版本说明

修订内容	时间	修订者	版本号
编写文档	20210430	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
LongSwap 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 25 日开始到 2021 年 4 月 30 日结束，在此期间针对 **LongSwap 智能合约代码**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。

本次智能合约安全审计结果：

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
合约地址	
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

LongswapERC20.sol	a4d1d619032a719eb309d33fa8183165
LongswapFactory.sol	e0bbc2fd0ecc01882e896411a3653a54
LongswapPair.sol	c8c015f66b6336280051dd17f5ed29a5
IERC20.sol	3d2b948a9266ee66b6e739de97c7e6e9
ILongswapCallee.sol	21a8c1f3b171a441efb1b5a3f2e2df5f
ILongswapERC20.sol	49ec4de36c2502a967bcff7977e99c2c
ILongswapFactory.sol	1ce03f0689c2743e4a4100d8a4f11bad
ILongswapPair.sol	3c6e340f31416917349cf03c5dfa71f6
EnumerableSet.sol	4389cb806b15ba4aba514bcb59c15c19
Math.sol	bee6c1959728d5d42c14151076cebd83
SafeMath.sol	a65356c329efeffba75210664f5df3cd
UQ112x112.sol	2a703244e9f645c6fb297a983ce7c9c4
LPMining.sol	bab786023f10ebf3f174bc7f51f554f5
LbpToken.sol	4eec3a6a6c81092e8bee36d52915a697
LongswapRouter.sol	e320dc31568cb03cf5508879d984465e
MultMining.sol	31bbc7007267c94fd38e311208b98ae7
Oracle.sol	67e5b3d643bdb55e0ead0e754fc9f81e
SwapMining.sol	e6732889454df860fadab693a14bbaef
ILPMining.sol	269d7a7ef9dc6a641944d775a7b6942b
ILbp.sol	3764b652a746c68b8254c9441aa12905

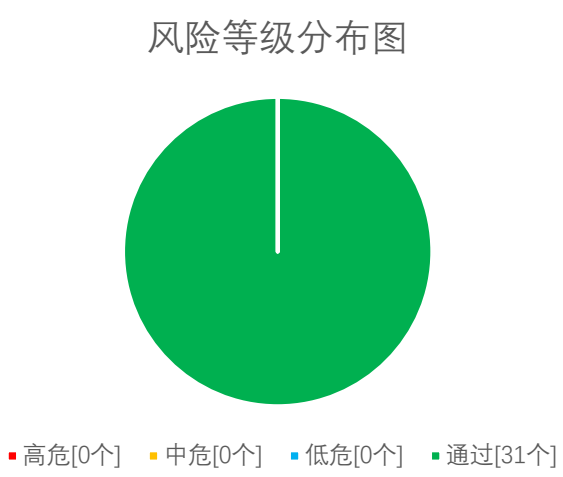
ILongswapFactory.sol	8e2ab86487f8b542efefd721fe8283fe
ILongswapPair.sol	b6194d61191f2606e44b478c3ec18148
ILongswapRouter.sol	fec864e9a2d35b24f8719548ab0a6c05
IOracle.sol	02c8d15c375ae36cf20074a4600ee362
ISwapMining.sol	ed31ec9c8a5a0484d4e86e250f1a854c
IWBNB.sol	59c49bc73c09c1a1e296837b08a46cee
OracleLibrary.sol	d0964b2a99a70adb49cde10fbe934b34
TransferHelper.sol	f4d52b980e6e15db47a2b6ee9e133b7f

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	30



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	流动性基础业务	通过	经检测，不存在安全问题。
	Router 合约业务功能	通过	经检测，不存在安全问题。
	流动性挖矿与兑换挖矿	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. 流动性基础业务【通过】

审计分析：LongSwap 项目实现了流动性基础业务功能，其中包含工厂合约 LongswapFactory 进行流动性交易对合约创建及部署，流动性交易对合约 LongswapPair 实现了流动性代币的铸造与销毁以及对应流动性代币的兑换基础功能，流动性代币基础合约 LongswapERC20 实现了一个标准 ERC20 代币及基本功能用于对应流动性代币的流通。

```
// knownsec // LongswapFactory
pragma solidity =0.5.16;

import './interfaces/ILongswapFactory.sol';
import './LongswapPair.sol';
import "./libraries/EnumerableSet.sol";

contract LongswapFactory is ILongswapFactory {

    using SafeMath for uint256;
    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _supportList;

    bytes32 public initCodeHash; // knownsec // pair 合约字节码

    address public feeTo;
    address public feeToSetter;
    uint256 public feeRateNumerator = 3; // knownsec // pair 手续费比例
    uint256 public constant FEE_RATE_DENOMINATOR = 1e3;

    mapping(address => mapping(address => address)) public getPair;
```

```
    address[] public allPairs;
    mapping(address => uint256) public pairFees;

    event PairCreated(address indexed token0, address indexed token1, address pair, uint);
    // knownsec //  构造函数初始化相关参数
    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
        initCodeHash = keccak256(abi.encodePacked(type(LongswapPair).creationCode));
    }
    // knownsec //  获取 pair 数量
    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }
    // knownsec //  创建 pair
    function createPair(address tokenA, address tokenB) external returns (address pair) {
        require(tokenA != tokenB, 'Longswap: IDENTICAL_ADDRESSES'); // knownsec //  交易对代币重复性检查
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'Longswap: ZERO_ADDRESS'); // knownsec //  非零地址检查
        require(getPair[token0][token1] == address(0), 'Longswap: PAIR_EXISTS'); // single check is sufficient
        bytes memory bytecode = type(LongswapPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt) // knownsec //  部署新的 pair
        }
        ILongswapPair(pair).initialize(token0, token1); // knownsec //  初始化 pair 流动性代币信息
        // knownsec //  确保 pair 双代币不重复创建
        getPair[token0][token1] = pair;
```

```
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair); // knownsec //  添加 pair
        emit PairCreated(token0, token1, pair, allPairs.length);
    }
    // knownsec //  设置 feeTo 地址
    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'Longswap: FORBIDDEN');
        feeTo = _feeTo;
    }
    // knownsec //  变更 setter 权限
    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'Longswap: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }
    // knownsec //  添加 pair 地址
    function addPair(address pair) external returns (bool){
        require(msg.sender == feeToSetter, 'LongSwap: FORBIDDEN');
        require(pair != address(0), 'LongSwap: pair is the zero address');
        return EnumerableSet.add(_supportList, pair);
    }
    // knownsec //  移除 pair 地址
    function delPair(address pair) external returns (bool){
        require(msg.sender == feeToSetter, 'LongSwap: FORBIDDEN');
        require(pair != address(0), 'LongSwap: pair is the zero address');
        return EnumerableSet.remove(_supportList, pair);
    }
    // knownsec //  获取项目支持 pair 数量
    function getSupportListLength() public view returns (uint256) {
        return EnumerableSet.length(_supportList);
    }
    // knownsec //  判断 pair 是否存在
    function isSupportPair(address pair) internal view returns (bool){
        return EnumerableSet.contains(_supportList, pair);
```

```
    }

    // Set pair fee , max is 1%
    function setPairFees(address pair, uint256 fee) external {
        require(msg.sender == feeToSetter, 'LongSwap: FORBIDDEN');
        require(fee <= 10, 'LongSwap: EXCEEDS_FEE_RATE_DENOMINATOR');
        //          require(isSupportPair(pair), 'LongswapFactory: Cannot operate on this pair');
        pairFees[pair] = fee;
    }
    // knownsec //  获取手续费参数
    function getPairFees(address pair) public view returns (uint256, uint256){
        require(pair != address(0), 'Longswap: pair is the zero address');
        if (isSupportPair(pair)) {
            return (pairFees[pair], FEE_RATE_DENOMINATOR);
        } else {
            return (feeRateNumerator, FEE_RATE_DENOMINATOR);
        }
    }

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

    // calculates the CREATE2 address for a pair without making any external calls
    function pairFor(address tokenA, address tokenB) public view returns (address pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        pair = address(uint(keccak256(abi.encodePacked(
                hex'ff',
```

```
                address(this),
                keccak256(abi.encodePacked(token0, token1)),
                initCodeHash
            ))));
    }

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

    // given some amount of an asset and pair reserves, returns an equivalent amount of the other asset
    function quote(uint amountA, uint reserveA, uint reserveB) public pure returns (uint amountB) {
        require(amountA > 0, 'Longswap: INSUFFICIENT_AMOUNT');
        require(reserveA > 0 && reserveB > 0, 'Longswap: INSUFFICIENT_LIQUIDITY');
        amountB = amountA.mul(reserveB) / reserveA;
    }

    // given an input amount of an asset and pair reserves, returns the maximum output amount of the other asset
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut, address token0, address token1) public view returns (uint amountOut) {
        require(amountIn > 0, 'Longswap: INSUFFICIENT_INPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'Longswap: INSUFFICIENT_LIQUIDITY');
        (uint256 fee,) = getPairFees(pairFor(token0, token1));
        uint amountInWithFee = amountIn.mul(FEE_RATE_DENOMINATOR.sub(fee));
        uint numerator = amountInWithFee.mul(reserveOut);
        uint                                denominator                                =
```

```
reserveIn.mul(FEE_RATE_DENOMINATOR).add(amountInWithFee);
        amountOut = numerator / denominator;
    }

    // given an output amount of an asset and pair reserves, returns a required input amount of the other asset
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut, address token0, address token1) public view returns (uint amountIn) {
        require(amountOut > 0, 'Longswap: INSUFFICIENT_OUTPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'Longswap: INSUFFICIENT_LIQUIDITY');
        (uint256 fee,) = getPairFees(pairFor(token0, token1));
        uint numerator = reserveIn.mul(amountOut).mul(FEE_RATE_DENOMINATOR);
        uint denominator = reserveOut.sub(amountOut).mul(FEE_RATE_DENOMINATOR.sub(fee));
        amountIn = (numerator / denominator).add(1);
    }
    // performs chained getAmountOut calculations on any number of pairs
    function getAmountsOut(uint amountIn, address[] memory path) public view returns (uint[] memory amounts) {
        require(path.length >= 2, 'Longswap: INVALID_PATH');
        amounts = new uint[](path.length);
        amounts[0] = amountIn;
        for (uint i; i < path.length - 1; i++) {
            (uint reserveIn, uint reserveOut) = getReserves(path[i], path[i + 1]);
            amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut, path[i], path[i + 1]);
        }
    }

    // performs chained getAmountIn calculations on any number of pairs
    function getAmountsIn(uint amountOut, address[] memory path) public view returns (uint[] memory amounts) {
        require(path.length >= 2, 'Longswap: INVALID_PATH');
```

```
            amounts = new uint[](path.length);
            amounts[amounts.length - 1] = amountOut;
            for (uint i = path.length - 1; i > 0; i--) {
                (uint reserveIn, uint reserveOut) = getReserves(path[i - 1], path[i]);
                amounts[i - 1] = getAmountIn(amounts[i], reserveIn, reserveOut, path[i - 1],
path[i]);
            }
        }
}

// knownsec // LongswapPair
pragma solidity =0.5.16;

import './interfaces/IERC20.sol';
import './interfaces/ILongswapFactory.sol';
import './interfaces/ILongswapPair.sol';
import './interfaces/ILongswapCallee.sol';
import './libraries/Math.sol';
import './libraries/UQ112x112.sol';
import './LongswapERC20.sol';

contract LongswapPair is ILongswapPair, LongswapERC20 {
    using SafeMath  for uint;
    using UQ112x112 for uint224;

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

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

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

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

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

    uint private unlocked = 1;
    modifier lock() {
        require(unlocked == 1, 'Longswap: LOCKED');
        unlocked = 0;
        _;
        unlocked = 1;
    }
    // knownsec //  获取 pair 储备
    function getReserves() public view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTimestampLast) {
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }
    // knownsec //  流动性代币转账函数
    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'Longswap: TRANSFER_FAILED');
    }

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
```

```
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);
    // knownsec //  通过 factory 部署后初始化本合约 factory 地址为调用合约地址
    constructor() public {
        factory = msg.sender;
    }
    // knownsec //  初始化交易对合约对应代币
    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external {
        require(msg.sender == factory, 'Longswap: FORBIDDEN'); // sufficient check
        token0 = _token0;
        token1 = _token1;
    }
    // knownsec //  私有函数更新
    // update reserves and, on the first call per block, price accumulators
    function _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1) private
{
        require(balance0 <= uint112(-1) && balance1 <= uint112(-1), 'Longswap:
OVERFLOW');
        uint32 blockTimestamp = uint32(block.timestamp % 2**32);
        uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is desired
        if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
            // * never overflows, and + overflow is desired
            price0CumulativeLast += uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) *
timeElapsed;
            price1CumulativeLast += uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) *
timeElapsed;
        }
```

```
        reserve0 = uint112(balance0);
        reserve1 = uint112(balance1);
        blockTimestampLast = blockTimestamp;
        emit Sync(reserve0, reserve1);
    }
    // knownsec //  流动性手续费
    // if fee is on, mint liquidity equivalent to 1/6th of the growth in sqrt(k)
    function _mintFee(uint112 _reserve0, uint112 _reserve1) private returns (bool feeOn) {
        address feeTo = ILongswapFactory(factory).feeTo();
        feeOn = feeTo != address(0);
        uint _kLast = kLast; // gas savings
        if (feeOn) {
            if (_kLast != 0) {
                uint rootK = Math.sqrt(uint(_reserve0).mul(_reserve1));
                uint rootKLast = Math.sqrt(_kLast);
                if (rootK > rootKLast) {
                    uint numerator = totalSupply.mul(rootK.sub(rootKLast));
                    uint denominator = rootK.mul(5).add(rootKLast);
                    uint liquidity = numerator / denominator;
                    if (liquidity > 0) _mint(feeTo, liquidity);
                }
            }
        } else if (_kLast != 0) {
            kLast = 0;
        }
    }

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

```
        uint amount0 = balance0.sub(_reserve0);
        uint amount1 = balance1.sub(_reserve1);

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

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

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

        bool feeOn = _mintFee(_reserve0, _reserve1);
```

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

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

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

        uint balance0;
        uint balance1;
        { // scope for _token{0,1}, avoids stack too deep errors
        address _token0 = token0;
```

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

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

    // force balances to match reserves
    function skim(address to) external lock {
        address _token0 = token0; // gas savings
```

```
        address _token1 = token1; // gas savings
        _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).sub(reserve0));
        _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
    }

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

// knownsec // LongswapERC20
pragma solidity =0.5.16;

import './interfaces/ILongswapERC20.sol';
import './libraries/SafeMath.sol';

contract LongswapERC20 is ILongswapERC20 {
    using SafeMath for uint;

    string public constant name = 'Longswap';
    string public constant symbol = 'LONG';
    uint8 public constant decimals = 18;
    uint  public totalSupply;
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint)) public allowance;

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

```
    mapping(address => uint) public nonces;

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

    constructor() public {
        uint chainId;
        assembly {
            chainId := chainid
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256('EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)'),
                keccak256(bytes(name)),
                keccak256(bytes('1')),
                chainId,
                address(this)
            )
        );
    }
    // knownsec //  内部函数铸币
    function _mint(address to, uint value) internal {
        totalSupply = totalSupply.add(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(address(0), to, value);
    }
    // knownsec //  内部函数销毁
    function _burn(address from, uint value) internal {
        balanceOf[from] = balanceOf[from].sub(value);
        totalSupply = totalSupply.sub(value);
        emit Transfer(from, address(0), value);
    }
```

```
    // knownsec //  内部函数授权
    function _approve(address owner, address spender, uint value) private {
        allowance[owner][spender] = value;
        emit Approval(owner, spender, value);
    }
    // knownsec //  内部函数转账
    function _transfer(address from, address to, uint value) private {
        balanceOf[from] = balanceOf[from].sub(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(from, to, value);
    }
    // knownsec //  授权函数
    function approve(address spender, uint value) external returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }
    // knownsec //  转账函数
    function transfer(address to, uint value) external returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
    }
    // knownsec //  代理转账函数
    function transferFrom(address from, address to, uint value) external returns (bool) {
        if (allowance[from][msg.sender] != uint(-1)) {
            allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);
        } // knownsec //  修改授权值
        _transfer(from, to, value);
        return true;
    }
    // knownsec //  签名授权
    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r,
bytes32 s) external {
        require(deadline >= block.timestamp, 'Longswap: EXPIRED');
```

```
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value, nonces[owner]++, deadline))
            )
        );
        address recoveredAddress = ecrecover(digest, v, r, s); // knownsec // 恢复签名地址
        require(recoveredAddress != address(0) && recoveredAddress == owner, 'Longswap: INVALID_SIGNATURE');
        _approve(owner, spender, value); // knownsec // 修改签名地址对指定地址的授权值
    }
}
```

安全建议：无。

3.2. Router 合约业务功能【通过】

审计分析：LongSwap 项目实现了 LongswapRouter 合约用于提供流动性添加与移除功能，不同代币兑换，交易兑换挖矿等功能。

```
// 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 './interfaces/IWBNB.sol';
import './interfaces/ILongswapFactory.sol';
import './interfaces/ILongswapPair.sol';
import './interfaces/ILongswapRouter.sol';
```

```
import './interfaces/ISwapMining.sol';
import './libraries/TransferHelper.sol';

contract LongswapRouter is ILongswapRouter, Ownable {
    using SafeMath for uint;

    address public immutable override factory;
    address public immutable override WBNB;
    address public swapMining;

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

    constructor(address _factory, address _WBNB) public {
        factory = _factory;
        WBNB = _WBNB;
    }

    function pairFor(address tokenA, address tokenB) public view returns (address pair){
        pair = ILongswapFactory(factory).pairFor(tokenA, tokenB);
    }

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

    // **** ADD LIQUIDITY ****
    function _addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
```

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

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

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

    function addLiquidityBNB(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountBNBMin,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline) returns (uint amountToken, uint amountBNB, uint liquidity) {
        (amountToken, amountBNB) = _addLiquidity(
            token,
            WBNB,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountBNBMin
        );
```

```
        address pair = pairFor(token, WBNB);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWBNB(WBNB).deposit{value: amountBNB}();
        assert(IWBNB(WBNB).transfer(pair, amountBNB));
        liquidity = ILongswapPair(pair).mint(to);
        // refund dust BNB, if any
        if (msg.value > amountBNB) TransferHelper.safeTransferBNB(msg.sender, msg.value - amountBNB);
    }

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

    function removeLiquidityBNB(
        address token,
        uint liquidity,
        uint amountTokenMin,
```

```
        uint amountBNBMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountToken, uint amountBNB) {
        (amountToken, amountBNB) = removeLiquidity(
            token,
            WBNB,
            liquidity,
            amountTokenMin,
            amountBNBMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, amountToken);
        IWBNB(WBNB).withdraw(amountBNB);
        TransferHelper.safeTransferBNB(to, amountBNB);
    }

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

```
    }

    function removeLiquidityBNBWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountBNBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountToken, uint amountBNB) {
        address pair = pairFor(token, WBNB);
        uint value = approveMax ? uint(-1) : liquidity;
        ILongswapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountToken, amountBNB) = removeLiquidityBNB(token, liquidity, amountTokenMin,
amountBNBMin, to, deadline);
    }

    // **** REMOVE LIQUIDITY (supporting fee-on-transfer tokens) ****
    function removeLiquidityBNBSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountBNBMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountBNB) {
        (, amountBNB) = removeLiquidity(
            token,
            WBNB,
            liquidity,
            amountTokenMin,
            amountBNBMin,
```

```
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, IERC20(token).balanceOf(address(this)));
        IWBNB(WBNB).withdraw(amountBNB);
        TransferHelper.safeTransferBNB(to, amountBNB);
    }

    function removeLiquidityBNBWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountBNBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountBNB) {
        address pair = pairFor(token, WBNB);
        uint value = approveMax ? uint(-1) : liquidity;
        ILongswapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        amountBNB = removeLiquidityBNBSupportingFeeOnTransferTokens(
            token, liquidity, amountTokenMin, amountBNBMin, to, deadline
        );
    }

    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, address[] memory path, address _to) internal virtual
{
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = ILongswapFactory(factory).sortTokens(input, output);
            uint amountOut = amounts[i + 1];
```

```
            if (swapMining != address(0)) {
                ISwapMining(swapMining).swap(msg.sender, input, output, amountOut);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) :
(amountOut, uint(0));
            address to = i < path.length - 2 ? pairFor(output, path[i + 2]) : _to;
            ILongswapPair(pairFor(input, output)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }

    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = ILongswapFactory(factory).getAmountsOut(amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'LongswapRouter:
INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }

    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
```

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

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

    function swapTokensForExactBNB(uint amountOut, uint amountInMax, address[] calldata path, address to, uint deadline)
        external
        virtual
```

```
        override
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WBNB, 'LongswapRouter: INVALID_PATH');
        amounts = ILongswapFactory(factory).getAmountsIn(amountOut, path);
        require(amounts[0] <= amountInMax, 'LongswapRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWBNB(WBNB).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferBNB(to, amounts[amounts.length - 1]);
    }

    function swapExactTokensForBNB(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
        external
        virtual
        override
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WBNB, 'LongswapRouter: INVALID_PATH');
        amounts = ILongswapFactory(factory).getAmountsOut(amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'LongswapRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWBNB(WBNB).withdraw(amounts[amounts.length - 1]);
```

```
        TransferHelper.safeTransferBNB(to, amounts[amounts.length - 1]);
    }

    function swapBNBForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
        external
        virtual
        override
        payable
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[0] == WBNB, 'LongswapRouter: INVALID_PATH');
        amounts = ILongswapFactory(factory).getAmountsIn(amountOut, path);
        require(amounts[0] <= msg.value, 'LongswapRouter: EXCESSIVE_INPUT_AMOUNT');
        IWBNB(WBNB).deposit{value: amounts[0]}();
        assert(IWBNB(WBNB).transfer(pairFor(path[0], path[1]), amounts[0]));
        _swap(amounts, path, to);
        // refund dust BNB, if any
        if (msg.value > amounts[0]) TransferHelper.safeTransferBNB(msg.sender, msg.value - amounts[0]);
    }

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) {
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amountIn
```

```
        );
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >=
amountOutMin,
            'LongswapRouter: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }

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

```
    }

    function swapExactTokensForBNBSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
        external
        virtual
        override
        ensure(deadline)
    {
        require(path[path.length - 1] == WBNB, 'LongswapRouter: INVALID_PATH');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, pairFor(path[0], path[1]), amountIn
        );
        _swapSupportingFeeOnTransferTokens(path, address(this));
        uint amountOut = IERC20(WBNB).balanceOf(address(this));
        require(amountOut >= amountOutMin, 'LongswapRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        IWBNB(WBNB).withdraw(amountOut);
        TransferHelper.safeTransferBNB(to, amountOut);
    }

    // **** SWAP (supporting fee-on-transfer tokens) ****
    // requires the initial amount to have already been sent to the first pair
    function _swapSupportingFeeOnTransferTokens(address[] memory path, address _to) internal virtual {
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = ILongswapFactory(factory).sortTokens(input, output);
```

```
            ILongswapPair pair = ILongswapPair(pairFor(input, output));
            uint amountInput;
            uint amountOutput;
            { // scope to avoid stack too deep errors
            (uint reserve0, uint reserve1,) = pair.getReserves();
            (uint reserveInput, uint reserveOutput) = input == token0 ? (reserve0, reserve1) :
(reserve1, reserve0);
            amountInput = IERC20(input).balanceOf(address(pair)).sub(reserveInput);
            amountOutput = ILongswapFactory(factory).getAmountOut(amountInput,
reserveInput, reserveOutput, input, output);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOutput) :
(amountOutput, uint(0));
            address to = i < path.length - 2 ? pairFor(output, path[i + 2]) : _to;
            pair.swap(amount0Out, amount1Out, to, new bytes(0));
        }
    }

    // **** LIBRARY FUNCTIONS ****
    function quote(
        uint256 amountA,
        uint256 reserveA,
        uint256 reserveB
    ) public view override returns (uint256 amountB) {
        return ILongswapFactory(factory).quote(amountA, reserveA, reserveB);
    }

    function getAmountOut(
        uint256 amountIn,
        uint256 reserveIn,
        uint256 reserveOut,
        address token0,
        address token1
```

```
    ) public view override returns (uint256 amountOut){
        return ILongswapFactory(factory).getAmountOut(amountIn, reserveIn, reserveOut, token0, token1);
    }

    function getAmountIn(
        uint256 amountOut,
        uint256 reserveIn,
        uint256 reserveOut,
        address token0,
        address token1
    ) public view override returns (uint256 amountIn){
        return ILongswapFactory(factory).getAmountIn(amountOut, reserveIn, reserveOut, token0, token1);
    }

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

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

安全建议：无。

3.3. 流动性挖矿与兑换挖矿【通过】

审计分析：LongSwap 项目实现了 LPMining、MultMining 合约用于流动性挖矿，以及 SwapMining 合约用于在进行代币兑换时进行兑换挖矿获取对应数量的 lbp 奖励代币，其中 Oracle 合约用于计算相应价格的兑换数量来进行挖矿。

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

import "@openzeppelin/contracts/access/Ownable.sol";
import '@openzeppelin/contracts/token/ERC20/ERC20.sol';
import "@openzeppelin/contracts/utils/EnumerableSet.sol";

contract LbpToken is ERC20, Ownable{

    EnumerableSet.AddressSet private _minters;

    uint256 private constant maxSupply = 1000000000 * 1e18;         // the total supply

    /**
     * @notice Constructs the HERC-20 contract.
     */
    constructor() public ERC20('LBP BSC', 'lbp') {}
    // knownsec //  添加 minter
    function addMinter(address _addMinter) public onlyOwner returns (bool) {
        require(_addMinter != address(0), "LbpToken: _addMinter is the zero address");
        return EnumerableSet.add(_minters, _addMinter);
    }
    // knownsec //  删除 minter
    function delMinter(address _delMinter) public onlyOwner returns (bool) {
        require(_delMinter != address(0), "LbpToken: _delMinter is the zero address");
```

```
        return EnumerableSet.remove(_minters, _delMinter);
    }
    // knownsec //  检查 minter
    function isMinter(address account) public view returns (bool) {
        return EnumerableSet.contains(_minters, account);
    }
    // knownsec //  铸币操作，限制铸币上限为 10 亿
    function mint(address _to, uint256 _amount) public onlyMinter returns (bool) {
        if (totalSupply().add(_amount) > maxSupply) {
            return false;
        }
        _mint(_to, _amount);
        return true;
    }
    // knownsec // minter 限制修饰器
    // modifier for mint function
    modifier onlyMinter() {
        require(isMinter(msg.sender), "caller is not the minter");
        _;
    }
}

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

import '@openzeppelin/contracts/access/Ownable.sol';
import '@openzeppelin/contracts/token/ERC20/IERC20.sol';
import '@openzeppelin/contracts/token/ERC20/SafeERC20.sol';
import '@openzeppelin/contracts/math/SafeMath.sol';
import './interfaces/ILPMining.sol';
import './interfaces/ILbp.sol';
```

```
contract LPMining is ILPMining, Ownable {

    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    // Info of each user.
    struct UserInfo {
        uint256 amount;         // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt. See explanation below.
    }

    // Info of each pool.
    struct PoolInfo {
        IERC20 lpToken;                 // Address of LP token contract.
        uint256 allocPoint;             // How many allocation points assigned to this pool. LBPs to
distribute per block.
        uint256 lastRewardBlock;    // Last block number that LBPs distribution occurs.
        uint256 accLbpPerShare; // Accumulated LBPs per share, times 1e12. See below.
    }

    // The LBP TOKEN!
    ILbp public lbp;
    // LBP tokens created per block.
    uint256 public lbpPerBlock;
    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping (uint256 => mapping (address => UserInfo)) public userInfo;
    // Total allocation poitns. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;
    // The block number when LBP mining starts.
    uint256 public startBlock;
```

```
event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);

constructor(
    ILbp _lbp,
    uint256 _lbpPerBlock,
    uint256 _startBlock
) public {
    lbp = _lbp;
    lbpPerBlock = _lbpPerBlock;
    startBlock = _startBlock;
}

// Add a new lp to the pool. Can only be called by the owner.
// XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolInfo.push(PoolInfo({
        lpToken: _lpToken,
        allocPoint: _allocPoint,
        lastRewardBlock: lastRewardBlock,
        accLbpPerShare: 0
    }));
}

// Update the given pool's LBP allocation point. Can only be called by the owner.
function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {
    if (_withUpdate) {
```

```
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // Update reward vairables for all pools. Be careful of gas spending!
    function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }

    // Update reward variables of the given pool to be up-to-date.
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return;
        }
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (lpSupply == 0) {
            pool.lastRewardBlock = block.number;
            return;
        }
        uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
        uint256                                    lbpReward                                    =
multiplier.mul(lbpPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
        pool.accLbpPerShare                                                                     =
pool.accLbpPerShare.add(lbpReward.mul(1e12).div(lpSupply));
        pool.lastRewardBlock = block.number;
    }
```

```
    function poolLength() external view returns (uint256) {
        return poolInfo.length;
    }

    // Deposit LP tokens to MiningPool for LBP allocation.
    function deposit(uint256 _pid, uint256 _amount) public override {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pendingAmount = user.amount.mul(pool.accLbpPerShare).div(1e12).sub(user.rewardDebt);
            safeLbpTransfer(msg.sender, pendingAmount);
        }
        pool.lpToken.safeTransferFrom(address(msg.sender), address(this), _amount);
        user.amount = user.amount.add(_amount);
        user.rewardDebt = user.amount.mul(pool.accLbpPerShare).div(1e12);
        emit Deposit(msg.sender, _pid, _amount);
    }

    // Withdraw LP tokens from MiningPool.
    function withdraw(uint256 _pid, uint256 _amount) public override {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount >= _amount, "withdraw: not good");
        updatePool(_pid);
        uint256 pendingAmount = user.amount.mul(pool.accLbpPerShare).div(1e12).sub(user.rewardDebt);
        safeLbpTransfer(msg.sender, pendingAmount);
        user.amount = user.amount.sub(_amount);
        user.rewardDebt = user.amount.mul(pool.accLbpPerShare).div(1e12);
        pool.lpToken.safeTransfer(address(msg.sender), _amount);
        emit Withdraw(msg.sender, _pid, _amount);
```

```
    }

    // Withdraw without caring about rewards. EMERGENCY ONLY.
    function emergencyWithdraw(uint256 _pid) public override {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        pool.lpToken.safeTransfer(address(msg.sender), user.amount);
        emit EmergencyWithdraw(msg.sender, _pid, user.amount);
        user.amount = 0;
        user.rewardDebt = 0;
    }

    // View function to see pending LBPs on frontend.
    function pending(uint256 _pid, address _user) external view override returns (uint256) {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accLbpPerShare = pool.accLbpPerShare;
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (block.number > pool.lastRewardBlock && lpSupply != 0) {
            uint256 multiplier = getMultiplier(pool.lastRewardBlock, block.number);
            uint256                                lbpReward                                =
multiplier.mul(lbpPerBlock).mul(pool.allocPoint).div(totalAllocPoint);
            accLbpPerShare = accLbpPerShare.add(lbpReward.mul(1e12).div(lpSupply));
        }
        return user.amount.mul(accLbpPerShare).div(1e12).sub(user.rewardDebt);
    }

    // Return reward multiplier over the given _from to _to block.
    function getMultiplier(uint256 _from, uint256 _to) public pure returns (uint256) {
        return _to.sub(_from);
    }

    // Safe lbp transfer function, just in case if rounding error causes pool to not have enough lbps.
```

```
    function safeLbpTransfer(address _to, uint256 _amount) internal {
        uint256 lbpBal = lbp.balanceOf(address(this));
        if (_amount > lbpBal) {
            lbp.mint(_to, lbpBal);
        } else {
            lbp.mint(_to, _amount);
        }
    }
}

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

import '@openzeppelin/contracts/access/Ownable.sol';
import '@openzeppelin/contracts/utils/EnumerableSet.sol';
import '@openzeppelin/contracts/token/ERC20/IERC20.sol';
import '@openzeppelin/contracts/token/ERC20/SafeERC20.sol';
import '@openzeppelin/contracts/math/SafeMath.sol';
import './interfaces/ILbp.sol';
import './interfaces/ILPMining.sol';

contract MultMining is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _multLP;

    // Info of each user.
    struct UserInfo {
        uint256 amount;         // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt.
```

```
        uint256 multLpRewardDebt; //multLp Reward debt.
    }

    // Info of each pool.
    struct PoolInfo {
        IERC20 lpToken;           // Address of LP token contract.
        uint256 allocPoint;       // How many allocation points assigned to this pool. LBPs to
distribute per block.
        uint256 lastRewardBlock;  // Last block number that LBPs distribution occurs.
        uint256 accLbpPerShare; // Accumulated LBPs per share, times 1e12.
        uint256 accMultLpPerShare; //Accumulated multLp per share
        uint256 totalAmount;    // Total amount of current pool deposit.
    }

    // The LBP Token!
    ILbp public lbp;
    // LBP tokens created per block.
    uint256 public lbpPerBlock;
    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;
    // Corresponding to the pid of the multLP pool
    mapping(uint256 => uint256) public poolCorrespond;
    // pid corresponding address
    mapping(address => uint256) public LpOfPid;
    // Control mining
    bool public paused = false;
    // Total allocation points. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;
    // The block number when LBP mining starts.
    uint256 public startBlock;
    // multLP MasterChef
```

```
address public multLpChef;
// multLP Token
address public multLpToken;
// How many blocks are halved
uint256 public halvingPeriod = 5256000;

event Deposit(address indexed user, uint256 indexed pid, uint256 amount);
event Withdraw(address indexed user, uint256 indexed pid, uint256 amount);
event EmergencyWithdraw(address indexed user, uint256 indexed pid, uint256 amount);

constructor(
    ILbp _lbp,
    uint256 _lbpPerBlock,
    uint256 _startBlock
) public {
    lbp = _lbp;
    lbpPerBlock = _lbpPerBlock;
    startBlock = _startBlock;
}

function setHalvingPeriod(uint256 _block) public onlyOwner {
    halvingPeriod = _block;
}

// Set the number of lbp produced by each block
function setLbpPerBlock(uint256 _newPerBlock) public onlyOwner {
    massUpdatePools();
    lbpPerBlock = _newPerBlock;
}

function poolLength() public view returns (uint256) {
    return poolInfo.length;
}
```

```
    // knownsec // 添加挖矿池
    function addMultLP(address _addLP) public onlyOwner returns (bool) {
        require(_addLP != address(0), 'LP is the zero address');
        IERC20(_addLP).approve(multLpChef, uint256(- 1));
        return EnumerableSet.add(_multLP, _addLP);
    }
    // knownsec // 检查挖矿池是否存在
    function isMultLP(address _LP) public view returns (bool) {
        return EnumerableSet.contains(_multLP, _LP);
    }
    // knownsec // 查询挖矿池数量
    function getMultLPLength() public view returns (uint256) {
        return EnumerableSet.length(_multLP);
    }
    // knownsec // 查询指定 id 对应地址
    function getMultLPAddress(uint256 _pid) public view returns (address){
        require(_pid <= getMultLPLength() - 1, 'not find this multLP');
        return EnumerableSet.at(_multLP, _pid);
    }
    // knownsec // 设置暂停
    function setPause() public onlyOwner {
        paused = !paused;
    }
    // knownsec // 更新挖矿合约地址
    function setMultLP(address _multLpToken, address _multLpChef) public onlyOwner {
        require(_multLpToken != address(0) && _multLpChef != address(0), 'is the zero
address');
        multLpToken = _multLpToken;
        multLpChef = _multLpChef;
    }
    // knownsec // 替换挖矿合约相关地址
    function replaceMultLP(address _multLpToken, address _multLpChef) public onlyOwner {
        require(_multLpToken != address(0) && _multLpChef != address(0), 'is the zero
```

```
address');
        require(paused == true, 'No mining suspension');
        // knownsec //  变更挖矿合约地址
        multLpToken = _multLpToken;
        multLpChef = _multLpChef;
        uint256 length = getMultLPLength();
        // knownsec //  遍历紧急提取抵押资产
        while (length > 0) {
            address dAddress = EnumerableSet.at(_multLP, 0);
            uint256 pid = LpOfPid[dAddress];
            ILPMining(multLpChef).emergencyWithdraw(poolCorrespond[pid]);
            EnumerableSet.remove(_multLP, dAddress);
            length--;
        }
    }

    // Add a new lp to the pool. Can only be called by the owner.
    // XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
    function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {
        require(address(_lpToken) != address(0), '_lpToken is the zero address');
        if (_withUpdate) {
            massUpdatePools();
        }
        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
        lpToken : _lpToken,
        allocPoint : _allocPoint,
        lastRewardBlock : lastRewardBlock,
        accLbpPerShare : 0,
        accMultLpPerShare : 0,
        totalAmount : 0
        }));
```

```
        LpOfPid[address(_lpToken)] = poolLength() - 1;
    }

    // Update the given pool's LBP allocation point. Can only be called by the owner.
    function set(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
            massUpdatePools();
        }
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // The current pool corresponds to the pid of the multLP pool
    function setPoolCorr(uint256 _pid, uint256 _sid) public onlyOwner {
        require(_pid <= poolLength() - 1, 'not find this pool');
        poolCorrespond[_pid] = _sid;
    }

    function phase(uint256 blockNumber) public view returns (uint256) {
        if (halvingPeriod == 0) {
            return 0;
        }
        if (blockNumber > startBlock) {
            return (blockNumber.sub(startBlock).sub(1)).div(halvingPeriod);
        }
        return 0;
    }

    function reward(uint256 blockNumber) public view returns (uint256) {
        uint256 _phase = phase(blockNumber);
        return lbpPerBlock.div(2 ** _phase);
    }
```

```
    function getLbpBlockReward(uint256 _lastRewardBlock) public view returns (uint256) {
        uint256 blockReward = 0;
        uint256 n = phase(_lastRewardBlock);
        uint256 m = phase(block.number);
        while (n < m) {
            n++;
            uint256 r = n.mul(halvingPeriod).add(startBlock);
            blockReward = blockReward.add((r.sub(_lastRewardBlock)).mul(reward(r)));
            _lastRewardBlock = r;
        }
        blockReward = blockReward.add((block.number.sub(_lastRewardBlock)).mul(reward(block.number)));
        return blockReward;
    }

    // Update reward variables for all pools. Be careful of gas spending!
    function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }

    // Update reward variables of the given pool to be up-to-date.
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return;
        }
        uint256 lpSupply;
        if (isMultLP(address(pool.lpToken))) {
            if (pool.totalAmount == 0) {
                pool.lastRewardBlock = block.number;
```

```
                return;
            }
            lpSupply = pool.totalAmount;
        } else {
            lpSupply = pool.lpToken.balanceOf(address(this));
            if (lpSupply == 0) {
                pool.lastRewardBlock = block.number;
                return;
            }
        }
        uint256 blockReward = getLbpBlockReward(pool.lastRewardBlock);
        if (blockReward <= 0) {
            return;
        }
        uint256 lbpReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
        bool minRet = lbp.mint(address(this), lbpReward);
        if (minRet) {
            pool.accLbpPerShare                                                    =
pool.accLbpPerShare.add(lbpReward.mul(1e12).div(lpSupply));
        }
        pool.lastRewardBlock = block.number;
    }

    // View function to see pending LBPs on frontend.
    function pending(uint256 _pid, address _user) external view returns (uint256, uint256){
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            (uint256 lbpAmount, uint256 tokenAmount) = pendingLbpAndToken(_pid, _user);
            return (lbpAmount, tokenAmount);
        } else {
            uint256 lbpAmount = pendingLbp(_pid, _user);
            return (lbpAmount, 0);
        }
```

```
    }
    // knownsec // 查询奖励
    function pendingLbpAndToken(uint256 _pid, address _user) private view returns (uint256, uint256){
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accLbpPerShare = pool.accLbpPerShare;
        uint256 accMultLpPerShare = pool.accMultLpPerShare;
        if (user.amount > 0) {
            uint256 TokenPending = ILPMining(multLpChef).pending(poolCorrespond[_pid], address(this));
            accMultLpPerShare = accMultLpPerShare.add(TokenPending.mul(1e12).div(pool.totalAmount));
            uint256 userPending = user.amount.mul(accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
            if (block.number > pool.lastRewardBlock) {
                uint256 blockReward = getLbpBlockReward(pool.lastRewardBlock);
                uint256 lbpReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
                accLbpPerShare = accLbpPerShare.add(lbpReward.mul(1e12).div(pool.totalAmount));
                return (user.amount.mul(accLbpPerShare).div(1e12).sub(user.rewardDebt), userPending);
            }
            if (block.number == pool.lastRewardBlock) {
                return (user.amount.mul(accLbpPerShare).div(1e12).sub(user.rewardDebt), userPending);
            }
        }
        return (0, 0);
    }
    // knownsec // 查询奖励
    function pendingLbp(uint256 _pid, address _user) private view returns (uint256){
        PoolInfo storage pool = poolInfo[_pid];
```

```
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accLbpPerShare = pool.accLbpPerShare;
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (user.amount > 0) {
            if (block.number > pool.lastRewardBlock) {
                uint256 blockReward = getLbpBlockReward(pool.lastRewardBlock);
                uint256 lbpReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
                accLbpPerShare = accLbpPerShare.add(lbpReward.mul(1e12).div(lpSupply));
                return user.amount.mul(accLbpPerShare).div(1e12).sub(user.rewardDebt);
            }
            if (block.number == pool.lastRewardBlock) {
                return user.amount.mul(accLbpPerShare).div(1e12).sub(user.rewardDebt);
            }
        }
        return 0;
    }

    // Deposit LP tokens to HecoPool for LBP allocation.
    function deposit(uint256 _pid, uint256 _amount) public notPause {
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            depositLbpAndToken(_pid, _amount, msg.sender);
        } else {
            depositLbp(_pid, _amount, msg.sender);
        }
    }
    // knownsec // 抵押 lbp 和流动性代币
    function depositLbpAndToken(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);
        // knownsec // 已经存在抵押进行领取奖励
        if (user.amount > 0) {
```

```
            uint256 pendingAmount = user.amount.mul(pool.accLbpPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeLbpTransfer(_user, pendingAmount); // knownsec // 领取奖励
            }
            uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
            ILPMining(multLpChef).deposit(poolCorrespond[_pid], 0); // knownsec // 先前提供的暂不抵押
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare = pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
            uint256 tokenPending = user.amount.mul(pool.accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
            if (tokenPending > 0) {
                IERC20(multLpToken).safeTransfer(_user, tokenPending);
            }
        }
        // knownsec // 新抵押进池子并进行流动性挖矿
        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(_user, address(this), _amount);
            if (pool.totalAmount == 0) {
                ILPMining(multLpChef).deposit(poolCorrespond[_pid], _amount);
                user.amount = user.amount.add(_amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
            } else {
                uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
                ILPMining(multLpChef).deposit(poolCorrespond[_pid], _amount);
                uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
                pool.accMultLpPerShare = pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
                user.amount = user.amount.add(_amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
            }
```

```
        }
        user.rewardDebt = user.amount.mul(pool.accLbpPerShare).div(1e12);
        user.multLpRewardDebt = user.amount.mul(pool.accMultLpPerShare).div(1e12);
        emit Deposit(_user, _pid, _amount);
    }
    // knownsec // 单抵押流动性代币但不进行挖矿抵押
    function depositLbp(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pendingAmount = user.amount.mul(pool.accLbpPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeLbpTransfer(_user, pendingAmount);
            }
        }
        // knownsec // 抵押流动性代币
        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(_user, address(this), _amount);
            user.amount = user.amount.add(_amount);
            pool.totalAmount = pool.totalAmount.add(_amount);
        }
        user.rewardDebt = user.amount.mul(pool.accLbpPerShare).div(1e12); // knownsec // 更新负债
        emit Deposit(_user, _pid, _amount);
    }

    // Withdraw LP tokens from HecoPool.
    function withdraw(uint256 _pid, uint256 _amount) public notPause {
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            withdrawLbpAndToken(_pid, _amount, msg.sender);
```

```
        } else {
            withdrawLbp(_pid, _amount, msg.sender);
        }
    }
    // knownsec //  提取抵押流动性代币以及对应奖励
    function withdrawLbpAndToken(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        require(user.amount >= _amount, 'withdrawLbpAndToken: not good');
        updatePool(_pid);
        uint256 pendingAmount =
user.amount.mul(pool.accLbpPerShare).div(1e12).sub(user.rewardDebt);
        // knownsec //  领取奖励
        if (pendingAmount > 0) {
            safeLbpTransfer(_user, pendingAmount);
        }
        if (_amount > 0) {
            uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
            ILPMining(multLpChef).withdraw(poolCorrespond[_pid], _amount);
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare =
pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
            uint256 tokenPending =
user.amount.mul(pool.accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
            if (tokenPending > 0) {
                IERC20(multLpToken).safeTransfer(_user, tokenPending);
            }
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            pool.lpToken.safeTransfer(_user, _amount);
        }
        user.rewardDebt = user.amount.mul(pool.accLbpPerShare).div(1e12);
        user.multLpRewardDebt = user.amount.mul(pool.accMultLpPerShare).div(1e12);
```

```
        emit Withdraw(_user, _pid, _amount);
    }
    // knownsec //  提取抵押流动性代币
    function withdrawLbp(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        require(user.amount >= _amount, 'withdrawLbp: not good');
        updatePool(_pid);
        uint256                                pendingAmount                                =
user.amount.mul(pool.accLbpPerShare).div(1e12).sub(user.rewardDebt);
        if (pendingAmount > 0) {
            safeLbpTransfer(_user, pendingAmount);
        }
        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            pool.lpToken.safeTransfer(_user, _amount);
        }
        user.rewardDebt = user.amount.mul(pool.accLbpPerShare).div(1e12);
        emit Withdraw(_user, _pid, _amount);
    }

    // Withdraw without caring about rewards. EMERGENCY ONLY.
    function emergencyWithdraw(uint256 _pid) public notPause {
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            emergencyWithdrawLbpAndToken(_pid, msg.sender);
        } else {
            emergencyWithdrawLbp(_pid, msg.sender);
        }
    }
    // knownsec //  紧急提取
    function emergencyWithdrawLbpAndToken(uint256 _pid, address _user) private {
```

```
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 amount = user.amount;
        uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
        ILPMining(multLpChef).withdraw(poolCorrespond[_pid], amount);
        uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
        pool.accMultLpPerShare =
pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
        user.amount = 0;
        user.rewardDebt = 0;
        pool.lpToken.safeTransfer(_user, amount);
        pool.totalAmount = pool.totalAmount.sub(amount);
        emit EmergencyWithdraw(_user, _pid, amount);
    }
    // knownsec //  紧急提取
    function emergencyWithdrawLbp(uint256 _pid, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        pool.lpToken.safeTransfer(_user, amount);
        pool.totalAmount = pool.totalAmount.sub(amount);
        emit EmergencyWithdraw(_user, _pid, amount);
    }

    // Safe LBP transfer function, just in case if rounding error causes pool to not have enough
LBPs.
    function safeLbpTransfer(address _to, uint256 _amount) internal {
        uint256 lbpBal = lbp.balanceOf(address(this));
        if (_amount > lbpBal) {
            lbp.transfer(_to, lbpBal);
        } else {
```

```
                lbp.transfer(_to, _amount);
            }
        }

        modifier notPause() {
            require(paused == false, 'Mining has been suspended');
            _;
        }
}

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

import "@openzeppelin/contracts/access/Ownable.sol";
import '@openzeppelin/contracts/token/ERC20/ERC20.sol';
import "@openzeppelin/contracts/utils/EnumerableSet.sol";

contract LbpToken is ERC20, Ownable{

    EnumerableSet.AddressSet private _minters;

    uint256 private constant maxSupply = 1000000000 * 1e18;          // the total supply

    /**
     * @notice Constructs the HERC-20 contract.
     */
    constructor() public ERC20('LBP BSC', 'lbp') {}
    // knownsec //  添加 minter
    function addMinter(address _addMinter) public onlyOwner returns (bool) {
        require(_addMinter != address(0), "LbpToken: _addMinter is the zero address");
        return EnumerableSet.add(_minters, _addMinter);
    }
```

```
    // knownsec //  删除 minter
    function delMinter(address _delMinter) public onlyOwner returns (bool) {
        require(_delMinter != address(0), "LbpToken: _delMinter is the zero address");
        return EnumerableSet.remove(_minters, _delMinter);
    }
    // knownsec //  检查 minter
    function isMinter(address account) public view returns (bool) {
        return EnumerableSet.contains(_minters, account);
    }
    // knownsec //  铸币操作，限制铸币上限为 10 亿
    function mint(address _to, uint256 _amount) public onlyMinter returns (bool) {
        if (totalSupply().add(_amount) > maxSupply) {
            return false;
        }
        _mint(_to, _amount);
        return true;
    }
    // knownsec // minter 限制修饰器
    // modifier for mint function
    modifier onlyMinter() {
        require(isMinter(msg.sender), "caller is not the minter");
        _;
    }
}

// knownsec // SwapMining
// 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/utils/EnumerableSet.sol';
```

```
import './interfaces/ILongswapFactory.sol';
import './interfaces/ILongswapPair.sol';
import './interfaces/IOracle.sol';
import './interfaces/ILbp.sol';

contract SwapMining is Ownable {

    using SafeMath for uint256;
    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _whitelist;

    IOracle public oracle;
    // LONG tokens created per block
    uint256 public lbpPerBlock;
    // The block number when LONG mining starts.
    uint256 public startBlock;
    // How many blocks are halved
    uint256 public halvingPeriod = 1670400;
    // Total allocation points
    uint256 public totalAllocPoint = 0;
    // router address
    address public router;
    // factory address
    ILongswapFactory public factory;
    // lbp token address
    ILbp public lbp;
    // Calculate price based on USDT
    address public targetToken;
    // pair corresponding pid
    mapping(address => uint256) public pairOfPid;

    constructor(
```

```
        ILbp _lbp,
        ILongswapFactory _factory,
        IOracle _oracle,
        address _router,
        address _targetToken,
        uint256 _lbpPerBlock,
        uint256 _startBlock
    ) public {
        lbp = _lbp;
        factory = _factory;
        oracle = _oracle;
        router = _router;
        targetToken = _targetToken;
        lbpPerBlock = _lbpPerBlock;
        startBlock = _startBlock;
    }

    struct UserInfo {
        uint256 quantity;       // How many LP tokens the user has provided
        uint256 blockNumber;    // Last transaction block
    }

    struct PoolInfo {
        address pair;           // Trading pairs that can be mined
        uint256 quantity;       // Current amount of LPs
        uint256 totalQuantity;  // All quantity
        uint256 allocPoint;     // How many allocation points assigned to this pool
        uint256 allocLbpAmount; // How many LBPs
        uint256 lastRewardBlock;// Last transaction block
    }

    PoolInfo[] public poolInfo;
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;
```

```
modifier onlyRouter() {
        require(msg.sender == router, "SwapMining: caller is not the router");
        _;
}

function poolLength() public view returns (uint256) {
        return poolInfo.length;
}

function addPair(uint256 _allocPoint, address _pair, bool _withUpdate) public onlyOwner {
        require(_pair != address(0), "_pair is the zero address");
        if (_withUpdate) {
                massMintPools();
        }
        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
        pair : _pair,
        quantity : 0,
        totalQuantity : 0,
        allocPoint : _allocPoint,
        allocLbpAmount : 0,
        lastRewardBlock : lastRewardBlock
        }));
        pairOfPid[_pair] = poolLength() - 1;
}

// Update the allocPoint of the pool
function setPair(uint256 _pid, uint256 _allocPoint, bool _withUpdate) public onlyOwner {
        if (_withUpdate) {
                massMintPools();
        }
```

```
        totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
        poolInfo[_pid].allocPoint = _allocPoint;
    }

    // Set the number of lbp produced by each block
    function setLbpPerBlock(uint256 _newPerBlock) public onlyOwner {
        massMintPools();
        lbpPerBlock = _newPerBlock;
    }

    // Only tokens in the whitelist can be mined LBP
    function addWhitelist(address _addToken) public onlyOwner returns (bool) {
        require(_addToken != address(0), "SwapMining: token is the zero address");
        return EnumerableSet.add(_whitelist, _addToken);
    }

    function delWhitelist(address _delToken) public onlyOwner returns (bool) {
        require(_delToken != address(0), "SwapMining: token is the zero address");
        return EnumerableSet.remove(_whitelist, _delToken);
    }

    function getWhitelistLength() public view returns (uint256) {
        return EnumerableSet.length(_whitelist);
    }

    function isWhitelist(address _token) public view returns (bool) {
        return EnumerableSet.contains(_whitelist, _token);
    }

    function getWhitelist(uint256 _index) public view returns (address){
        require(_index <= getWhitelistLength() - 1, "SwapMining: index out of bounds");
        return EnumerableSet.at(_whitelist, _index);
    }
```

```
function setHalvingPeriod(uint256 _block) public onlyOwner {
    halvingPeriod = _block;
}

function setRouter(address newRouter) public onlyOwner {
    require(newRouter != address(0), "SwapMining: new router is the zero address");
    router = newRouter;
}

function setOracle(IOracle _oracle) public onlyOwner {
    require(address(_oracle) != address(0), "SwapMining: new oracle is the zero address");
    oracle = _oracle;
}

// At what phase
function phase(uint256 blockNumber) public view returns (uint256) {
    if (halvingPeriod == 0) {
        return 0;
    }
    if (blockNumber > startBlock) {
        return (blockNumber.sub(startBlock).sub(1)).div(halvingPeriod);
    }
    return 0;
}

function phase() public view returns (uint256) {
    return phase(block.number);
}

function reward(uint256 blockNumber) public view returns (uint256) {
    uint256 _phase = phase(blockNumber);
    return lbpPerBlock.div(2 ** _phase);
```

```
    }

    function reward() public view returns (uint256) {
        return reward(block.number);
    }

    // Rewards for the current block
    function getLbpReward(uint256 _lastRewardBlock) public view returns (uint256) {
        require(_lastRewardBlock <= block.number, "SwapMining: must little than the current block number");
        uint256 blockReward = 0;
        uint256 n = phase(_lastRewardBlock);
        uint256 m = phase(block.number);
        // If it crosses the cycle
        while (n < m) {
            n++;
            // Get the last block of the previous cycle
            uint256 r = n.mul(halvingPeriod).add(startBlock);
            // Get rewards from previous periods
            blockReward = blockReward.add((r.sub(_lastRewardBlock)).mul(reward(r)));
            _lastRewardBlock = r;
        }
        blockReward = blockReward.add((block.number.sub(_lastRewardBlock)).mul(reward(block.number)));
        return blockReward;
    }

    // Update all pools Called when updating allocPoint and setting new blocks
    function massMintPools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            mint(pid);
        }
```

```
    }

    function mint(uint256 _pid) public returns (bool) {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return false;
        }
        uint256 blockReward = getLbpReward(pool.lastRewardBlock);
        if (blockReward <= 0) {
            return false;
        }
        // Calculate the rewards obtained by the pool based on the allocPoint
        uint256 lbpReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
        lbp.mint(address(this), lbpReward);
        // Increase the number of tokens in the current pool
        pool.allocLbpAmount = pool.allocLbpAmount.add(lbpReward);
        pool.lastRewardBlock = block.number;
        return true;
    }

    // swapMining only router
    function swap(address account, address input, address output, uint256 amount) public
onlyRouter returns (bool) {
        require(account != address(0), "SwapMining: taker swap account is the zero address");
        require(input != address(0), "SwapMining: taker swap input is the zero address");
        require(output != address(0), "SwapMining: taker swap output is the zero address");

        if (poolLength() <= 0) {
            return false;
        }

        if (!isWhitelist(input) || !isWhitelist(output)) {
            return false;
```

```
        }

        address pair = ILongswapFactory(factory).pairFor(input, output);

        PoolInfo storage pool = poolInfo[pairOfPid[pair]];
        // If it does not exist or the allocPoint is 0 then return
        if (pool.pair != pair || pool.allocPoint <= 0) {
            return false;
        }

        uint256 quantity = getQuantity(output, amount, targetToken);
        if (quantity <= 0) {
            return false;
        }

        mint(pairOfPid[pair]);

        pool.quantity = pool.quantity.add(quantity);
        pool.totalQuantity = pool.totalQuantity.add(quantity);
        UserInfo storage user = userInfo[pairOfPid[pair]][account];
        user.quantity = user.quantity.add(quantity);
        user.blockNumber = block.number;
        return true;
    }

    // The user withdraws all the transaction rewards of the pool
    function takerWithdraw() public {
        uint256 userSub;
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            PoolInfo storage pool = poolInfo[pid];
            UserInfo storage user = userInfo[pid][msg.sender];
            if (user.quantity > 0) {
```

```
                mint(pid);
                // The reward held by the user in this pool
                uint256 userReward =
pool.allocLbpAmount.mul(user.quantity).div(pool.quantity);
                pool.quantity = pool.quantity.sub(user.quantity);
                pool.allocLbpAmount = pool.allocLbpAmount.sub(userReward);
                user.quantity = 0;
                user.blockNumber = block.number;
                userSub = userSub.add(userReward);
            }
        }
        if (userSub <= 0) {
            return;
        }
        lbp.transfer(msg.sender, userSub);
    }

    // Get rewards from users in the current pool
    function getUserReward(uint256 _pid) public view returns (uint256, uint256){
        require(_pid <= poolInfo.length - 1, "SwapMining: Not find this pool");
        uint256 userSub;
        PoolInfo memory pool = poolInfo[_pid];
        UserInfo memory user = userInfo[_pid][msg.sender];
        if (user.quantity > 0) {
            uint256 blockReward = getLbpReward(pool.lastRewardBlock);
            uint256 lbpReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
            userSub =
userSub.add((pool.allocLbpAmount.add(lbpReward)).mul(user.quantity).div(pool.quantity));
        }
        //LBP available to users, User transaction amount
        return (userSub, user.quantity);
    }
```

```
    // Get details of the pool
    function getPoolInfo(uint256 _pid) public view returns (address, address, uint256, uint256, uint256, uint256){
        require(_pid <= poolInfo.length - 1, "SwapMining: Not find this pool");
        PoolInfo memory pool = poolInfo[_pid];
        address token0 = ILongswapPair(pool.pair).token0();
        address token1 = ILongswapPair(pool.pair).token1();
        uint256 lbpAmount = pool.allocLbpAmount;
        uint256 blockReward = getLbpReward(pool.lastRewardBlock);
        uint256 lbpReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
        lbpAmount = lbpAmount.add(lbpReward);
        //token0,token1,Pool remaining reward,Total /Current transaction volume of the pool
        return (token0, token1, lbpAmount, pool.totalQuantity, pool.quantity, pool.allocPoint);
    }

    function getQuantity(address outputToken, uint256 outputAmount, address anchorToken) public view returns (uint256) {
        uint256 quantity = 0;
        if (outputToken == anchorToken) {
            quantity = outputAmount;
        } else if (ILongswapFactory(factory).getPair(outputToken, anchorToken) != address(0)) {
            quantity = IOracle(oracle).consult(outputToken, outputAmount, anchorToken);
        } else {
            uint256 length = getWhitelistLength();
            for (uint256 index = 0; index < length; index++) {
                address intermediate = getWhitelist(index);
                if (ILongswapFactory(factory).getPair(outputToken, intermediate) != address(0) && ILongswapFactory(factory).getPair(intermediate, anchorToken) != address(0)) {
                    uint256 interQuantity = IOracle(oracle).consult(outputToken, outputAmount, intermediate);
                    quantity = IOracle(oracle).consult(intermediate, interQuantity, anchorToken);
```

```
                    break;
                }
            }
        }
        return quantity;
    }
}
```

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 BNB，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 BNB 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 BNB 的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送 BNB 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 A：合约代码

本次测试代码来源：

6. 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。

7. 附录 C：智能合约安全审计工具简介

7.1 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。

7.2 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。

7.3 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。

7.4 Echidna

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

7.5 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。

7.6 ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

7.7 ida-evm

ida-evm 是一个针对以太坊虚拟机（EVM）的 IDA 处理器模块。

7.8 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。

7.9 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