromToken.isBNB()) {
                msg.sender.transfer(msg.value);
                return msg.value;
            }
            return amount;
        }

        for (uint i = 0; i < distribution.length; i++) {
            if (distribution[i] == 0) {
                continue;
            }

            uint256 swapAmount = amount.mul(distribution[i]).div(parts);//knownsec// 按占比取amount
            if (i == lastNonZeroIndex) {
                swapAmount = amount;
            }
            amount -= swapAmount;//knownsec// 无溢出
            IExchange exchange = IExchange(exchanges[i]);
            fromToken.universalApprove(address(exchange), swapAmount);
            exchange.swap{value:fromToken.isBNB() ? swapAmount : 0}(fromToken, destToken, swapAmount, flags);
        }
    }
```

安全建议：无。

3.3. 增添与设置交易所【通过】

审计分析：交易所的增添与设置功能由 LongBitController.sol 合约文件中的

addExchangeh 函数和 setExchange 函数实现，用于控制器合约 owner 增添或设置交易所。

```
function addExchange(address exchange) public onlyOwner {//knownsec//  添加交易所,仅 owner 调用
    exchanges.push(exchange);
    DEXES_COUNT = exchanges.length;
}

function setExchange(uint256 index, address exchange) public onlyOwner {
    exchanges[index] = exchange;
}
```

安全建议：无。

3.4. 交易兑换框架【通过】

审计分析：LongBit 项目实现了相同框架的兑换交易合约，包含 bakery、jul、long、以及 pancake 总计四个模块，该部分代码实现框架一致，仅在命名和对应地址初始化存在不一致。代码主要实现了对应模块内兑换数量的计算以及执行兑换逻辑。以下代码内容以 bakery 模块为主。

```
//knownsec// bakery
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "../../interface/IExchange.sol";
import "./BakeryBase.sol";

contract Bakery is IExchange, BakeryBase{

    uint256 internal constant FLAG_DISABLE_BAKERY = 0x2000000000000;
```

```
    //knownsec// 检查 flag 函数，用于检查传入 flag 与指定标识一致性
    function checkFlag(uint256 flags) external view override returns (bool){
        return flags.check(FLAG_DISABLE_BAKERY_ALL | FLAG_DISABLE_BAKERY);
    }

    /*
     * 计算兑换量
     */
    function calculate(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) external view override returns(uint256[] memory rets, uint256 gas) {
        return _calculateUniswapV2(
            fromToken,
            destToken,
            _linearInterpolation(amount, parts), //knownsec// 获取对应的线性插值
            flags
        ); //knownsec// 调用 BakeryBase 合约中的_calculateUniswapV2 函数查询兑换数量相关参数
    }

    /*
     * 执行兑换
     */
    function swap(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 flags
    )external payable override {
```

```
            _getSwapAmount(fromToken, amount); //knownsec// 兑换代币转账
            _swapOnUniswapV2Internal(
                fromToken,
                destToken,
                amount,
                flags
            ); //knownsec// 进行代币兑换
            _swapReturnAmount(destToken);
        }
    }

    //knwonsec// BakeryBase
    // SPDX-License-Identifier: MIT
    pragma solidity ^0.6.12;

    import '../ExchangeBase.sol';
    import './IBakeryFactory.sol';
    import './IBakeryPair.sol';
    import './BakeryLibrary.sol';

    contract BakeryBase is ExchangeBase{

        using BakeryLibrary for IBakeryPair;

        uint256 constant FLAG_DISABLE_BAKERY_ALL = 0x1000000000000;

        //bakery
        IBakeryFactory constant internal bakery =
    IBakeryFactory(0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f);



        function _calculateUniswapV2OverMidToken(
            IERC20 fromToken,
```

```
        IERC20 midToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) internal view returns(uint256[] memory rets, uint256 gas) {
        rets = _linearInterpolation(amount, parts);

        uint256 gas1;
        uint256 gas2;
        (rets, gas1) = _calculateUniswapV2(fromToken, midToken, rets, flags);
        (rets, gas2) = _calculateUniswapV2(midToken, destToken, rets, flags);
        return (rets, gas1 + gas2);
    }
    //knownsec// 内部函数用于计算兑换数量
    function _calculateUniswapV2(
        IERC20 fromToken,
        IERC20 destToken,
        uint256[] memory amounts,
        uint256 /*flags*/
    ) internal view returns(uint256[] memory rets, uint256 gas) {
        rets = new uint256[](amounts.length);
        //knownsec// 获取对应数字货币余额
        IERC20 fromTokenReal = fromToken.isBNB() ? wbnb : fromToken;
        IERC20 destTokenReal = destToken.isBNB() ? wbnb : destToken;
        address pairAddr = bakery.getPair(address(fromTokenReal), address(destTokenReal));
//knownsec// 获取对应交易对地址
        if (pairAddr != address(0)) {
            uint256 fromTokenBalance = fromTokenReal.universalBalanceOf(pairAddr);
            uint256 destTokenBalance = destTokenReal.universalBalanceOf(pairAddr);
            for (uint i = 0; i < amounts.length; i++) {
                rets[i] = _calculateUniswapFormula(fromTokenBalance, destTokenBalance,
amounts[i]);
```

```
            }
            return (rets, 50_000); //knownsec// 返回对应数值
        }
    }
    //knownsec// 计算兑换数量公式
    function _calculateUniswapFormula(uint256 fromBalance, uint256 toBalance, uint256 amount) internal pure returns(uint256) {
        if (amount == 0) {
            return 0;
        }
        return amount.mul(toBalance).mul(997).div(
            fromBalance.mul(1000).add(amount.mul(997))
        );
    }
    //knownsec// 代币兑换
    function _swapOnUniswapV2Internal(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 /*flags*/
    ) internal returns(uint256 returnAmount) {
        if (fromToken.isBNB()) {
            wbnb.deposit{value:amount}(); //knownsec// 进行 BNB 的 ERC20 代币兑换
        }

        IERC20 fromTokenReal = fromToken.isBNB() ? wbnb : fromToken;
        IERC20 toTokenReal = destToken.isBNB() ? wbnb : destToken;
        address pairAddr = bakery.getPair(address(fromTokenReal), address(toTokenReal));
        if(pairAddr != address(0)){
            IBakeryPair pair = IBakeryPair(pairAddr);
            bool needSync;
            bool needSkim;
            (returnAmount, needSync, needSkim) = pair.getReturn(fromTokenReal,
```

```
toTokenReal, amount);
                if (needSync) {
                    pair.sync();
                }
                else if (needSkim) {
                    pair.skim(skimAddress);
                }

                fromTokenReal.universalTransfer(pairAddr, amount);
                if (uint256(address(fromTokenReal)) < uint256(address(toTokenReal))) {
                    pair.swap(0, returnAmount, address(this), "");
                } else {
                    pair.swap(returnAmount, 0, address(this), "");
                }

                if (destToken.isBNB()) {
                    wbnb.withdraw(wbnb.balanceOf(address(this))); //knownsec// 兑换后进行对
应数量 BNB 的 ERC20 代币提现
                }
            }
        }

        function _swapOnUniswapV2OverMid(
            IERC20 fromToken,
            IERC20 midToken,
            IERC20 destToken,
            uint256 amount,
            uint256 flags
        ) internal {
            _swapOnUniswapV2Internal(
                midToken,
                destToken,
                _swapOnUniswapV2Internal(
```

```
                    fromToken,
                    midToken,
                    amount,
                    flags
                ),
                flags
            );
        }
}

//knownsec// BakeryBNB
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "../../interface/IExchange.sol";
import "./BakeryBase.sol";

contract BakeryBNB is IExchange, BakeryBase{

    uint256 internal constant FLAG_DISABLE_BAKERY_BNB = 0x4000000000000;

    function checkFlag(uint256 flags) external view override returns (bool){
        return flags.check(FLAG_DISABLE_BAKERY_ALL | FLAG_DISABLE_BAKERY_BNB);
    }

    /*
     * 计算兑换量
     */
    function calculate(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
```

```
        uint256 parts,
        uint256 flags
    ) external view override returns(uint256[] memory rets, uint256 gas) {
        if (fromToken.isBNB() || fromToken == wbnb || destToken.isBNB() || destToken == wbnb)
{
            return (new uint256[](parts), 0);
        }

        return _calculateUniswapV2OverMidToken(
            fromToken,
            wbnb,
            destToken,
            amount,
            parts,
            flags
        );
    }

    /*
     * 执行兑换
     */
    function swap(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 flags
    )external payable override {
        _getSwapAmount(fromToken, amount);
        _swapOnUniswapV2OverMid(
            fromToken,
            wbnb,
            destToken,
            amount,
```

```
                flags
            ); //knownsec// 兑换操作执行
            _swapReturnAmount(destToken); //knownsec// 对应代币获取
        }


}

//knownsec// BakeryLibrary
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/math/Math.sol";
import '../../lib/ERC20Lib.sol';
import './IBakeryPair.sol';

library BakeryLibrary {

    using Math for uint256;
    using SafeMath for uint256;
    using ERC20Lib for IERC20;
    //knownsec// 获取可兑换数量
    function getReturn(
        IBakeryPair pair,
        IERC20 fromToken,
        IERC20 destToken,
        uint amountIn
    ) internal view returns (uint256 result, bool needSync, bool needSkim) {
        uint256 reserveIn = fromToken.universalBalanceOf(address(pair));
        uint256 reserveOut = destToken.universalBalanceOf(address(pair));
        (uint112 reserve0, uint112 reserve1,) = pair.getReserves();
        if (fromToken > destToken) {
            (reserve0, reserve1) = (reserve1, reserve0);
```

```
            }
            needSync = (reserveIn < reserve0 || reserveOut < reserve1);
            needSkim = !needSync && (reserveIn > reserve0 || reserveOut > reserve1);

            uint256 amountInWithFee = amountIn.mul(997);
            uint256 numerator = amountInWithFee.mul(Math.min(reserveOut, reserve1));
            uint256 denominator = Math.min(reserveIn, reserve0).mul(1000).add(amountInWithFee);
            result = (denominator == 0) ? 0 : numerator.div(denominator);
        }
}

//knwonsec// IBakeryFactory
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

interface IBakeryFactory {
    function getPair(address tokenA, address tokenB) external view returns (address pair);
}

//knownsec// IbakeryPair
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

interface IBakeryPair {
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;
}

//knownsec// ExchangeBase
```

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import '../lib/ERC20Lib.sol';
import '../lib/FlagLib.sol';
import '../interface/IWBNB.sol';

contract ExchangeBase {

    using DisableFlags for uint256;

    using SafeMath for uint256;
    using ERC20Lib for IERC20;

    //wbnb
    IWBNB constant internal wbnb = IWBNB(0x6179c606c5796E9A17468838Bf6803E68bD49e00);
    //skim
    address constant skimAddress= 0x71819C78cc5c33A37460eE681ef39f98bfb4e5BA;
    //knownsec// 内部函数用于计算线性插值
    function _linearInterpolation(
        uint256 value,
        uint256 parts
    ) internal pure returns(uint256[] memory rets) {
        rets = new uint256[](parts);
        for (uint i = 0; i < parts; i++) {
            rets[i] = value.mul(i + 1).div(parts);
        }
    }
    //knownsec// 内部函数用于获取兑换数量的代币
    function _getSwapAmount(
        IERC20 fromToken,
        uint256 amount
```

```
    ) internal {
        fromToken.universalTransferFromSenderToThis(amount);    //knownsec//    调    用
universalTransferFromSenderToThis 函数用于对应数字货币的转账
    }
    //knownsec// 内部函数用于获取兑换返还代币
    function _swapReturnAmount(
        IERC20 destToken
    ) internal {
        destToken.universalTransfer(msg.sender, destToken.universalBalanceOf(address(this)));
//knownsec// 调用 universalTransfer 函数用于指定代币转账，转账数量为本合约中对应代币
余额
    }
}
```

安全建议：无。

4. 代码基本漏洞检测

4.1. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本

检测结果：经检测，智能合约代码中制定了编译器版本^0.6.12，不存在该安全问题。

安全建议：无。

4.2. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.3. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

检测结果：经检测，智能合约代码中已使用 SafeMath 安全算数库，不存在该安全问题。

安全建议：无。

4.4. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.5. require/assert 的合理使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.6. fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.7. tx.origin 身份验证【通过】

tx.origin 是 Solidity 的一个全局变量，它遍历整个调用栈并返回最初发送调用（或事务）的帐户的地址。在智能合约中使用此变量进行身份验证会使合约容易受到类似网络钓鱼的攻击。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.8. owner 权限控制【通过】

检查合约代码实现中的 owner 是否具有过高的权限。例如，任意修改其他账户余额等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.9. gas 消耗检测【通过】

检查 gas 的消耗是否超过区块最大限制

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.10. call 注入攻击【通过】

call 函数调用时，应该做严格的权限控制，或直接写死 call 调用的函数。

检测结果：经检测，智能合约不存在此漏洞。

安全建议：无。

4.11. 低级函数安全【通过】

检查合约代码实现中低级函数（call/delegatecall）的使用是否存在安全漏洞

call 函数的执行上下文是在被调用的合约中；而 delegatecall 函数的执行上下文是在当前调用该函数的合约中

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.12. 增发代币漏洞【通过】

检查在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.13. 访问控制缺陷检测【通过】

合约中不同函数应设置合理的权限

检查合约中各函数是否正确使用了 public、private 等关键词进行可见性修饰，检查合约是否正确定义并使用了 modifier 对关键函数进行访问限制，避免越权导致的问题。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.14. 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢。

Solidity 最多能处理 256 位的数字（2^256-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。

整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.15. 算术精度误差【通过】

Solidity 作为一门编程语言具备和普通编程语言相似的数据结构设计，比如：变量、常量、函数、数组、函数、结构体等等，Solidity 和普通编程语言也有一个较大的区别——Solidity 没有浮点型，且 Solidity 所有的数值运算结果都只会是整数，不会出现小数的情况，同时也不允许定义小数类型数据。合约中的数值运算必不可少，而数值运算的设计有可能造成相对误差，例如同级运算：5/2*10=20，而 5*10/2=25，从而产生误差，在数据更大时产生的误差也会更大，更明显。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.16. 错误使用随机数【通过】

智能合约中可能需要使用随机数，虽然 Solidity 提供的函数和变量可以访问明显难以预测的值，如 block.number 和 block.timestamp，但是它们通常或者比看起来更公开，或者受到矿工的影响，即这些随机数在一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.17. 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.18. 变量覆盖【通过】

检查合约代码实现中是否存在变量覆盖导致的安全问题

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.19. 未初始化的储存指针【通过】

在 solidity 中允许一个特殊的数据结构为 struct 结构体，而函数内的局部变量默认使用 storage 或 memory 储存。

而存在 storage(存储器)和 memory(内存)是两个不同的概念，solidity 允许指针指向一个未初始化的引用，而未初始化的局部 stroage 会导致变量指向其他储存变量，导致变量覆盖，甚至其他更严重的后果，在开发中应该避免在函数中初始化 struct 变量。

检测结果：经检测，智能合约代码不存在该问题。

安全建议：无。

4.20. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送 Ether，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；

传递所有可用 gas 进行调用（可通过传入 gas_value 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继续执行后面的代码，可能由于 Ether 发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.21. 交易顺序依赖【通过】

由于矿工总是通过代表外部拥有地址（EOA）的代码获取 gas 费用，因此用户可以指定更高的费用以便更快地开展交易。由于以太坊区块链是公开的，每个人都可以看到其他人未决交易的内容。这意味着，如果某个用户提交了一个有价值的解决方案，恶意用户可以窃取该解决方案并以较高的费用复制其交易，以抢占原始解决方案。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.22. 时间戳依赖攻击【通过】

数据块的时间戳通常来说都是使用矿工的本地时间，而这个时间大约能有 900 秒的范围波动，当其他节点接受一个新区块时，只需要验证时间戳是否晚于之前的区块并且与本地时间误差在 900 秒以内。一个矿工可以通过设置区块的时间戳来尽可能满足有利于他的条件来从中获利。

检查合约代码实现中是否存在有依赖于时间戳的关键功能

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.23. 拒绝服务攻击【通过】

在以太坊的世界中，拒绝服务是致命的，遭受该类型攻击的智能合约可能永远无法恢复正常工作状态。导致智能合约拒绝服务的原因可能有很多种，包括在作为交易接收方时的恶意行为，人为增加计算功能所需 gas 导致 gas 耗尽，滥用访问控制访问智能合约的 private 组件，利用混淆和疏忽等等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.24. 假充值漏洞【通过】

在代币合约的 transfer 函数对转账发起人(msg.sender)的余额检查用的是 if 判断方式，当 balances[msg.sender] < value 时进入 else 逻辑部分并 return false，最终没有抛出异常，我们认为仅 if/else 这种温和的判断方式在 transfer 这类敏感函数场景中是一种不严谨的编码方式。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.25. 重入攻击检测【通过】

重入漏洞是最著名的以太坊智能合约漏洞，曾导致了以太坊的分叉（The DAO hack）。

Solidity 中的 call.value()函数在被用来发送 Ether 的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送 Ether 的操作发生在实际减少发送者账户的余额之前时，就会存在重入攻击的风险。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

在资产管理体系中，常有委托管理的情况，委托人将资产给受托人管理，委托人支付一定的费用给受托人。这个业务场景在智能合约中也比较普遍。。

检测结果：经检测，智能合约不存在此漏洞。

安全建议：无。

4.27. 重排攻击检测【通过】

重排攻击是指矿工或其他方试图通过将自己的信息插入列表(list)或映射(mapping)中来与智能合约参与者进行“竞争”，从而使攻击者有机会将自己的信息存储到合约中。

检测结果:经检测，智能合约代码中不存在相关漏洞。

安全建议:无。

5. 附录 B：安全风险评级标准

智能合约漏洞评级标准	
漏洞评级	**漏洞评级说明**
高危漏洞	能直接造成代币合约或用户资金损失的漏洞，如：能造成代币价值归零的数值溢出漏洞、能造成交易所损失代币的假充值漏洞、能造成合约账户损失 ETH 或代币的重入漏洞等； 能造成代币合约归属权丢失的漏洞，如：关键函数的访问控制缺陷、call 注入导致关键函数访问控制绕过等； 能造成代币合约无法正常工作的漏洞，如：因向恶意地址发送 ETH 导致的拒绝服务漏洞、因 gas 耗尽导致的拒绝服务漏洞。
中危漏洞	需要特定地址才能触发的高风险漏洞，如代币合约拥有者才能触发的数值溢出漏洞等；非关键函数的访问控制缺陷、不能造成直接资金损失的逻辑设计缺陷等。
低危漏洞	难以被触发的漏洞、触发之后危害有限的漏洞，如需要大量 ETH 或代币才能触发的数值溢出漏洞、触发数值溢出后攻击者无法直接获利的漏洞、通过指定高 gas 触发的事务顺序依赖风险等。

6. 附录 C：智能合约安全审计工具简介

6.1 Manticore

Manticore 是一个分析二进制文件和智能合约的符号执行工具，Manticore 包含一个符号以太坊虚拟机（EVM），一个 EVM 反汇编器/汇编器以及一个用于自动编译和分析 Solidity 的方便界面。它还集成了 Ethersplay，用于 EVM 字节码的 Bit of Traits of Bits 可视化反汇编程序，用于可视化分析。 与二进制文件一样，Manticore 提供了一个简单的命令行界面和一个用于分析 EVM 字节码的 Python API。

6.2 Oyente

Oyente 是一个智能合约分析工具，Oyente 可以用来检测智能合约中常见的 bug，比如 reentrancy、事务排序依赖等等。更方便的是，Oyente 的设计是模块化的，所以这让高级用户可以实现并插入他们自己的检测逻辑，以检查他们的合约中自定义的属性。

7. 附录 A：合约代码

本次测试代码来源：

LongBitAudit.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/math/Math.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import './interface/ILongBitController.sol';
import './lib/ArrayLib.sol';
import './lib/ERC20Lib.sol';

contract LongBitAudit is Ownable{

    using SafeMath for uint256;
    using ERC20Lib for IERC20;
    using Array for IERC20[];

    //手续费比例
    uint256 public feeRatio;
    //
    address public feeAddress;
    //控制器
    ILongBitController public controller;

    struct Balances {
        uint256 ofFromToken;
        uint256 ofDestToken;
    }

    event Swapped(
        address account,
        IERC20 indexed fromToken,
        IERC20 indexed destToken,
        uint256 fromTokenAmount,
        uint256 destTokenAmount,
        uint256 minReturn,
        uint256[] distribution,
        uint256[] flags,
        address referral,
        uint256 feePercent
    );

    event ImplementationUpdated(address indexed newImpl);

    function setController(ILongBitController impl) public onlyOwner {
        controller = impl;
        emit ImplementationUpdated(address(impl));
    }

    function setFee(uint256 _feeRatio, address _feeAddress) public onlyOwner {
        feeRatio = _feeRatio;
        feeAddress = _feeAddress;
    }

    receive() payable external {
        require(msg.sender != tx.origin, "LongBit: do not send BNB directly");
    }

    /*
     * 获取估算兑换量
     */
    function getExpectedReturn(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    )
        public
        view
        returns(
            uint256 returnAmount,
            uint256[] memory distribution
        )
    {
        (returnAmount, , distribution) = getExpectedReturnWithGas(
```

```
            fromToken,
            destToken,
            amount,
            parts,
            flags,
            0
        );
    }

    /*
     * 获取估算兑换量 + 估算 gas 使用量
     */
    function getExpectedReturnWithGas(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags,
        uint256 destTokenEthPriceTimesGasPrice
    )
        public
        view
        returns(
            uint256 returnAmount,
            uint256 estimateGasAmount,
            uint256[] memory distribution
        )
    {
        return controller.getExpectedReturnWithGas(
            fromToken,
            destToken,
            amount,
            parts,
            flags,
            destTokenEthPriceTimesGasPrice
        );
    }

    /*
     * 获取多 token 估算兑换量 + 估算 gas 使用量
     */
    function getExpectedReturnWithGasMulti(
        IERC20[] memory tokens,
        uint256 amount,
        uint256[] memory parts,
        uint256[] memory flags,
        uint256[] memory destTokenEthPriceTimesGasPrices
    )
        public
        view
        returns(
            uint256[] memory returnAmounts,
            uint256 estimateGasAmount,
            uint256[] memory distribution
        )
    {
        uint256[] memory dist;

        returnAmounts = new uint256[](tokens.length - 1);
        for (uint i = 1; i < tokens.length; i++) {
            if (tokens[i - 1] == tokens[i]) {
                returnAmounts[i - 1] = (i == 1) ? amount : returnAmounts[i - 2];
                continue;
            }

            IERC20[] memory _tokens = tokens;

            (
                returnAmounts[i - 1],
                amount,
                dist
            ) = getExpectedReturnWithGas(
                _tokens[i - 1],
                _tokens[i],
                (i == 1) ? amount : returnAmounts[i - 2],
                parts[i - 1],
                flags[i - 1],
                destTokenEthPriceTimesGasPrices[i - 1]
            );
            estimateGasAmount = estimateGasAmount.add(amount);

            if (distribution.length == 0) {
                distribution = new uint256[](dist.length);
            }
            for (uint j = 0; j < distribution.length; j++) {
                distribution[j] = distribution[j].add(dist[j] << (8 * (i - 1)));
            }
```

```
        }
    }

    /*
     * 兑换
     */
    function swap(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 minReturn,
        uint256[] memory distribution,
        uint256 flags
    ) public payable returns(uint256) {
        return swapWithReferral(
            fromToken,
            destToken,
            amount,
            minReturn,
            distribution,
            flags,
            address(0),
            0
        );
    }

    /*
     * 兑换 + 推荐人功能
     */
    function swapWithReferral(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 minReturn,
        uint256[] memory distribution,
        uint256 flags, // See contants in IOneSplit.sol
        address referral,
        uint256 feePercent
    ) public payable returns(uint256) {
        IERC20[] memory tokens = new IERC20[](2);
        tokens[0] = fromToken;
        tokens[1] = destToken;

        uint256[] memory flagsArray = new uint256[](1);
        flagsArray[0] = flags;

        swapWithReferralMulti(
            tokens,
            amount,
            minReturn,
            distribution,
            flagsArray,
            referral,
            feePercent
        );
    }

    /*
     * 多 token 兑换
     */
    function swapMulti(
        IERC20[] memory tokens,
        uint256 amount,
        uint256 minReturn,
        uint256[] memory distribution,
        uint256[] memory flags
    ) public payable returns(uint256) {
        swapWithReferralMulti(
            tokens,
            amount,
            minReturn,
            distribution,
            flags,
            address(0),
            0
        );
    }

    /*
     * 多 token 兑换 + 推荐人功能
     */
    function swapWithReferralMulti(
        IERC20[] memory tokens,
        uint256 amount,
        uint256 minReturn,
        uint256[] memory distribution,
        uint256[] memory flags,
```

```
        address referral,
        uint256 feePercent
    ) public payable returns(uint256 returnAmount) {
        require(tokens.length >= 2 && amount > 0, "LongBit: swap makes no sense");
        require(flags.length == tokens.length - 1, "LongBit: flags array length is invalid");
        require((msg.value != 0) == tokens.first().isBNB(), "LongBit: msg.value should be used only for BNB swap");
        require(feePercent <= 0.03e18, "LongBit: feePercent out of range");
        require(msg.sender != referral, "LongBit:   sender and referral cannot be the same");

        Balances memory beforeBalances = _getFirstAndLastBalances(tokens, true);

        // Transfer From
        if (amount == uint256(-1)) {
            amount = Math.min(
                tokens.first().balanceOf(msg.sender),
                tokens.first().allowance(msg.sender, address(this))
            );
        }
        tokens.first().universalTransferFromSenderToThis(amount);//knownsec//  转入 token0
        uint256 confirmed = tokens.first().universalBalanceOf(address(this)).sub(beforeBalances.ofFromToken);//knownsec//  实际额

        // Swap
        tokens.first().universalApprove(address(controller), confirmed);
        controller.swapMulti{value:tokens.first().isBNB() ? confirmed : 0}(
            tokens,
            confirmed,
            minReturn,
            distribution,
            flags
        );

        Balances memory afterBalances = _getFirstAndLastBalances(tokens, false);

        // Return
        returnAmount = afterBalances.ofDestToken.sub(beforeBalances.ofDestToken);//knownsec//  兑换实际额
        require(returnAmount >= minReturn, "LongBit: actual return amount is less than minReturn");
        //手续费
        returnAmount = _mintFee(tokens, returnAmount);
        //转回
        tokens.last().universalTransfer(msg.sender, returnAmount);

        emit Swapped(
            msg.sender,
            tokens.first(),
            tokens.last(),
            amount,
            returnAmount,
            minReturn,
            distribution,
            flags,
            referral,
            feePercent
        );

        // Return remainder
        if (afterBalances.ofFromToken > beforeBalances.ofFromToken) {//knownsec//  转回剩余 fromToken
            tokens.first().universalTransfer(msg.sender, afterBalances.ofFromToken.sub(beforeBalances.ofFromToken));
        }
    }

    /*
     * 获取剩余使用量 + 返回量
     */
    function _getFirstAndLastBalances(IERC20[] memory tokens, bool subValue) internal view returns(Balances memory) {
        return Balances({
            ofFromToken: tokens.first().universalBalanceOf(address(this)).sub(subValue ? msg.value : 0),
            ofDestToken: tokens.last().universalBalanceOf(address(this))
        });
    }

    /*
     * 收取手续费
     */
    function _mintFee(IERC20[] memory tokens, uint256 returnAmount) internal returns(uint256 newAmount){
        uint256 feeAmount = returnAmount.mul(feeRatio).div(1e18);
        tokens.first().universalTransfer(feeAddress, feeAmount);
        newAmount = returnAmount.sub(feeAmount);//knownsec//  扣除手续费
    }
}
```

LongBitController.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/access/Ownable.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import './interface/ILongBitController.sol';
import "./interface/IExchange.sol";
import './lib/FlagLib.sol';
import './lib/ERC20Lib.sol';

contract LongBitController is ILongBitController, Ownable{

    using SafeMath for uint256;
    using DisableFlags for uint256;

    using ERC20Lib for IERC20;

    //交易所
    address[] public exchanges;
    //交易所数量
    uint256 internal DEXES_COUNT;
    //
    int256 internal constant VERY_NEGATIVE_VALUE = -1e72;
    //
    uint256 internal constant FLAG_DISABLE_SPLIT_RECALCULATION = 0x800000000000;
    //
    uint256 internal constant FLAG_DISABLE_ALL_SPLIT_SOURCES = 0x20000000;

    struct Args {
        IERC20 fromToken;
        IERC20 destToken;
        uint256 amount;
        uint256 parts;
        uint256 flags;
        uint256 destTokenEthPriceTimesGasPrice;
        int256[][] matrix;
        uint256[] gases;
        bool invert;
    }

    event print(
        uint256 amount
    );

    constructor() public {
    }

    function addExchange(address exchange) public onlyOwner {//knownsec//   添加交易所,仅 owner 调用
        exchanges.push(exchange);
        DEXES_COUNT = exchanges.length;
    }

    function setExchange(uint256 index, address exchange) public onlyOwner {
        exchanges[index] = exchange;
    }

    /*
     * 获取可兑换量 + 预估 gas + 交易所权重
     */
    function getExpectedReturnWithGas(//knownsec// 查询预估可获取量
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags,
        uint256 destTokenEthPriceTimesGasPrice
    )
        public
        view
        override
        returns(
            uint256 returnAmount,
            uint256 estimateGasAmount,
            uint256[] memory distribution
        )
    {
        if (fromToken == destToken) {
            return (amount, 0, new uint256[](DEXES_COUNT));
        }

        bool invert = flags.check(FLAG_DISABLE_ALL_SPLIT_SOURCES);
        int256[][] memory matrix = new int256[][](DEXES_COUNT);
        uint256[] memory gases = new uint256[](DEXES_COUNT);

        (returnAmount, estimateGasAmount, distribution) = _getAllReserves(
            Args({
                fromToken: fromToken,
```

```
                destToken: destToken,
                amount: amount,
                parts: parts,
                flags: flags,
                destTokenEthPriceTimesGasPrice: destTokenEthPriceTimesGasPrice,
                matrix: matrix,
                gases: gases,
                invert: invert
            })
        );
    }

    function _getAllReserves(Args memory args)//knownsec//  查询储量
        internal
        view
        returns(
            uint256 returnAmount,
            uint256 estimateGasAmount,
            uint256[] memory distribution
        )
    {

        bool atLeastOnePositive = false;
        for (uint i = 0; i < DEXES_COUNT; i++) {
            uint256[] memory rets;
            IExchange exchange = IExchange(exchanges[i]);
            (rets, args.gases[i]) = (args.invert != exchange.checkFlag(args.flags))?_calculateNoReturn(args.parts):exchange.calculate(args.fromToken, args.destToken, args.amount, args.parts, args.flags);

            // Prepend zero and sub gas
            int256 gas = int256(args.gases[i].mul(args.destTokenEthPriceTimesGasPrice).div(1e18));
            args.matrix[i] = new int256[](args.parts + 1);
            for (uint j = 0; j < rets.length; j++) {
                args.matrix[i][j + 1] = int256(rets[j]) - gas;
                atLeastOnePositive = atLeastOnePositive || (args.matrix[i][j + 1] > 0);
            }
        }

        if (!atLeastOnePositive) {
            for (uint i = 0; i < DEXES_COUNT; i++) {
                for (uint j = 1; j < args.parts + 1; j++) {
                    if (args.matrix[i][j] == 0) {
                        args.matrix[i][j] = VERY_NEGATIVE_VALUE;
                    }
                }
            }
        }

        (returnAmount, estimateGasAmount, distribution) = _getReturnAndGasByDistribution(args);
    }


    function _getReturnAndGasByDistribution(
        Args memory args
    ) internal view returns(uint256 returnAmount, uint256 estimateGasAmount, uint256[] memory distribution) {

        (, distribution) = _findBestDistribution(args.parts, args.matrix, DEXES_COUNT);

        for (uint i = 0; i < DEXES_COUNT; i++) {
            if (distribution[i] > 0) {
                if (distribution[i] == args.parts || args.flags.check(FLAG_DISABLE_SPLIT_RECALCULATION)) {
                    estimateGasAmount = estimateGasAmount.add(args.gases[i]);
                    int256 value = args.matrix[i][distribution[i]];
                    returnAmount = returnAmount.add(uint256(
                        (value == VERY_NEGATIVE_VALUE ? 0 : value) +
                        int256(args.gases[i].mul(args.destTokenEthPriceTimesGasPrice).div(1e18))
                    ));
                }
                else {
                    IExchange exchange = IExchange(exchanges[i]);
                    (uint256[] memory rets, uint256 gas) = (args.invert != exchange.checkFlag(args.flags))?_calculateNoReturn(args.parts):exchange.calculate(args.fromToken, args.destToken, args.amount.mul(distribution[i]).div(args.parts), 1, args.flags);
                    estimateGasAmount = estimateGasAmount.add(gas);
                    returnAmount = returnAmount.add(rets[0]);
                }
            }
        }
    }


    /*
     * 状态被禁用时的返回
     */
    function _calculateNoReturn(
```

```
        uint256 parts
    ) internal view returns(uint256[] memory rets, uint256 gas) {
        this;
        return (new uint256[](parts), 0);
    }

    function _findBestDistribution(
        uint256 s,
        int256[][] memory amounts,
        uint256 exchangeCount
    )
        internal
        pure
        returns(
            int256 returnAmount,
            uint256[] memory distribution
        )
    {
        uint256 n = amounts.length;
        int256[][] memory answer = new int256[][](n);
        uint256[][] memory parent = new uint256[][](n);

        for (uint i = 0; i < n; i++) {
            answer[i] = new int256[](s + 1);
            parent[i] = new uint256[](s + 1);
        }

        for (uint j = 0; j <= s; j++) {
            answer[0][j] = amounts[0][j];
            for (uint i = 1; i < n; i++) {
                answer[i][j] = -1e72;
            }
            parent[0][j] = 0;
        }

        for (uint i = 1; i < n; i++) {
            for (uint j = 0; j <= s; j++) {
                answer[i][j] = answer[i - 1][j];
                parent[i][j] = j;
                for (uint k = 1; k <= j; k++) {
                    if (answer[i - 1][j - k] + amounts[i][k] > answer[i][j]) {
                        answer[i][j] = answer[i - 1][j - k] + amounts[i][k];
                        parent[i][j] = j - k;
                    }
                }
            }
        }
        distribution = new uint256[](exchangeCount);
        uint256 partsLeft = s;
        for (uint curExchange = n - 1; partsLeft > 0; curExchange--) {
            distribution[curExchange] = partsLeft - parent[curExchange][partsLeft];
            partsLeft = parent[curExchange][partsLeft];
        }
        returnAmount = (answer[n - 1][s] == VERY_NEGATIVE_VALUE) ? 0 : answer[n - 1][s];
    }

    /*
     * 根据权重兑换
     */
    function swapMulti(
        IERC20[] memory tokens,
        uint256 amount,
        uint256 minReturn,
        uint256[] memory distribution,
        uint256[] memory flags
    ) public payable override returns(uint256 returnAmount) {
        tokens[0].universalTransferFrom(msg.sender, address(this), amount);//knownsec// 转入

        returnAmount = tokens[0].universalBalanceOf(address(this));
        for (uint i = 1; i < tokens.length; i++) {
            if (tokens[i - 1] == tokens[i]) {
                continue;
            }

            uint256[] memory dist = new uint256[](distribution.length);
            for (uint j = 0; j < distribution.length; j++) {
                dist[j] = (distribution[j] >> (8 * (i - 1))) & 0xFF;
            }

            _swap(
                tokens[i - 1],
                tokens[i],
                returnAmount,
                dist,
                flags[i - 1]
            );
```

```
                returnAmount = tokens[i].universalBalanceOf(address(this));
            }
            require(returnAmount >= minReturn, "Gswap: actual return amount is less than minReturn");
            tokens[tokens.length - 1].universalTransfer(msg.sender, returnAmount);//knownsec//  转出
        }


        function _swap(
            IERC20 fromToken,
            IERC20 destToken,
            uint256 amount,
            uint256[] memory distribution,
            uint256 flags
        ) internal returns(uint256 returnAmount) {
            require(distribution.length <= exchanges.length, "Gswap: Distribution array should not exceed reserves
array size");

            uint256 parts = 0;
            uint256 lastNonZeroIndex = 0;
            for (uint i = 0; i < distribution.length; i++) {
                if (distribution[i] > 0) {
                    parts = parts.add(distribution[i]);
                    lastNonZeroIndex = i;
                }
            }

            if (parts == 0) {
                if (fromToken.isBNB()) {
                    msg.sender.transfer(msg.value);
                    return msg.value;
                }
                return amount;
            }

            for (uint i = 0; i < distribution.length; i++) {
                if (distribution[i] == 0) {
                    continue;
                }

                uint256 swapAmount = amount.mul(distribution[i]).div(parts);//knownsec//  按占比取 amount
                if (i == lastNonZeroIndex) {
                    swapAmount = amount;
                }
                amount -= swapAmount;//knownsec//  无溢出
                IExchange exchange = IExchange(exchanges[i]);
                fromToken.universalApprove(address(exchange), swapAmount);
                exchange.swap{value:fromToken.isBNB() ? swapAmount : 0}(fromToken, destToken, swapAmount,
flags);
            }
        }
}
```

Migrations.sol

```
// SPDX-License-Identifier: MIT
pragma solidity >=0.4.22 <0.8.0;

contract Migrations {
  address public owner = msg.sender;
  uint public last_completed_migration;

  modifier restricted() {
    require(
      msg.sender == owner,
      "This function is restricted to the contract's owner"
    );
    _;
  }

  function setCompleted(uint completed) public restricted {
    last_completed_migration = completed;
  }
}
```

Bakery.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "../../interface/IExchange.sol";
import "./BakeryBase.sol";

contract Bakery is IExchange, BakeryBase{

    uint256 internal constant FLAG_DISABLE_BAKERY = 0x2000000000000;
```

```
    //knownsec// 检查flag 函数，用于检查传入flag 与指定标识一致性
    function checkFlag(uint256 flags) external view override returns (bool){
        return flags.check(FLAG_DISABLE_BAKERY_ALL | FLAG_DISABLE_BAKERY);
    }

    /*
     * 计算兑换量
     */
    function calculate(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) external view override returns(uint256[] memory rets, uint256 gas) {
        return _calculateUniswapV2(
            fromToken,
            destToken,
            _linearInterpolation(amount, parts), //knownsec// 获取对应的线性插值
            flags
        ); //knownsec// 调用BakeryBase 合约中的_calculateUniswapV2 函数查询兑换数量相关参数
    }

    /*
     * 执行兑换
     */
    function swap(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 flags
    )external payable override {
        _getSwapAmount(fromToken, amount); //knownsec// 兑换代币转账
        _swapOnUniswapV2Internal(
            fromToken,
            destToken,
            amount,
            flags
        ); //knownsec// 进行代币兑换
        _swapReturnAmount(destToken);
    }
}
```

BakeryBase.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import '../ExchangeBase.sol';
import './IBakeryFactory.sol';
import './IBakeryPair.sol';
import './BakeryLibrary.sol';

contract BakeryBase is ExchangeBase{

    using BakeryLibrary for IBakeryPair;

    uint256 constant FLAG_DISABLE_BAKERY_ALL = 0x1000000000000;

    //bakery
    IBakeryFactory constant internal bakery =
IBakeryFactory(0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f);


    function _calculateUniswapV2OverMidToken(
        IERC20 fromToken,
        IERC20 midToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) internal view returns(uint256[] memory rets, uint256 gas) {
        rets = _linearInterpolation(amount, parts);

        uint256 gas1;
        uint256 gas2;
        (rets, gas1) = _calculateUniswapV2(fromToken, midToken, rets, flags);
        (rets, gas2) = _calculateUniswapV2(midToken, destToken, rets, flags);
        return (rets, gas1 + gas2);
    }
    //knownsec// 内部函数用于计算兑换数量
    function _calculateUniswapV2(
        IERC20 fromToken,
        IERC20 destToken,
        uint256[] memory amounts,
        uint256 /*flags*/
```

```
    ) internal view returns(uint256[] memory rets, uint256 gas) {
        rets = new uint256[](amounts.length);
        //knownsec// 获取对应数字货币余额
        IERC20 fromTokenReal = fromToken.isBNB() ? wbnb : fromToken;
        IERC20 destTokenReal = destToken.isBNB() ? wbnb : destToken;
        address pairAddr = bakery.getPair(address(fromTokenReal), address(destTokenReal)); //knownsec// 获取对应交易对地址
        if (pairAddr != address(0)) {
            uint256 fromTokenBalance = fromTokenReal.universalBalanceOf(pairAddr);
            uint256 destTokenBalance = destTokenReal.universalBalanceOf(pairAddr);
            for (uint i = 0; i < amounts.length; i++) {
                rets[i] = _calculateUniswapFormula(fromTokenBalance, destTokenBalance, amounts[i]);
            }
            return (rets, 50_000); //knownsec// 返回对应数值
        }
    }
    //knownsec// 计算兑换数量公式
    function _calculateUniswapFormula(uint256 fromBalance, uint256 toBalance, uint256 amount) internal pure returns(uint256) {
        if (amount == 0) {
            return 0;
        }
        return amount.mul(toBalance).mul(997).div(
            fromBalance.mul(1000).add(amount.mul(997))
        );
    }
    //knownsec// 代币兑换
    function _swapOnUniswapV2Internal(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 /*flags*/
    ) internal returns(uint256 returnAmount) {
        if (fromToken.isBNB()) {
            wbnb.deposit{value:amount}(); //knownsec// 进行 BNB 的 ERC20 代币兑换
        }

        IERC20 fromTokenReal = fromToken.isBNB() ? wbnb : fromToken;
        IERC20 toTokenReal = destToken.isBNB() ? wbnb : destToken;
        address pairAddr = bakery.getPair(address(fromTokenReal), address(toTokenReal));
        if(pairAddr != address(0)){
            IBakeryPair pair = IBakeryPair(pairAddr);
            bool needSync;
            bool needSkim;
            (returnAmount, needSync, needSkim) = pair.getReturn(fromTokenReal, toTokenReal, amount);
            if (needSync) {
                pair.sync();
            }
            else if (needSkim) {
                pair.skim(skimAddress);
            }

            fromTokenReal.universalTransfer(pairAddr, amount);
            if (uint256(address(fromTokenReal)) < uint256(address(toTokenReal))) {
                pair.swap(0, returnAmount, address(this), "");
            } else {
                pair.swap(returnAmount, 0, address(this), "");
            }

            if (destToken.isBNB()) {
                wbnb.withdraw(wbnb.balanceOf(address(this))); //knownsec// 兑换后进行对应数量 BNB 的 ERC20 代币提现
            }
        }
    }

    function _swapOnUniswapV2OverMid(
        IERC20 fromToken,
        IERC20 midToken,
        IERC20 destToken,
        uint256 amount,
        uint256 flags
    ) internal {
        _swapOnUniswapV2Internal(
            midToken,
            destToken,
            _swapOnUniswapV2Internal(
                fromToken,
                midToken,
                amount,
                flags
            ),
            flags
        );
    }
}
```

BakeryBNB.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "../../interface/IExchange.sol";
import "./BakeryBase.sol";

contract BakeryBNB is IExchange, BakeryBase{

    uint256 internal constant FLAG_DISABLE_BAKERY_BNB = 0x4000000000000;

    function checkFlag(uint256 flags) external view override returns (bool){
        return flags.check(FLAG_DISABLE_BAKERY_ALL | FLAG_DISABLE_BAKERY_BNB);
    }

    /*
     * 计算兑换量
     */
    function calculate(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) external view override returns(uint256[] memory rets, uint256 gas) {
        if (fromToken.isBNB() || fromToken == wbnb || destToken.isBNB() || destToken == wbnb) {
            return (new uint256[](parts), 0);
        }

        return _calculateUniswapV2OverMidToken(
            fromToken,
            wbnb,
            destToken,
            amount,
            parts,
            flags
        );
    }

    /*
     * 执行兑换
     */
    function swap(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 flags
    )external payable override {
        _getSwapAmount(fromToken, amount);
        _swapOnUniswapV2OverMid(
            fromToken,
            wbnb,
            destToken,
            amount,
            flags
        ); //knownsec//  兑换操作执行
        _swapReturnAmount(destToken); //knownsec//  对应代币获取
    }


}
```

BakeryLibrary.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/math/Math.sol";
import '../../lib/ERC20Lib.sol';
import './IBakeryPair.sol';

library BakeryLibrary {

    using Math for uint256;
    using SafeMath for uint256;
    using ERC20Lib for IERC20;
    //knownsec//  获取可兑换数量
    function getReturn(
        IBakeryPair pair,
        IERC20 fromToken,
        IERC20 destToken,
        uint amountIn
    ) internal view returns (uint256 result, bool needSync, bool needSkim) {
        uint256 reserveIn = fromToken.universalBalanceOf(address(pair));
```

```
            uint256 reserveOut = destToken.universalBalanceOf(address(pair));
            (uint112 reserve0, uint112 reserve1,) = pair.getReserves();
            if (fromToken > destToken) {
                (reserve0, reserve1) = (reserve1, reserve0);
            }
            needSync = (reserveIn < reserve0 || reserveOut < reserve1);
            needSkim = !needSync && (reserveIn > reserve0 || reserveOut > reserve1);

            uint256 amountInWithFee = amountIn.mul(997);
            uint256 numerator = amountInWithFee.mul(Math.min(reserveOut, reserve1));
            uint256 denominator = Math.min(reserveIn, reserve0).mul(1000).add(amountInWithFee);
            result = (denominator == 0) ? 0 : numerator.div(denominator);
        }
    }
```

IBakeryFactory.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

interface IBakeryFactory {
    function getPair(address tokenA, address tokenB) external view returns (address pair);
}
```

IBakeryPair.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

interface IBakeryPair {
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;
}
```

Jul.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "../../interface/IExchange.sol";
import "./JulBase.sol";

contract Jul is IExchange, JulBase{

    uint256 internal constant FLAG_DISABLE_JUL = 0x01;

    function checkFlag(uint256 flags) external view override returns (bool){
        return flags.check(FLAG_DISABLE_JUL_ALL | FLAG_DISABLE_JUL);
    }

    /*
     * 计算兑换量
     */
    function calculate(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) external view override returns(uint256[] memory rets, uint256 gas) {
        return _calculateUniswapV2(
            fromToken,
            destToken,
            linearInterpolation(amount, parts),
            flags
        );
    }

    /*
     * 执行兑换
     */
    function swap(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 flags
    )external payable override {
        _getSwapAmount(fromToken, amount);
        _swapOnUniswapV2Internal(
            fromToken,
            destToken,
            amount,
```

```
                flags
            );
            _swapReturnAmount(destToken);
        }
    }
```

JulBase.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import '../ExchangeBase.sol';
import './IJulFactory.sol';
import './IJulPair.sol';
import './JulLibrary.sol';

contract JulBase is ExchangeBase{

    using JulLibrary for IJulPair;

    uint256 constant FLAG_DISABLE_JUL_ALL = 0x100000000000;

    //jul
    IJulFactory constant internal jul = IJulFactory(0x66D19a8a9DDBD2cECf39152634D8076aa33D4a63);


    function _calculateUniswapV2OverMidToken(
        IERC20 fromToken,
        IERC20 midToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) internal view returns(uint256[] memory rets, uint256 gas) {
        rets = _linearInterpolation(amount, parts);

        uint256 gas1;
        uint256 gas2;
        (rets, gas1) = _calculateUniswapV2(fromToken, midToken, rets, flags);
        (rets, gas2) = _calculateUniswapV2(midToken, destToken, rets, flags);
        return (rets, gas1 + gas2);
    }

    function _calculateUniswapV2(
        IERC20 fromToken,
        IERC20 destToken,
        uint256[] memory amounts,
        uint256 /*flags*/
    ) internal view returns(uint256[] memory rets, uint256 gas) {
        rets = new uint256[](amounts.length);

        IERC20 fromTokenReal = fromToken.isBNB() ? wbnb : fromToken;
        IERC20 destTokenReal = destToken.isBNB() ? wbnb : destToken;
        address pairAddr = jul.getPair(address(fromTokenReal), address(destTokenReal));
        if (pairAddr != address(0)) {
            uint256 fromTokenBalance = fromTokenReal.universalBalanceOf(pairAddr);
            uint256 destTokenBalance = destTokenReal.universalBalanceOf(pairAddr);
            for (uint i = 0; i < amounts.length; i++) {
                rets[i] = _calculateUniswapFormula(fromTokenBalance, destTokenBalance, amounts[i]);
            }
            return (rets, 50_000);
        }
    }

    function _calculateUniswapFormula(uint256 fromBalance, uint256 toBalance, uint256 amount) internal pure
returns(uint256) {
        if (amount == 0) {
            return 0;
        }
        return amount.mul(toBalance).mul(997).div(
            fromBalance.mul(1000).add(amount.mul(997))
        );
    }

    function _swapOnUniswapV2Internal(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 /*flags*/
    ) internal returns(uint256 returnAmount) {
        if (fromToken.isBNB()) {
            wbnb.deposit{value:amount}();
        }

        IERC20 fromTokenReal = fromToken.isBNB() ? wbnb : fromToken;
        IERC20 toTokenReal = destToken.isBNB() ? wbnb : destToken;
```

```
            address pairAddr = jul.getPair(address(fromTokenReal), address(toTokenReal));
            if(pairAddr != address(0)){
                IJulPair pair = IJulPair(pairAddr);
                bool needSync;
                bool needSkim;
                (returnAmount, needSync, needSkim) = pair.getReturn(fromTokenReal, toTokenReal, amount);
                if (needSync) {
                    pair.sync();
                }
                else if (needSkim) {
                    pair.skim(skimAddress);
                }

                fromTokenReal.universalTransfer(pairAddr, amount);
                if (uint256(address(fromTokenReal)) < uint256(address(toTokenReal))) {
                    pair.swap(0, returnAmount, address(this), "");
                } else {
                    pair.swap(returnAmount, 0, address(this), "");
                }

                if (destToken.isBNB()) {
                    wbnb.withdraw(wbnb.balanceOf(address(this)));
                }
            }
        }

        function _swapOnUniswapV2OverMid(
            IERC20 fromToken,
            IERC20 midToken,
            IERC20 destToken,
            uint256 amount,
            uint256 flags
        ) internal {
            _swapOnUniswapV2Internal(
                midToken,
                destToken,
                _swapOnUniswapV2Internal(
                    fromToken,
                    midToken,
                    amount,
                    flags
                ),
                flags
            );
        }
}
```

JulBNB.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "../../interface/IExchange.sol";
import "./JulBase.sol";

contract JulBNB is IExchange, JulBase{

    uint256 internal constant FLAG_DISABLE_JUL_BNB = 0x100000;

    function checkFlag(uint256 flags) external view override returns (bool){
        return flags.check(FLAG_DISABLE_JUL_ALL | FLAG_DISABLE_JUL_BNB);
    }

    /*
     * 计算兑换量
     */
    function calculate(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) external view override returns(uint256[] memory rets, uint256 gas) {
        if (fromToken.isBNB() || fromToken == wbnb || destToken.isBNB() || destToken == wbnb) {
            return (new uint256[](parts), 0);
        }

        return _calculateUniswapV2OverMidToken(
            fromToken,
            wbnb,
            destToken,
            amount,
            parts,
            flags
        );
    }
```

```
    /*
     * 执行兑换
     */
    function swap(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 flags
    )external payable override {
        _getSwapAmount(fromToken, amount);
        _swapOnUniswapV2OverMid(
            fromToken,
            wbnb,
            destToken,
            amount,
            flags
        );
        _swapReturnAmount(destToken);
    }
}
```

JulLibrary.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/math/Math.sol";
import '../../lib/ERC20Lib.sol';
import './IJulPair.sol';

library JulLibrary {

    using Math for uint256;
    using SafeMath for uint256;
    using ERC20Lib for IERC20;

    function getReturn(
        IJulPair pair,
        IERC20 fromToken,
        IERC20 destToken,
        uint amountIn
    ) internal view returns (uint256 result, bool needSync, bool needSkim) {
        uint256 reserveIn = fromToken.universalBalanceOf(address(pair));
        uint256 reserveOut = destToken.universalBalanceOf(address(pair));
        (uint112 reserve0, uint112 reserve1,) = pair.getReserves();
        if (fromToken > destToken) {
            (reserve0, reserve1) = (reserve1, reserve0);
        }
        needSync = (reserveIn < reserve0 || reserveOut < reserve1);
        needSkim = !needSync && (reserveIn > reserve0 || reserveOut > reserve1);

        uint256 amountInWithFee = amountIn.mul(997);
        uint256 numerator = amountInWithFee.mul(Math.min(reserveOut, reserve1));
        uint256 denominator = Math.min(reserveIn, reserve0).mul(1000).add(amountInWithFee);
        result = (denominator == 0) ? 0 : numerator.div(denominator);
    }
}
```

IJulFactory.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

interface IJulFactory {
    function getPair(address tokenA, address tokenB) external view returns (address pair);
}
```

IJulPair.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

interface IJulPair {
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;
}
```

Long.sol

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.6.12;

import "../../interface/IExchange.sol";
import "./LongBase.sol";

contract Long is IExchange, LongBase{

    uint256 internal constant FLAG_DISABLE_LONG = 0x2000000;

    function checkFlag(uint256 flags) external view override returns (bool){
        return flags.check(FLAG_DISABLE_LONG_ALL | FLAG_DISABLE_LONG);
    }

    /*
     * 计算兑换量
     */
    function calculate(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) external view override returns(uint256[] memory rets, uint256 gas) {
        return _calculateUniswapV2(
            fromToken,
            destToken,
            _linearInterpolation(amount, parts),
            flags
        );
    }

    /*
     * 执行兑换
     */
    function swap(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 flags
    )external payable override {
        _getSwapAmount(fromToken, amount);//knownsec// 转入 fromToken
        _swapOnUniswapV2Internal(//knownsec// 执行兑换
            fromToken,
            destToken,
            amount,
            flags
        );
        _swapReturnAmount(destToken);//knownsec// 转出本合约所有 destToken
    }
}
```

LongBase.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import '../ExchangeBase.sol';
import './ILongFactory.sol';
import './ILongPair.sol';
import './LongLibrary.sol';

contract LongBase is ExchangeBase{

    using LongLibrary for ILongPair;

    uint256 constant FLAG_DISABLE_LONG_ALL = 0x400000000000;

    //pancake
    ILongFactory constant internal long = ILongFactory(0xEAcc9d515477ab798187165E46d266A191C8e9b0);


    function _calculateUniswapV2OverMidToken(
        IERC20 fromToken,
        IERC20 midToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) internal view returns(uint256[] memory rets, uint256 gas) {
        rets = _linearInterpolation(amount, parts);

        uint256 gas1;
        uint256 gas2;
        (rets, gas1) = _calculateUniswapV2(fromToken, midToken, rets, flags);
        (rets, gas2) = _calculateUniswapV2(midToken, destToken, rets, flags);
        return (rets, gas1 + gas2);
```

```
    }

    function _calculateUniswapV2(
        IERC20 fromToken,
        IERC20 destToken,
        uint256[] memory amounts,
        uint256 /*flags*/
    ) internal view returns(uint256[] memory rets, uint256 gas) {
        rets = new uint256[](amounts.length);

        IERC20 fromTokenReal = fromToken.isBNB() ? wbnb : fromToken;
        IERC20 destTokenReal = destToken.isBNB() ? wbnb : destToken;
        address pairAddr = long.getPair(address(fromTokenReal), address(destTokenReal));
        if (pairAddr != address(0)) {
            uint256 fromTokenBalance = fromTokenReal.universalBalanceOf(pairAddr);
            uint256 destTokenBalance = destTokenReal.universalBalanceOf(pairAddr);
            for (uint i = 0; i < amounts.length; i++) {
                rets[i] = _calculateUniswapFormula(fromTokenBalance, destTokenBalance, amounts[i]);
            }
            return (rets, 50_000);
        }
    }

    function _calculateUniswapFormula(uint256 fromBalance, uint256 toBalance, uint256 amount) internal pure
returns(uint256) {//knownsec// 计算输出量
        if (amount == 0) {
            return 0;
        }
        return amount.mul(toBalance).mul(998).div(//knownsec// 千分之二手续费
            fromBalance.mul(1000).add(amount.mul(998))
        );
    }

    function _swapOnUniswapV2Internal(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 /*flags*/
    ) internal returns(uint256 returnAmount) {
        if (fromToken.isBNB()) {//knownsec// 若为BNB,则将 BNB 转为 WBNB
            wbnb.deposit{value:amount}();
        }

        IERC20 fromTokenReal = fromToken.isBNB() ? wbnb : fromToken;
        IERC20 toTokenReal = destToken.isBNB() ? wbnb : destToken;
        address pairAddr = long.getPair(address(fromTokenReal), address(toTokenReal));//knownsec// 取交易
对地址
        if(pairAddr != address(0)){
            ILongPair pair = ILongPair(pairAddr);
            bool needSync;
            bool needSkim;
            (returnAmount, needSync, needSkim) = pair.getReturn(fromTokenReal, toTokenReal, amount);
            if (needSync) {
                pair.sync();
            }
            else if (needSkim) {
                pair.skim(skimAddress);
            }

            fromTokenReal.universalTransfer(pairAddr, amount);//knownsec// fromToken 转入交易对地址
            if (uint256(address(fromTokenReal)) < uint256(address(toTokenReal))) {//knownsec// swap 兑换
                pair.swap(0, returnAmount, address(this), "");
            } else {
                pair.swap(returnAmount, 0, address(this), "");
            }

            if (destToken.isBNB()) {//knownsec// WBNB 赎回为 BNB
                wbnb.withdraw(wbnb.balanceOf(address(this)));
            }
        }
    }

    function _swapOnUniswapV2OverMid(//knownsec// fromToken<>midTken<>destToken
        IERC20 fromToken,
        IERC20 midToken,
        IERC20 destToken,
        uint256 amount,
        uint256 flags
    ) internal {
        _swapOnUniswapV2Internal(
            midToken,
            destToken,
            _swapOnUniswapV2Internal(
                fromToken,
                midToken,
                amount,
                flags
```

```
                ),
                flags
            );
        }
}
```

LongBNB.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "../../interface/IExchange.sol";
import "./LongBase.sol";

contract LongBNB is IExchange, LongBase{

    uint256 internal constant FLAG_DISABLE_LONG_BNB = 0x4000000;

    function checkFlag(uint256 flags) external view override returns (bool){
        return flags.check(FLAG_DISABLE_LONG_ALL | FLAG_DISABLE_LONG_BNB);
    }

    /*
     * 计算兑换量
     */
    function calculate(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) external view override returns(uint256[] memory rets, uint256 gas) {
        if (fromToken.isBNB() || fromToken == wbnb || destToken.isBNB() || destToken == wbnb) {
            return (new uint256[](parts), 0);
        }

        return _calculateUniswapV2OverMidToken(
            fromToken,
            wbnb,
            destToken,
            amount,
            parts,
            flags
        );
    }

    /*
     * 执行兑换
     */
    function swap(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 flags
    )external payable override{
        _getSwapAmount(fromToken, amount);
        _swapOnUniswapV2OverMid(
            fromToken,
            wbnb,
            destToken,
            amount,
            flags
        );
        _swapReturnAmount(destToken);
    }
}
```

LongLibrary.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/math/Math.sol";
import '../../lib/ERC20Lib.sol';
import './ILongPair.sol';

library LongLibrary {

    using Math for uint256;
    using SafeMath for uint256;
    using ERC20Lib for IERC20;

    function getReturn(//knownsec//  根据输入量获得输出量
        ILongPair pair,
        IERC20 fromToken,
```

```
        IERC20 destToken,
        uint amountIn
    ) internal view returns (uint256 result, bool needSync, bool needSkim) {
        uint256 reserveIn = fromToken.universalBalanceOf(address(pair));//knownsec// 交易对 fromToken 储量
        uint256 reserveOut = destToken.universalBalanceOf(address(pair));//knownsec// 交易对 destToken 储
量
        (uint112 reserve0, uint112 reserve1,) = pair.getReserves();
        if (fromToken > destToken) {
            (reserve0, reserve1) = (reserve1, reserve0);
        }
        needSync = (reserveIn < reserve0 || reserveOut < reserve1);
        needSkim = !needSync && (reserveIn > reserve0 || reserveOut > reserve1);

        uint256 amountInWithFee = amountIn.mul(998);
        uint256 numerator = amountInWithFee.mul(Math.min(reserveOut, reserve1));
        uint256 denominator = Math.min(reserveIn, reserve0).mul(1000).add(amountInWithFee);
        result = (denominator == 0) ? 0 : numerator.div(denominator);
    }
}
```

ILongFactory.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

interface ILongFactory {
    function getPair(address tokenA, address tokenB) external view returns (address pair);
}
```

ILongPair.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

interface ILongPair {
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;
}
```

Pancake.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "../../interface/IExchange.sol";
import "./PancakeBase.sol";

contract Pancake is IExchange, PancakeBase{

    uint256 internal constant FLAG_DISABLE_PANCAKE = 0x1000000;

    function checkFlag(uint256 flags) external view override returns (bool){
        return flags.check(FLAG_DISABLE_PANCAKE_ALL | FLAG_DISABLE_PANCAKE);
    }

    /*
     * 计算兑换量
     */
    function calculate(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) external view override returns(uint256[] memory rets, uint256 gas) {
        return _calculateUniswapV2(
            fromToken,
            destToken,
            linearInterpolation(amount, parts),
            flags
        );
    }

    /*
     * 执行兑换
     */
    function swap(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 flags
    )external payable override {
```

```
            _getSwapAmount(fromToken, amount);
            _swapOnUniswapV2Internal(
                fromToken,
                destToken,
                amount,
                flags
            );
            _swapReturnAmount(destToken);
        }
}
```

PancakeBase.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import '../ExchangeBase.sol';
import './IPancakeFactory.sol';
import './IPancakePair.sol';
import './PancakeLibrary.sol';

contract PancakeBase is ExchangeBase{

    using PancakeLibrary for IPancakePair;

    uint256 constant FLAG_DISABLE_PANCAKE_ALL = 0x8000000000000000;

    //pancake
    IPancakeFactory constant internal pancake =
IPancakeFactory(0xEAcc9d515477ab798187165E46d266A191C8e9b0);


    function _calculateUniswapV2OverMidToken(
        IERC20 fromToken,
        IERC20 midToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) internal view returns(uint256[] memory rets, uint256 gas) {
        rets = _linearInterpolation(amount, parts);

        uint256 gas1;
        uint256 gas2;
        (rets, gas1) = _calculateUniswapV2(fromToken, midToken, rets, flags);
        (rets, gas2) = _calculateUniswapV2(midToken, destToken, rets, flags);
        return (rets, gas1 + gas2);
    }

    function _calculateUniswapV2(
        IERC20 fromToken,
        IERC20 destToken,
        uint256[] memory amounts,
        uint256 /*flags*/
    ) internal view returns(uint256[] memory rets, uint256 gas) {
        rets = new uint256[](amounts.length);

        IERC20 fromTokenReal = fromToken.isBNB() ? wbnb : fromToken;
        IERC20 destTokenReal = destToken.isBNB() ? wbnb : destToken;
        address pairAddr = pancake.getPair(address(fromTokenReal), address(destTokenReal));
        if (pairAddr != address(0)) {
            uint256 fromTokenBalance = fromTokenReal.universalBalanceOf(pairAddr);
            uint256 destTokenBalance = destTokenReal.universalBalanceOf(pairAddr);
            for (uint i = 0; i < amounts.length; i++) {
                rets[i] = _calculateUniswapFormula(fromTokenBalance, destTokenBalance, amounts[i]);
            }
            return (rets, 50_000);
        }
    }

    function _calculateUniswapFormula(uint256 fromBalance, uint256 toBalance, uint256 amount) internal pure
returns(uint256) {
        if (amount == 0) {
            return 0;
        }
        return amount.mul(toBalance).mul(998).div(
            fromBalance.mul(1000).add(amount.mul(998))
        );
    }

    function _swapOnUniswapV2Internal(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 /*flags*/
    ) internal returns(uint256 returnAmount) {
```

```
            if (fromToken.isBNB()) {
                wbnb.deposit{value:amount}();
            }

            IERC20 fromTokenReal = fromToken.isBNB() ? wbnb : fromToken;
            IERC20 toTokenReal = destToken.isBNB() ? wbnb : destToken;
            address pairAddr = pancake.getPair(address(fromTokenReal), address(toTokenReal));
            if(pairAddr != address(0)){
                IPancakePair pair = IPancakePair(pairAddr);
                bool needSync;
                bool needSkim;
                (returnAmount, needSync, needSkim) = pair.getReturn(fromTokenReal, toTokenReal, amount);
                if (needSync) {
                    pair.sync();
                }
                else if (needSkim) {
                    pair.skim(skimAddress);
                }

                fromTokenReal.universalTransfer(pairAddr, amount);
                if (uint256(address(fromTokenReal)) < uint256(address(toTokenReal))) {
                    pair.swap(0, returnAmount, address(this), "");
                } else {
                    pair.swap(returnAmount, 0, address(this), "");
                }

                if (destToken.isBNB()) {
                    wbnb.withdraw(wbnb.balanceOf(address(this)));
                }
            }
        }

        function _swapOnUniswapV2OverMid(
            IERC20 fromToken,
            IERC20 midToken,
            IERC20 destToken,
            uint256 amount,
            uint256 flags
        ) internal {
            _swapOnUniswapV2Internal(
                midToken,
                destToken,
                _swapOnUniswapV2Internal(
                    fromToken,
                    midToken,
                    amount,
                    flags
                ),
                flags
            );
        }
    }
```

PancakeBNB.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "../../interface/IExchange.sol";
import "./PancakeBase.sol";

contract PancakeBNB is IExchange, PancakeBase{

    uint256 internal constant FLAG_DISABLE_PANCAKE_BNB = 0x10000000000000000;

    function checkFlag(uint256 flags) external view override returns (bool){
        return flags.check(FLAG_DISABLE_PANCAKE_ALL | FLAG_DISABLE_PANCAKE_BNB);
    }

    /*
     * 计算兑换量
     */
    function calculate(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) external view override returns(uint256[] memory rets, uint256 gas) {
        if (fromToken.isBNB() || fromToken == wbnb || destToken.isBNB() || destToken == wbnb) {
            return (new uint256[](parts), 0);
        }

        return _calculateUniswapV2OverMidToken(
            fromToken,
            wbnb,
```

```
                destToken,
                amount,
                parts,
                flags
            );
        }

        /*
         * 执行兑换
         */
        function swap(
            IERC20 fromToken,
            IERC20 destToken,
            uint256 amount,
            uint256 flags
        )external payable override{
            _getSwapAmount(fromToken, amount);
            _swapOnUniswapV2OverMid(
                fromToken,
                wbnb,
                destToken,
                amount,
                flags
            );
            _swapReturnAmount(destToken);
        }
}
```

PancakeLibrary.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/math/Math.sol";
import '../../lib/ERC20Lib.sol';
import './IPancakePair.sol';

library PancakeLibrary {

    using Math for uint256;
    using SafeMath for uint256;
    using ERC20Lib for IERC20;

    function getReturn(
        IPancakePair pair,
        IERC20 fromToken,
        IERC20 destToken,
        uint amountIn
    ) internal view returns (uint256 result, bool needSync, bool needSkim) {
        uint256 reserveIn = fromToken.universalBalanceOf(address(pair));
        uint256 reserveOut = destToken.universalBalanceOf(address(pair));
        (uint112 reserve0, uint112 reserve1,) = pair.getReserves();
        if (fromToken > destToken) {
            (reserve0, reserve1) = (reserve1, reserve0);
        }
        needSync = (reserveIn < reserve0 || reserveOut < reserve1);
        needSkim = !needSync && (reserveIn > reserve0 || reserveOut > reserve1);

        uint256 amountInWithFee = amountIn.mul(998);
        uint256 numerator = amountInWithFee.mul(Math.min(reserveOut, reserve1));
        uint256 denominator = Math.min(reserveIn, reserve0).mul(1000).add(amountInWithFee);
        result = (denominator == 0) ? 0 : numerator.div(denominator);
    }
}
```

IPancakeFactory.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

interface IPancakeFactory {
    function getPair(address tokenA, address tokenB) external view returns (address pair);
}
```

IPancakePair.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

interface IPancakePair {
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;
```

```
}
```

ExchangeBase.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import '../lib/ERC20Lib.sol';
import '../lib/FlagLib.sol';
import '../interface/IWBNB.sol';

contract ExchangeBase {

    using DisableFlags for uint256;

    using SafeMath for uint256;
    using ERC20Lib for IERC20;

    //wbnb
    IWBNB constant internal wbnb = IWBNB(0x6179c606c5796E9A17468838Bf6803E68bD49e00);
    //skim
    address constant skimAddress= 0x71819C78cc5c33A37460eE681ef39f98bfb4e5BA;
    //knownsec//  内部函数用于计算线性插值
    function _linearInterpolation(
        uint256 value,
        uint256 parts
    ) internal pure returns(uint256[] memory rets) {
        rets = new uint256[](parts);
        for (uint i = 0; i < parts; i++) {
            rets[i] = value.mul(i + 1).div(parts);
        }
    }
    //knownsec//  内部函数用于获取兑换数量的代币
    function _getSwapAmount(
        IERC20 fromToken,
        uint256 amount
    ) internal {
        fromToken.universalTransferFromSenderToThis(amount);          //knownsec//          调          用
universalTransferFromSenderToThis 函数用于对应数字货币的转账
    }
    //knownsec//  内部函数用于获取兑换返还代币
    function _swapReturnAmount(
        IERC20 destToken
    ) internal {
        destToken.universalTransfer(msg.sender, destToken.universalBalanceOf(address(this))); //knownsec//  调
用 universalTransfer 函数用于指定代币转账，转账数量为本合约中对应代币余额
    }
}
```

IExchange.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

interface IExchange {

    /*
     * 验证状态标识
     */
     function checkFlag(uint256 flags) external view returns (bool);

    /*
     * 计算兑换量
     */
    function calculate(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags
    ) external view returns(uint256[] memory rets, uint256 gas);

    /*
     * 执行兑换
     */
    function swap(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 flags
    )external payable;

}
```

ILongBitController.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

interface ILongBitController {

    function getExpectedReturnWithGas(
        IERC20 fromToken,
        IERC20 destToken,
        uint256 amount,
        uint256 parts,
        uint256 flags,
        uint256 destTokenEthPriceTimesGasPrice
    )
        external
        view
        returns(
            uint256 returnAmount,
            uint256 estimateGasAmount,
            uint256[] memory distribution
        );

    function swapMulti(
        IERC20[] calldata tokens,
        uint256 amount,
        uint256 minReturn,
        uint256[] calldata distribution,
        uint256[] calldata flags
    ) external payable returns(uint256 returnAmount);

}
```

IWBNB.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";


interface IWBNB is IERC20 {
    function deposit() external payable;
    function withdraw(uint256 amount) external;
}
```

ArrayLib.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";

library Array {
    function first(IERC20[] memory arr) internal pure returns(IERC20) {
        return arr[0];
    }

    function last(IERC20[] memory arr) internal pure returns(IERC20) {
        return arr[arr.length - 1];
    }
}
```

ERC20Lib.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/token/ERC20/SafeERC20.sol";


library ERC20Lib {

    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    IERC20 private constant ZERO_ADDRESS = IERC20(0x0000000000000000000000000000000000000000);
    IERC20 private constant BNB_ADDRESS = IERC20(0xEeeeeEeeeEeEeeEeEeEeeEEEeeeeEeeeeeeeEEeE);
```

```
    function universalTransfer(IERC20 token, address to, uint256 amount) internal returns(bool) {
        if (amount == 0) {
            return true;
        }

        if (isBNB(token)) {//knownsec// BNB 或 0 地址,则直接转去 BNB
            address(uint160(to)).transfer(amount);
        } else {//knownsec//  否则调用 safeTransfer
            token.safeTransfer(to, amount);
            return true;
        }
    }

    function universalTransferFrom(IERC20 token, address from, address to, uint256 amount) internal {
        if (amount == 0) {
            return;
        }

        if (isBNB(token)) {
            require(from == msg.sender && msg.value >= amount, "Wrong useage of
ETH.universalTransferFrom()");
            if (to != address(this)) {
                address(uint160(to)).transfer(amount);
            }
            if (msg.value > amount) {//knownsec//  返还多余部分
                msg.sender.transfer(msg.value.sub(amount));
            }
        } else {
            token.safeTransferFrom(from, to, amount);
        }
    }

    function universalTransferFromSenderToThis(IERC20 token, uint256 amount) internal {
        if (amount == 0) {
            return;
        }

        if (isBNB(token)) {
            if (msg.value > amount) {//knownsec//  返还多余部分
                // Return remainder if exist
                msg.sender.transfer(msg.value.sub(amount));
            }
        } else {
            token.safeTransferFrom(msg.sender, address(this), amount);
        }
    }

    function universalApprove(IERC20 token, address to, uint256 amount) internal {
        if (!isBNB(token)) {
            if (amount == 0) {
                token.safeApprove(to, 0);
                return;
            }

            uint256 allowance = token.allowance(address(this), to);
            if (allowance < amount) {
                if (allowance > 0) {
                    token.safeApprove(to, 0);
                }
                token.safeApprove(to, amount);
            }
        }
    }

    function universalBalanceOf(IERC20 token, address who) internal view returns (uint256) {
        if (isBNB(token)) {
            return who.balance;
        } else {
            return token.balanceOf(who);
        }
    }

    function universalDecimals(IERC20 token) internal view returns (uint256) {

        if (isBNB(token)) {
            return 18;
        }

        (bool success, bytes memory data) = address(token).staticcall{gas:10000}(
            abi.encodeWithSignature("decimals()")
        );
        if (!success || data.length == 0) {
            (success, data) = address(token).staticcall{gas:10000}(
                abi.encodeWithSignature("DECIMALS()")
            );
        }
```

```
        return (success && data.length > 0) ? abi.decode(data, (uint256)) : 18;
    }

    function isBNB(IERC20 token) internal pure returns(bool) {//knownsec// 0 地址或 BNB 返回 true
        return (address(token) == address(ZERO_ADDRESS) || address(token) == address(BNB_ADDRESS));
    }

    function eq(IERC20 a, IERC20 b) internal pure returns(bool) {
        return a == b || (isBNB(a) && isBNB(b));
    }

    function notExist(IERC20 token) internal pure returns(bool) {
        return (address(token) == address(-1));
    }
}
```

FlagLib.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.6.12;

library DisableFlags {
    function check(uint256 flags, uint256 flag) internal pure returns(bool) {
        return (flags & flag) != 0;
    }
}
```

6.3 securify.sh

Securify 可以验证以太坊智能合约常见的安全问题，例如交易乱序和缺少输入验证，它在全自动化的同时分析程序所有可能的执行路径，此外，Securify 还具有用于指定漏洞的特定语言，这使 Securify 能够随时关注当前的安全性和其他可靠性问题。

6.4 Echidna

Echidna 是一个为了对 EVM 代码进行模糊测试而设计的 Haskell 库。

6.5 MAIAN

MAIAN 是一个用于查找以太坊智能合约漏洞的自动化工具，Maian 处理合约的字节码，并尝试建立一系列交易以找出并确认错误。

6.6 ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

6.7 ida-evm

ida-evm 是一个针对以太坊虚拟机（EVM）的 IDA 处理器模块。

6.8 Remix-ide

Remix 是一款基于浏览器的编译器和 IDE，可让用户使用 Solidity 语言构建以太坊合约并调试交易。

6.9 知道创宇区块链安全审计人员专用工具包

知道创宇渗透测试人员专用工具包，由知道创宇渗透测试工程师研发，收集和使用，包含专用于测试人员的批量自动测试工具，自主研发的工具、脚本或利用工具等。

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587

邮　　箱 sec@knownsec.com

官　　网 www.knownsec.com

地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509