知道创宇

密级 ★ | 项目组公开

智能合约审计报告

安全状态

安全

★ ★ ★ ★ ★

主测人： 知道创宇区块链安全研究团队

版本说明

修订内容	时间	修订者	版本号
编写文档	20201204	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	文档编号	保密级别
Abelo 智能合约审计报告	V1.0	Abelo-ZNHY-20201204	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 2020 年 12 月 1 日开始到 2020 年 12 月 4 日结束，在此期间针对 Abelo **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。

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、四章节）进行了全面的分析，综合评定为通过。

本次智能合约安全审计结果： 通过

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

本次测试的目标信息：

条目	描述
Token 名称	Abelo（foxdex（fox））
代码类型	代币代码、DeFi 协议代码、波场智能合约
代码语言	Solidity

合约文件及哈希：

合约文件	MD5
BFactory.sol	ad93e21b89d1c080790c864e5b15f7b6
BMath.sol	14487c85cdeb0219667bd0e8d87f1750
BNum.sol	00a657f84cef5a2aee36479fc9013c70
BParam.sol	d977daadea464077b2a0179f1d0586b8
BPool.sol	0b83759bd511c8608c0a690aa87d8450
FactoryManager.sol	82edd0e888814a867de0466847edd831

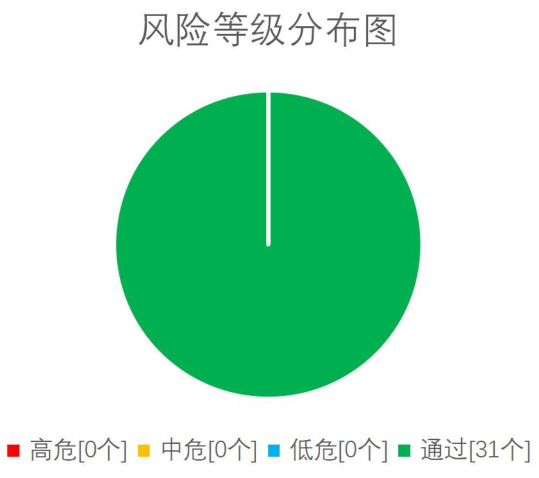
Migrations.sol	ca8d6ca8a6edf34f149a5095a8b074c9
PoolView.sol	a746d9d9d69ccab80b03e4920b9f1a69
TokenBase.sol	667947567e28ce5cb82bd122dc5e5d26
MockMath.sol	9221462a557e946609490118ad24b482
MockToken.sol	64dd491f604f76cae9361a64c3288832
FoxToken.sol	2b6efc0634811c72bc00dfe5cf48f5c9
MasterChef.sol	ecd903b9e4b688b87331f2b7a74581f5

2. 代码漏洞分析

2.1 漏洞等级分布

本次漏洞风险按等级统计:

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	0	31



2.2 审计结果汇总说明

审计结果			
审计项目	审计内容	状态	描述
业务安全性检测	代币功能	通过	经检测，不存在该安全问题。
	MasterChef 合约	通过	经检测，不存在该安全问题。
	资金池相关功能	通过	经检测，不存在该安全问题。
	FactoryManager 合约	通过	经检测，不存在该安全问题。
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。
	energy 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 代币功能【通过】

审计分析：经代币功能在 FoxToken.sol 合约文件中实现，符合标准。

```
contract FoxToken is ERC20, ERC20Detailed, Ownable {
    uint256 public MAX = 31 * 1e6 * 1e18;
    constructor () public ERC20Detailed("foxdex", "fox", 18) {
    }

    function mint(address _to) public onlyOwner {//knownsec// 仅代币总量为 0 时 owner 可调用
        require(totalSupply() == 0, "mint one");
        _mint(_to, MAX);
    }

}
```

安全建议：无。

3.2 MasterChef 合约【通过】

审计分析：MasterChef 合约基于 SushiSwap 的 MasterChef.sol 合约文件，增删一小部分功能，包括删去 IMigratorChef、新增一些奖励分成机制，以及新增一些用于 controller 地址提取奖励、设置分成地址成员和比率的功能函数等。

```
contract MasterChef is Ownable {
    using SafeMath for uint256;
    using EnumerableSet for EnumerableSet.AddressSet;

    // Info of each user.
    struct UserInfo {
        uint256 amount; // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt. See explanation below.
        //
        // We do some fancy math here. Basically, any point in time, the amount of dex
        // entitled to a user but is pending to be distributed is:
        //
        //     pending reward = (user.amount * pool.accPicklePerShare) - user.rewardDebt
        //
        // Whenever a user deposits or withdraws LP tokens to a pool. Here's what happens:
        //     1. The pool's `accPicklePerShare` (and `lastRewardBlock`) gets updated.
        //     2. User receives the pending reward sent to his/her address.
        //     3. User's `amount` gets updated.
        //     4. User's `rewardDebt` gets updated.
    }

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

    // 7 days
    uint256 public duration = 7 * 28800;//knownsec// 周期 7 天

    FoxToken public foxToken;
    // Dev fund (25%, initially)
    uint256 public devFundDivRate = 4;//knownsec// 开发者奖励分成比率,1/4=25%
    // gover fund (5%, initially)
    uint256 public goverFundDivRate = 20;//knownsec// 治理奖励分成比率,1/20=5%
    // insurance fund (5%, initially)
    uint256 public insuranceFundDivRate = 20;//knownsec// 保险奖励分成比率,1/20=5%
    uint256 public tokenPerBlock = 3 * 1e18;//knownsec// 区块奖励 token 量
    // tokens created per block.
    uint256 public tokenPerBlockForReward;//knownsec// 扣除各费后用于分发的区块奖励
token 量

    address public controller;
    EnumerableSet.AddressSet private devaddrs;

    address public governaddr;
    address public insuranceaddr;

    uint256 public devFund;
    uint256 public goverFund;
    uint256 public insuranceFund;
    ......
    constructor(
        FoxToken _fox,
        uint256 _startBlock,
        uint256 _duration
    ) public {
        foxToken = _fox;
        startBlock = _startBlock;
        duration = _duration;
        periodEndBlock = _startBlock.add(duration);
        tokenPerBlockForReward = calTokenPerBlock(tokenPerBlock);
        controller = msg.sender;//knownsec// 合约部署者为控制器地址
    }
    function calTokenPerBlock(uint256 _blockToken) view internal returns
(uint256){//knownsec// 计算扣除各费后的区块奖励
        uint256 _devFund = _blockToken.div(devFundDivRate);//knownsec// 25%
        uint256 _goverFund = _blockToken.div(goverFundDivRate);//knownsec// 5%
        uint256 _insuranceFund = _blockToken.div(insuranceFundDivRate);//knownsec// 5%

        return _blockToken.sub(_devFund).sub(_goverFund).sub(_insuranceFund);
    }
    ......
    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 = block.number - pool.lastRewardBlock;
        uint256 foxReward = multiplier
        .mul(calPerBlockToken(pool.perBlockToken))
        .mul(pool.allocPoint)
        .div(totalAllocPoint);
        pool.accFoxPerShare = pool.accFoxPerShare.add(
            foxReward.mul(1e12).div(lpSupply)
        );
        pool.lastRewardBlock = block.number;
        if (block.number >= periodEndBlock) {//knownsec//  若周期结束
            // Calculate the fund of the period
            uint256 durationReward = tokenPerBlock.mul(duration);

            uint256 _devFund = durationReward.div(devFundDivRate);
            uint256 _goverFund = durationReward.div(goverFundDivRate);
            uint256 _insuranceFund = durationReward.div(insuranceFundDivRate);
```

```
                devFund = devFund.add(_devFund);
                goverFund = goverFund.add(_goverFund);
                insuranceFund = insuranceFund.add(_insuranceFund);

                tokenPerBlock = tokenPerBlock.mul(98).div(100);//knownsec// 下一周期区块奖励调为 98%
                tokenPerBlockForReward = calTokenPerBlock(tokenPerBlock);

                pool.perBlockToken = tokenPerBlockForReward;
                periodEndBlock = block.number.add(duration);//knownsec// 开启下一周期
            }
        }
        ......
        function claimDevFund() public {//knownsec// 提取开发者奖励,仅 controller 调用
            require(msg.sender == controller, "!controller");
            uint256 perDevFund = devFund.div(devaddrs.length());
            for (uint256 i = 0; i < devaddrs.length() - 1; i++) {
                foxToken.transfer(devaddrs.get(i), perDevFund);
            }
            uint256 remainFund = devFund.sub(perDevFund.mul(devaddrs.length() - 1));
            foxToken.transfer(devaddrs.get(devaddrs.length() - 1), remainFund);
            devFund = 0;
        }

        function addDev(address _devaddr) public {//knownsec// 添加开发者地址,仅 controller 调用
            require(msg.sender == controller, "!controller");
            require(devaddrs.length() < 100, "less 100");
            devaddrs.add(_devaddr);
        }

        function removeDev(address _devaddr) public {//knownsec// 移除开发者地址,仅 controller 调用
            require(msg.sender == controller, "!controller");
            devaddrs.remove(_devaddr);
        }

        function contains(address _dev) public view returns (bool){
            return devaddrs.contains(_dev);
        }

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

        function setDevFundDivRate(uint256 _devFundDivRate) public {//knownsec// 设置开发者奖励分成比率,仅 controller 调用
            require(msg.sender == controller, "!controller");
            require(_devFundDivRate >= 3, "!devFundDivRate-0");
            devFundDivRate = _devFundDivRate;
        }

        function insurance(address _insuranceaddr) public {//knownsec// 设置保险地址,仅 controller 调用
            require(msg.sender == controller, "!controller");
            insuranceaddr = _insuranceaddr;
        }

        function setInsuranceFundDivRate(uint256 _insuranceFundDivRate) public {//knownsec// 设置保险奖励分成比率,仅 controller 调用
            require(msg.sender == controller, "!controller");
            require(_insuranceFundDivRate >= 15, "!devFundDivRate-0");
            insuranceFundDivRate = _insuranceFundDivRate;
        }

        function gover(address _goveraddr) public {//knownsec// 设置 gover 地址,仅 controller 调用
```

```
        require(msg.sender == controller, "!controller");
        governaddr = _goveraddr;
    }

    function setGoverFundDivRate(uint256 _goverFundDivRate) public {//knownsec// 设置治理奖励分成比率,仅 controller 调用
        require(msg.sender == controller, "!controller");
        require(_goverFundDivRate >= 15, "!devFundDivRate-0");
        goverFundDivRate = _goverFundDivRate;
    }
    ......
}
```

安全建议：无。

3.3 资金池相关功能【通过】

审计分析：资金池功能在 BPool.sol 合约文件中实现，资金池相关的功能 BPool、BFactory、TokenBase 等均是在 Balancer-core 项目部分核心合约的基础上做小部分增删实现的，如增加细化一些交易手续费分成等。

```
// BPool.sol
contract BPool is BToken, BMath {
    ......
    function _chargeFee(address _token, uint _amount) internal {
        uint foxFund = bmul(_amount, IFactoryManager(factoryManager).swapFeeForDex());//knownsec// 交易所手续费分成
        tokenFundForFox[_token] = badd(tokenFundForFox[_token], foxFund);
        uint buildFund = bmul(_amount, _swapFeeForBuilder);//knownsec// 创建者手续费分成
        tokenFundForBuilder[_token] = badd(tokenFundForBuilder[_token], buildFund);
    }
    ......
    function claimFactoryFund() public {//knownsec// 提取工厂合约管理员收益,仅工厂合约管理员调用
        require(msg.sender == factoryManager, "!factory manager");
        for (uint i = 0; i < _tokens.length; i++) {
            address token = _tokens[i];
            uint amount = tokenFundForFox[token];
            if (amount > 0) {
                IERC20(token).transfer(factoryManager, amount);
                tokenFundForFox[token] = 0;
            }
        }


    }

    function clainBuildFund() public {//knownsec// 提取创建者收益,仅 controller 调用
        require(msg.sender == _controller, "!controller");
        for (uint i = 0; i < _tokens.length; i++) {
            address token = _tokens[i];
            uint amount = tokenFundForBuilder[token];
            if (amount > 0) {
                IERC20(token).transfer(_controller, amount);
                tokenFundForBuilder[token] = 0;
            }
        }
    }
}

// BFactory.sol
contract BFactory {
    ......
    function newBPool() external returns (BPool){//knownsec// 创建新池
        BPool bpool = new BPool(factoryManager);
        _isBPool[address(bpool)] = true;
        emit LOG_NEW_POOL(msg.sender, address(bpool));
        bpool.setController(msg.sender);
        allPools.push(bpool);
        return bpool;
    }

    constructor(address _factoryManager) public {
        controller = msg.sender;
        factoryManager = _factoryManager;
    }

    function setController(address _controller) external {//knownsec// 更改 controller 地址，仅
controller 调用
        require(msg.sender == controller, "!controller");
        controller = _controller;
    }
}
```

安全建议：无。

3.4 FactoryManager 合约【通过】

审计分析：FactoryManager 工厂合约管理员在 FactoryManager.sol 合约文件中实现，作用于 BFactory 和 controller 之间，是 Balancer-core 项目中没有的新增功能合约。主要用于 controller 地址添加交易池、提取收益、设置交易手续费等，以及对资金池流动性的调整。

```
contract FactoryManager is BParam, BMath, Ownable {

    uint256 public goverFundDivRate = 3 * CENTI_FEE;
    address public governaddr;

    uint256 public burnRate = 15 * CENTI_FEE;

    uint public   exitFee = 0;
    uint public swapFeeForDex = 0;
    address public controller;
    mapping(address => address) swapPools;

    address public dexToken;
    address public baseToken;
    address public basePool;

    address public factory;

    uint256 private setFactoryManagerCount = 0;
    uint256 private setBasePoolCount = 0;

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

    function setController(address _controller) onlyOwner external {//knownsec// 设置 controller 地址,仅 controller 调用
        controller = _controller;
    }

    function setFactory(address _factory) external {//knownsec// 设置 facotry 地址,最多调用 2 次,仅 controller 调用
        require(msg.sender == controller, "!controller");
        require(setFactoryManagerCount < 2, "limit factory address");
        factory = _factory;
        setFactoryManagerCount = badd(setFactoryManagerCount, 1);
    }

    function getSwapPool(address token) public view returns (address){//knownsec// 指定 token 查找交易池地址
        return swapPools[token];
    }

    function addSwapPools(address token, address pool) public {//knownsec// 添加交易池,仅 controller 调用
        require(msg.sender == controller, "!controller");
        require(factory != address(0), "factory=0");
        require(BFactory(factory).isBPool(pool), "!pool");
        swapPools[token] = pool;

    }

    function removeSwapPools(address token) public {//knownsec// 移除交易池,仅 controller 调用
        require(msg.sender == controller, "!controller");
        swapPools[token] = address(0);

    }

    function setExitFee(uint _exitFee) public {
        require(msg.sender == controller, "!controller");
        require(_exitFee <= CENTI_FEE, "ERR_MAX_EXIT_FEE");
        exitFee = _exitFee;
    }

    function setSwapFeeForDex(uint _fee) public {//knownsec// 设置交易所的交易手续费分成,仅 controller 调用
        require(msg.sender == controller, "!controller");
        require(_fee <= CENTI_FEE, "ERR_MAX_EXIT_FEE");
        swapFeeForDex = _fee;
    }

    function claimToken(address token) public {//knownsec// 将指定 token 全部转换为 baseToken,仅 controller 调用
        require(msg.sender == controller, "!controller");
        address _pool = swapPools[token];
        require(_pool != address(0), "not set pool");
        require(baseToken != address(0), "not set base token");
        swapToken(_pool, token, baseToken);
    }


    function swapToken(//knownsec// 指定交易池将tokenIn 转换为tokenOut,仅 controller 调用
        address poolAddress, address tokenIn, address tokenOut
    ) public returns (uint){
        require(msg.sender == controller, "!controller");
        require(factory != address(0), "factory=0");
        require(BFactory(factory).isBPool(poolAddress), "!pool");
        BPool pool = BPool(poolAddress);
        uint _tokenInAmount = IERC20(tokenIn).balanceOf(address(this));
        IERC20(tokenIn).approve(poolAddress, _tokenInAmount);
        (uint tokenAmountOut,) = pool.swapExactAmountIn(tokenIn, _tokenInAmount, tokenOut, 0, MAX);
        return tokenAmountOut;
    }

    function addFoxPoolLiquidity() external {//knownsec// baseToken 扣除管理费 3%, 销毁 15%(转为 dexToken),剩余加入池中,仅 controller 调用
        require(msg.sender == controller, "!controller");
        uint _amount = IERC20(baseToken).balanceOf(address(this));
        uint256 _goverFund = bmul(_amount, goverFundDivRate);
        IERC20(baseToken).transfer(governaddr, _goverFund);
        uint _tokenInAmount = bmul(_amount, burnRate);

        IERC20(baseToken).approve(basePool, _tokenInAmount);
        BPool(basePool).swapExactAmountIn(baseToken, _tokenInAmount, dexToken, 0, MAX);

        _amount = bsub(_amount, _goverFund);
        _amount = bsub(_amount, _tokenInAmount);

        IERC20(baseToken).approve(basePool, _amount);
        BPool(basePool).joinswapExternAmountIn(baseToken, _amount, 0);
    }

    function collect(BPool pool) public {//knownsec// 提取指定池收益
        require(msg.sender == controller, "!controller");
        pool.claimFactoryFund();
        removeLiquidity(pool);

    }

    function removeLiquidity(BPool pool) public {//knownsec// 移除指定池的所有流动性
        require(msg.sender == controller, "!controller");
        uint _amount = IERC20(pool).balanceOf(address(this));
        if (_amount > 0) {
            uint[] memory amountOuts = new uint[](pool.getNumTokens());
            pool.exitPool(_amount, amountOuts);
        }
    }

    function burnToken() public {//knownsec// 销毁本合约所有 dexToken,无权限校验
        uint _amount = IERC20(dexToken).balanceOf(address(this));
        if (_amount > 0) {
            IERC20(dexToken).transfer(address(0), _amount);
        }
    }



    function setBasePoolToken(address _pool, address _dexToken, address _baseToken) public {
        require(msg.sender == controller, "!controller");
        require(factory != address(0), "factory is 0");
        require(BFactory(factory).isBPool(_pool), "!pool");
        require(BPool(_pool).isBound(_dexToken), "not bound");
        require(BPool(_pool).isBound(_baseToken), "not bound");
        require(setBasePoolCount < 2, "limit set base pool");
        setBasePoolCount = badd(setBasePoolCount, 1);
        basePool = _pool;
        dexToken = _dexToken;
        baseToken = _baseToken;

    }

    function gover(address _goveraddr) public {
        require(msg.sender == controller, "!controller");
        governaddr = _goveraddr;
    }

    function setBurnRate(uint _rate) public {
        require(msg.sender == controller, "!controller");
        require(_rate < 50 * CENTI_FEE, "< MAX_FEE");
        burnRate = _rate;
    }

    function setGoverFundDivRate(uint256 _goverFundDivRate) public {
        require(msg.sender == controller, "!controller");
        require(_goverFundDivRate < MAX_FEE, "< MAX_FEE");
        goverFundDivRate = _goverFundDivRate;
    }


    function claimOtherToken(address _pool, address _token) external {
        require(msg.sender == controller, "!controller");
        BPool(_pool).withdrawToken(_token);


    }


}
```

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 0.5.15 以上，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. energy 消耗检测【通过】

检查 energy 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，智能合约未使用 call 函数，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，智能合约代码中存在增发代币的功能，但仅在代币总量为 0 时初始化使用，故通过。

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，智能合约代码不使用结构体，不存在该问题。

安全建议：无。

4.20. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送 Ether，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300energy 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300energy 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 energy 进行调用（可通过传入 energy_value 参数进行限制），不能有效防止重入攻击。

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.21. 交易顺序依赖【通过】

由于矿工总是通过代表外部拥有地址（EOA）的代码获取 energy 费用，因此用户可以指定更高的费用以便更快地开展交易。由于波场区块链是公开的，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】

在波场的世界中，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 energy 导致 energy 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智能合约漏洞。

在 Solidity 中，调用其他地址的函数或者给合约地址转账，都会把自己的

地址作为被调合约的 msg.sender 传递过去。此时，如果传递的 energy 足够的话，就可能会被对方进行重入攻击，在波场中对合约地址调用 transfer/sender，只会传递 2300 的 Energy，这不足以发起一次重入攻击。所以要一定避免通过 call.value 的方式，调用未知的合约地址。因为 call.value 可以传入远大于 2300 的 energy。

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，智能合约未使用 call 函数，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：

BFactory.sol

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.

// This program is disstributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.   See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program.   If not, see <http://www.gnu.org/licenses/>.

pragma solidity ^0.5.9;

// Builds new BPools, logging their addresses and providing `isBPool(address) -> (bool)`

import "./BPool.sol";
import "./PoolView.sol";

contract BFactory {

        event LOG_NEW_POOL(
                address indexed caller,
                address indexed pool
        );

        BPool[] public allPools;
        mapping(address => bool) private _isBPool;
        address public controller;
        address public factoryManager;

        function isBPool(address b) external view returns (bool){
                return _isBPool[b];
        }

        function newBPool() external returns (BPool){//knownsec//  创建新池
                BPool bpool = new BPool(factoryManager);
                _isBPool[address(bpool)] = true;
                emit LOG_NEW_POOL(msg.sender, address(bpool));
                bpool.setController(msg.sender);
                allPools.push(bpool);
                return bpool;
        }


        constructor(address _factoryManager) public {
                controller = msg.sender;
                factoryManager = _factoryManager;
        }


        function setController(address _controller) external {//knownsec//  更改 controller 地址,仅 controller 调用
                require(msg.sender == controller, "!controller");
                controller = _controller;
        }



}
```

BMath.sol

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.   See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program.   If not, see <http://www.gnu.org/licenses/>.
```

```
pragma solidity ^0.5.9;

import "./BNum.sol";

contract BMath is BParam, BNum {

/**********************************************************************************************
***
    //                                                                              calcSpotPrice
//
    //                          sP                                =                     spotPrice
//
    // bI = tokenBalanceIn                 ( bI / wI )         1                                  //
    // bO = tokenBalanceOut         sP =  -----------  *  ----------                             //
    // wI = tokenWeightIn                  ( bO / wO )     ( 1 - sF )                            //
    //                         wO                              =                       tokenWeightOut
//
    //                          sF                                =                       swapFee
//

**********************************************************************************************
**/
    function calcSpotPrice(
        uint tokenBalanceIn,
        uint tokenWeightIn,
        uint tokenBalanceOut,
        uint tokenWeightOut,
        uint swapFee
    ) public pure returns (uint spotPrice){
        uint numer = bdiv(tokenBalanceIn, tokenWeightIn);
        uint denom = bdiv(tokenBalanceOut, tokenWeightOut);
        uint ratio = bdiv(numer, denom);
        uint scale = bdiv(BONE, bsub(BONE, swapFee));
        return (spotPrice = bmul(ratio, scale));
    }

/**********************************************************************************************
***
    //                                                                              calcOutGivenIn
//
    //                         aO                              =                    tokenAmountOut
//
    //                         bO                              =                   tokenBalanceOut
//
    // bI = tokenBalanceIn              /      /            bI              \    (wI / wO) \      //
    // aI = tokenAmountIn       aO = bO * |  1 - | --------------------------  | ^          |     //
    // wI = tokenWeightIn               \      \ ( bI + ( aI * ( 1 - sF )) /              /      //
    //                         wO                              =                    tokenWeightOut
//
    //                          sF                                =                       swapFee
//

**********************************************************************************************
**/
    function calcOutGivenIn(
        uint tokenBalanceIn,
        uint tokenWeightIn,
        uint tokenBalanceOut,
        uint tokenWeightOut,
        uint tokenAmountIn,
        uint swapFee
    ) public pure returns (uint tokenAmountOut){
        uint weightRatio = bdiv(tokenWeightIn, tokenWeightOut);
        uint adjustedIn = bsub(BONE, swapFee);
        adjustedIn = bmul(tokenAmountIn, adjustedIn);
        uint y = bdiv(tokenBalanceIn, badd(tokenBalanceIn, adjustedIn));
        uint foo = bpow(y, weightRatio);
        uint bar = bsub(BONE, foo);
        tokenAmountOut = bmul(tokenBalanceOut, bar);
        return tokenAmountOut;
    }

/**********************************************************************************************
***
    //                                                                              calcInGivenOut
//
    //                          aI                              =                    tokenAmountIn
//
    // bO = tokenBalanceOut                          /   /         bO          \     (wO / wI)       \
//
    // bI = tokenBalanceIn            bI * |  | ------------   | ^                - 1    |          //
    // aO = tokenAmountOut       aI =        \   \ ( bO - aO ) /                        /           //
    // wI = tokenWeightIn              --------------------------------------------             //
    //  wO  =  tokenWeightOut                                                          ( 1  -  sF  )
```

```
//
    //                          sF                           =                          swapFee
//
*******************************************************************************************
**/
    function calcInGivenOut(
        uint tokenBalanceIn,
        uint tokenWeightIn,
        uint tokenBalanceOut,
        uint tokenWeightOut,
        uint tokenAmountOut,
        uint swapFee
    ) public pure returns (uint tokenAmountIn){
        uint weightRatio = bdiv(tokenWeightOut, tokenWeightIn);
        uint diff = bsub(tokenBalanceOut, tokenAmountOut);
        uint y = bdiv(tokenBalanceOut, diff);
        uint foo = bpow(y, weightRatio);
        foo = bsub(foo, BONE);
        tokenAmountIn = bsub(BONE, swapFee);
        tokenAmountIn = bdiv(bmul(tokenBalanceIn, foo), tokenAmountIn);
        return tokenAmountIn;
    }

/******************************************************************************************
***
    //                                                                      calcPoolOutGivenSingleIn
//
    // pAo = poolAmountOut              /                                                      \
//
    // tAi = tokenAmountIn          ///        /      //     wI \          \\        \      wI \        //
    // wI = tokenWeightIn          //| tAi *| 1 - || 1 - --    | * sF || + tBi \     --   \             //
    // tW = totalWeight     pAo=||   \       \       \\    tW /        //            | ^ tW    | * pS - pS //
    // tBi = tokenBalanceIn     \\   ------------------------------------- /        /                //
    // pS = poolSupply                \\                                  tBi                     /       /
//
    // sF = swapFee                    \                                                                /
//
*******************************************************************************************
**/
    function calcPoolOutGivenSingleIn(
        uint tokenBalanceIn,
        uint tokenWeightIn,
        uint poolSupply,
        uint totalWeight,
        uint tokenAmountIn,
        uint swapFee
    ) public pure returns (uint poolAmountOut){
        // Charge the trading fee for the proportion of tokenAi
        ///   which is implicitly traded to the other pool tokens.
        // That proportion is (1- weightTokenIn)
        // tokenAiAfterFee = tAi * (1 - (1-weightTi) * poolFee);
        uint normalizedWeight = bdiv(tokenWeightIn, totalWeight);
        uint zaz = bmul(bsub(BONE, normalizedWeight), swapFee);
        uint tokenAmountInAfterFee = bmul(tokenAmountIn, bsub(BONE, zaz));

        uint newTokenBalanceIn = badd(tokenBalanceIn, tokenAmountInAfterFee);
        uint tokenInRatio = bdiv(newTokenBalanceIn, tokenBalanceIn);

        // uint newPoolSupply = (ratioTi ^ weightTi) * poolSupply;
        uint poolRatio = bpow(tokenInRatio, normalizedWeight);
        uint newPoolSupply = bmul(poolRatio, poolSupply);
        poolAmountOut = bsub(newPoolSupply, poolSupply);
        return poolAmountOut;
    }

/******************************************************************************************
***
    //                                                                      calcSingleInGivenPoolOut
//
    // tAi = tokenAmountIn                  //(pS + pAo)\      /      1      \\                     //
    // pS = poolSupply                     || ---------   | ^ | --------- || * bI - bI            //
    // pAo = poolAmountOut                   \\      pS     /       \(wI / tW)//                    //
    // bI = balanceIn           tAi =   ------------------------------------------             //
    //  wI  =  weightIn                                                           /              wI     \
//
    // tW = totalWeight                               |   1 - ----   |   * sF                          //
    //  sF  =  swapFee                                                            \              tW     /
//
*******************************************************************************************
**/
    function calcSingleInGivenPoolOut(
        uint tokenBalanceIn,
            uint tokenWeightIn,
            uint poolSupply,
            uint totalWeight,
            uint poolAmountOut,
            uint swapFee
        ) public pure returns (uint tokenAmountIn){
            uint normalizedWeight = bdiv(tokenWeightIn, totalWeight);
            uint newPoolSupply = badd(poolSupply, poolAmountOut);
            uint poolRatio = bdiv(newPoolSupply, poolSupply);

            //uint newBalTi = poolRatio^(1/weightTi) * balTi;
            uint boo = bdiv(BONE, normalizedWeight);
            uint tokenInRatio = bpow(poolRatio, boo);
            uint newTokenBalanceIn = bmul(tokenInRatio, tokenBalanceIn);
            uint tokenAmountInAfterFee = bsub(newTokenBalanceIn, tokenBalanceIn);
            // Do reverse order of fees charged in joinswap_ExternAmountIn, this way
            //          ``` pAo == joinswap_ExternAmountIn(Ti, joinswap_PoolAmountOut(pAo, Ti)) ```
            //uint tAi = tAiAfterFee / (1 - (1-weightTi) * swapFee) ;
            uint zar = bmul(bsub(BONE, normalizedWeight), swapFee);
            tokenAmountIn = bdiv(tokenAmountInAfterFee, bsub(BONE, zar));
            return tokenAmountIn;
        }

/**********************************************************************************************
***
        //                                                                    calcSingleOutGivenPoolIn
//
        // tAo = tokenAmountOut                 /          /                                        \\
//
        // bO = tokenBalanceOut                /         // pS - (pAi * (1 - eF)) \      /     1      \           \\   //
        // pAi = poolAmountIn               | bO - || ----------------------- | ^ | --------- | * b0 || //
        // ps = poolSupply                   \          \\                pS               /        \(wO / tW)/         //   //
        // wI = tokenWeightIn          tAo =      \            \                                                    //
//
        //  tW  =  totalWeight                                         /             /                wO  \                   \
//
        // sF = swapFee                               *   | 1 - |   1 - ---- | * sF   |                                //
        //  eF  =  exitFee                                               \             \                tW  /                  /
//

**********************************************************************************************
**/
        function calcSingleOutGivenPoolIn(
            uint tokenBalanceOut,
            uint tokenWeightOut,
            uint poolSupply,
            uint totalWeight,
            uint poolAmountIn,
            uint swapFee,
            uint exitFee
        ) public view returns (uint tokenAmountOut){
            uint normalizedWeight = bdiv(tokenWeightOut, totalWeight);
            // charge exit fee on the pool token side
            // pAiAfterExitFee = pAi*(1-exitFee)
            uint poolAmountInAfterExitFee = bmul(poolAmountIn, bsub(BONE, exitFee));
            uint newPoolSupply = bsub(poolSupply, poolAmountInAfterExitFee);
            uint poolRatio = bdiv(newPoolSupply, poolSupply);

            // newBalTo = poolRatio^(1/weightTo) * balTo;
            uint tokenOutRatio = bpow(poolRatio, bdiv(BONE, normalizedWeight));
            uint newTokenBalanceOut = bmul(tokenOutRatio, tokenBalanceOut);

            uint tokenAmountOutBeforeSwapFee = bsub(tokenBalanceOut, newTokenBalanceOut);

            // charge swap fee on the output token side
            //uint tAo = tAoBeforeSwapFee * (1 - (1-weightTo) * swapFee)
            uint zaz = bmul(bsub(BONE, normalizedWeight), swapFee);
            tokenAmountOut = bmul(tokenAmountOutBeforeSwapFee, bsub(BONE, zaz));
            return tokenAmountOut;
        }

/**********************************************************************************************
***
        //                                                                    calcPoolInGivenSingleOut
//
        // pAi = poolAmountIn                  // /                      tAo                  \\       / wO \        \     //
        // bO = tokenBalanceOut               // | bO - -------------------------- ||\     | ---- |        \   //
        // tAo = tokenAmountOut          pS - ||     \       1 - ((1 - (tO / tW)) * sF)/   | ^ \ tW /   * pS | //
        // ps = poolSupply                     \\ -----------------------------------/                        /   //
        // wO = tokenWeightOut   pAi =            \\                          bO                         /                     /
//
        // tW = totalWeight              -----------------------------------------------------------   //
        //  sF  =  swapFee                                                                                    ( 1 - eF )
//
        //                              eF                                    =                                exitFee
```

```
//
*********************************************************************************************
**/
    function calcPoolInGivenSingleOut(
        uint tokenBalanceOut,
        uint tokenWeightOut,
        uint poolSupply,
        uint totalWeight,
        uint tokenAmountOut,
        uint swapFee,
        uint exitFee
    ) public view returns (uint poolAmountIn){

        // charge swap fee on the output token side
        uint normalizedWeight = bdiv(tokenWeightOut, totalWeight);
        //uint tAoBeforeSwapFee = tAo / (1 - (1-weightTo) * swapFee) ;
        uint zoo = bsub(BONE, normalizedWeight);
        uint zar = bmul(zoo, swapFee);
        uint tokenAmountOutBeforeSwapFee = bdiv(tokenAmountOut, bsub(BONE, zar));

        uint newTokenBalanceOut = bsub(tokenBalanceOut, tokenAmountOutBeforeSwapFee);
        uint tokenOutRatio = bdiv(newTokenBalanceOut, tokenBalanceOut);

        //uint newPoolSupply = (ratioTo ^ weightTo) * poolSupply;
        uint poolRatio = bpow(tokenOutRatio, normalizedWeight);
        uint newPoolSupply = bmul(poolRatio, poolSupply);
        uint poolAmountInAfterExitFee = bsub(poolSupply, newPoolSupply);

        // charge exit fee on the pool token side
        // pAi = pAiAfterExitFee/(1-exitFee)
        poolAmountIn = bdiv(poolAmountInAfterExitFee, bsub(BONE, exitFee));
        return poolAmountIn;
    }


}
```

BNum.sol

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.

pragma solidity ^0.5.9;

import "./BParam.sol";

contract BNum is BParam {

    function btoi(uint a) internal pure returns (uint){
        return a / BONE;
    }

    function bfloor(uint a) internal pure returns (uint){
        return btoi(a) * BONE;
    }

    function badd(uint a, uint b) internal pure returns (uint){
        uint c = a + b;
        require(c >= a, "ERR_ADD_OVERFLOW");
        return c;
    }

    function bsub(uint a, uint b) internal pure returns (uint){
        (uint c, bool flag) = bsubSign(a, b);
        require(!flag, "ERR_SUB_UNDERFLOW");
        return c;
    }

    function bsubSign(uint a, uint b) internal pure returns (uint, bool){
        if (a >= b) {
            return (a - b, false);
        } else {
            return (b - a, true);
        }
    }
```

```
function bmul(uint a, uint b) internal pure returns (uint){
        uint c0 = a * b;
        require(a == 0 || c0 / a == b, "ERR_MUL_OVERFLOW");
        uint c1 = c0 + (BONE / 2);
        require(c1 >= c0, "ERR_MUL_OVERFLOW");
        uint c2 = c1 / BONE;
        return c2;
}

function bdiv(uint a, uint b) internal pure returns (uint){
        require(b != 0, "ERR_DIV_ZERO");
        uint c0 = a * BONE;
        require(a == 0 || c0 / a == BONE, "ERR_DIV_INTERNAL");
        // bmul overflow
        uint c1 = c0 + (b / 2);
        require(c1 >= c0, "ERR_DIV_INTERNAL");
        //    badd require
        uint c2 = c1 / b;
        return c2;
}

// DSMath.wpow
function bpowi(uint a, uint n) internal pure returns (uint){
        uint z = n % 2 != 0 ? a : BONE;

        for (n /= 2; n != 0; n /= 2) {
                a = bmul(a, a);

                if (n % 2 != 0) {
                        z = bmul(z, a);
                }
        }
        return z;
}

// Compute b^(e.w) by splitting it into (b^e)*(b^0.w).
// Use `bpowi` for `b^e` and `bpowK` for k iterations
// of approximation of b^0.w
function bpow(uint base, uint exp) internal pure returns (uint){
        require(base >= MIN_BPOW_BASE, "ERR_BPOW_BASE_TOO_LOW");
        require(base <= MAX_BPOW_BASE, "ERR_BPOW_BASE_TOO_HIGH");

        uint whole = bfloor(exp);
        uint remain = bsub(exp, whole);

        uint wholePow = bpowi(base, btoi(whole));

        if (remain == 0) {
                return wholePow;
        }

        uint partialResult = bpowApprox(base, remain, BPOW_PRECISION);
        return bmul(wholePow, partialResult);
}

function bpowApprox(uint base, uint exp, uint precision) internal pure returns (uint){
        // term 0:
        uint a = exp;
        (uint x, bool xneg) = bsubSign(base, BONE);
        uint term = BONE;
        uint sum = term;
        bool negative = false;


        // term(k) = numer / denom
        //         = (product(a - i - 1, i=1-->k) * x^k) / (k!)
        // each iteration, multiply previous term by (a-(k-1)) * x / k
        // continue until term is less than precision
        for (uint i = 1; term >= precision; i++) {
                uint bigK = i * BONE;
                (uint c, bool cneg) = bsubSign(a, bsub(bigK, BONE));
                term = bmul(term, bmul(c, x));
                term = bdiv(term, bigK);
                if (term == 0) break;

                if (xneg) negative = !negative;
                if (cneg) negative = !negative;
                if (negative) {
                        sum = bsub(sum, term);
                } else {
                        sum = badd(sum, term);
                }
        }

        return sum;
}
```

```
}
```

BParam.sol

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.   See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program.   If not, see <http://www.gnu.org/licenses/>.

pragma solidity ^0.5.9;

contract BParam {
        uint public constant MAX = 2 ** 256 - 1;
        uint public constant BONE = 10 ** 18;

        uint public constant MIN_BOUND_TOKENS = 2;
        uint public constant MAX_BOUND_TOKENS = 2;//knownsec// 池中只能存在两种token

        //Number of tokens: pools must contain at least two, and may contain up to eight tokens.
        //Swap fee: the fee must be between 0.0001% and 10%

        uint public constant MIN_FEE = BONE / 10 ** 6;
        uint public constant MAX_FEE = BONE / 10;

        uint public constant MIN_WEIGHT = BONE;
        uint public constant MAX_WEIGHT = BONE * 50;
        uint public constant MAX_TOTAL_WEIGHT = BONE * 50;
        uint public constant MIN_BALANCE = BONE / 10 ** 12;

        uint public constant INIT_POOL_SUPPLY = BONE * 100;

        uint public constant MIN_BPOW_BASE = 1;
        uint public constant MAX_BPOW_BASE = (2 * BONE) - 1;
        uint public constant BPOW_PRECISION = BONE / 10 ** 10;

        uint public constant MAX_IN_RATIO = BONE / 2;
        uint public constant MAX_OUT_RATIO = (BONE / 3) + 1;

        uint public constant CENTI_FEE = BONE / 100;
}
```

BPool.sol

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.   See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program.   If not, see <http://www.gnu.org/licenses/>.

pragma solidity ^0.5.9;

import "./TokenBase.sol";
import "./BMath.sol";
import "./IFactoryManager.sol";

contract BPool is BToken, BMath {

        struct Record {
                bool bound;     // is token bound to pool
                uint index;     // private
                uint denorm;    // denormalized weight
                uint balance;
        }

        event LOG_SWAP(
                address indexed caller,
                address indexed tokenIn,
                address indexed tokenOut,
                uint256 tokenAmountIn,
        uint256 tokenAmountOut
    );

    event LOG_JOIN(
        address indexed caller,
        address indexed tokenIn,
        uint256 tokenAmountIn
    );

    event LOG_EXIT(
        address indexed caller,
        address indexed tokenOut,
        uint256 tokenAmountOut
    );

    event LOG_CALL(
        bytes4  indexed sig,
        address indexed caller,
        bytes data
    ) anonymous;

    modifier _lock_() {
        require(!_mutex, "ERR_REENTRY");
        _mutex = true;
        _;
        _mutex = false;
    }

    modifier _viewlock_() {
        require(!_mutex, "ERR_REENTRY");
        _;
    }

    bool private _mutex;

    address private _factory;      // BFactory address to push token exitFee to
    address public factoryManager;
    address private _controller; // has CONTROL role
    bool private _publicSwap; // true if PUBLIC can call SWAP functions

    // `setSwapFee` and `finalize` require CONTROL
    // `finalize` sets `PUBLIC can SWAP`, `PUBLIC can JOIN`

    mapping(address => uint) private tokenFundForFox;
    mapping(address => uint) private tokenFundForBuilder;

    uint private _swapFeeForBuilder;

    uint private _swapFeeForLp;

    bool private _finalized;

    address[] private _tokens;
    mapping(address => Record) private  _records;
    uint private _totalWeight;



    constructor(address _factoryManager) public {
        _controller = msg.sender;
        _factory = msg.sender;
        factoryManager = _factoryManager;
        _swapFeeForLp = MIN_FEE;
        _swapFeeForBuilder = 0;
        _publicSwap = false;
        _finalized = false;
    }

    function isPublicSwap() external view returns (bool){
        return _publicSwap;
    }

    function isFinalized() external view returns (bool){
        return _finalized;
    }

    function isBound(address t) external view returns (bool){
        return _records[t].bound;
    }

    function getNumTokens() external view returns (uint){
        return _tokens.length;
    }

    function getCurrentTokens() external view _viewlock_ returns (address[] memory tokens){
        return _tokens;
    }
```

```
function getFinalTokens() external view _viewlock_ returns (address[] memory tokens){
        require(_finalized, "ERR_NOT_FINALIZED");
        return _tokens;
}

function getDenormalizedWeight(address token) external view _viewlock_ returns (uint){

        require(_records[token].bound, "ERR_NOT_BOUND");
        return _records[token].denorm;
}

function getTotalDenormalizedWeight() external view _viewlock_ returns (uint){
        return _totalWeight;
}

function getNormalizedWeight(address token) external view _viewlock_ returns (uint){
        require(_records[token].bound, "ERR_NOT_BOUND");
        uint denorm = _records[token].denorm;
        return bdiv(denorm, _totalWeight);
}

function getDenorm(address token) external view _viewlock_ returns (uint){
        require(_records[token].bound, "ERR_NOT_BOUND");
        return _records[token].denorm;
}

function getBalance(address token) external view _viewlock_ returns (uint){
        require(_records[token].bound, "ERR_NOT_BOUND");
        return _records[token].balance;
}

function getSwapFee() external view _viewlock_ returns (uint){
        return _getSwapFee();
}

function getController() external view _viewlock_ returns (address){
        return _controller;
}


function setSwapLpFee(uint _swapFee) external _lock_ {
        require(!_finalized, "ERR_IS_FINALIZED");
        require(msg.sender == _controller, "ERR_NOT_CONTROLLER");
        require(_swapFee >= MIN_FEE, "ERR_MIN_FEE");
        require(_swapFee <= CENTI_FEE, "ERR_MAX_FEE");
        _swapFeeForLp = _swapFee;
}


function setSwapBuilderFee(uint _swapFee) external _lock_ {
        require(!_finalized, "ERR_IS_FINALIZED");
        require(msg.sender == _controller, "ERR_NOT_CONTROLLER");
        require(_swapFee >= MIN_FEE, "ERR_MIN_FEE");
        require(_swapFee <= CENTI_FEE, "ERR_MAX_FEE");
        _swapFeeForBuilder = _swapFee;
}

function setController(address manager) external _lock_ {
        require(msg.sender == _controller, "ERR_NOT_CONTROLLER");
        _controller = manager;
}

function setPublicSwap(bool public_) external _lock_ {
        require(!_finalized, "ERR_IS_FINALIZED");
        require(msg.sender == _controller, "ERR_NOT_CONTROLLER");
        _publicSwap = public_;
}

function addToken(
        address tokenIn,
        uint balanceIn,
        uint denormIn,
        address tokenOut,
        uint balanceOut,
        uint denormOut
)external{
        require(msg.sender == _controller, "ERR_NOT_CONTROLLER");
        bind(tokenIn, balanceIn, denormIn);
        bind(tokenOut, balanceOut, denormOut);
        finalize();

}

function finalize() public _lock_ {
        require(msg.sender == _controller, "ERR_NOT_CONTROLLER");
        require(!_finalized, "ERR_IS_FINALIZED");
        require(_tokens.length >= MIN_BOUND_TOKENS, "ERR_MIN_TOKENS");

        _finalized = true;
        _publicSwap = true;

        _mintPoolShare(INIT_POOL_SUPPLY);
        _pushPoolShare(msg.sender, INIT_POOL_SUPPLY);
    }


    function bind(address token, uint balance, uint denorm) public {
        // _lock_    Bind does not lock because it jumps to `rebind`, which does
        require(msg.sender == _controller, "ERR_NOT_CONTROLLER");
        require(!_records[token].bound, "ERR_IS_BOUND");
        require(!_finalized, "ERR_IS_FINALIZED");

        require(_tokens.length < MAX_BOUND_TOKENS, "ERR_MAX_TOKENS");

        _records[token] = Record({
        bound : true,
        index : _tokens.length,
        denorm : 0, // balance and denorm will be validated
        balance : 0     // and set by `rebind`
        });
        _tokens.push(token);
        rebind(token, balance, denorm);
    }

    function rebind(address token, uint balance, uint denorm) public _lock_ {

        require(msg.sender == _controller, "ERR_NOT_CONTROLLER");
        require(_records[token].bound, "ERR_NOT_BOUND");
        require(!_finalized, "ERR_IS_FINALIZED");

        require(denorm >= MIN_WEIGHT, "ERR_MIN_WEIGHT");
        require(denorm <= MAX_WEIGHT, "ERR_MAX_WEIGHT");
        require(balance >= MIN_BALANCE, "ERR_MIN_BALANCE");

        // Adjust the denorm and totalWeight
        uint oldWeight = _records[token].denorm;
        if (denorm > oldWeight) {
            _totalWeight = badd(_totalWeight, bsub(denorm, oldWeight));
            require(_totalWeight <= MAX_TOTAL_WEIGHT, "ERR_MAX_TOTAL_WEIGHT");
        } else if (denorm < oldWeight) {
            _totalWeight = bsub(_totalWeight, bsub(oldWeight, denorm));
        }
        _records[token].denorm = denorm;

        // Adjust the balance record and actual token balance
        uint oldBalance = _records[token].balance;
        _records[token].balance = balance;
        if (balance > oldBalance) {
            _pullUnderlying(token, msg.sender, bsub(balance, oldBalance));
        } else if (balance < oldBalance) {
            // In this case liquidity is being withdrawn, so charge EXIT_FEE
            uint tokenBalanceWithdrawn = bsub(oldBalance, balance);
            uint _exitFee = IFactoryManager(factoryManager).exitFee();
            uint _tokenExitAmount = bmul(tokenBalanceWithdrawn, _exitFee);
            _pushUnderlying(token, msg.sender, bsub(tokenBalanceWithdrawn, _tokenExitAmount));
            _pushUnderlying(token, factoryManager, _tokenExitAmount);
        }
    }

    function unbind(address token) external _lock_ {

        require(msg.sender == _controller, "ERR_NOT_CONTROLLER");
        require(_records[token].bound, "ERR_NOT_BOUND");
        require(!_finalized, "ERR_IS_FINALIZED");

        uint tokenBalance = _records[token].balance;
        uint _exitFee = IFactoryManager(factoryManager).exitFee();
        uint _tokenExitAmount = bmul(tokenBalance, _exitFee);

        _totalWeight = bsub(_totalWeight, _records[token].denorm);

        // Swap the token-to-unbind with the last token,
        // then delete the last token
        uint index = _records[token].index;
        uint last = _tokens.length - 1;
        _tokens[index] = _tokens[last];
        _records[_tokens[index]].index = index;
        _tokens.pop();
        _records[token] = Record({
        bound : false,
        index : 0,
        denorm : 0,
        balance : 0
        });

        _pushUnderlying(token, msg.sender, bsub(tokenBalance, _tokenExitAmount));
        _pushUnderlying(token, factoryManager, _tokenExitAmount);
    }

    // Absorb any tokens that have been sent to this contract into the pool
    function gulp(address token) external _lock_ {
        require(_records[token].bound, "ERR_NOT_BOUND");
        _records[token].balance = IERC20(token).balanceOf(address(this));
    }

    function getSpotPrice(address tokenIn, address tokenOut) external view _viewlock_ returns (uint spotPrice){
        require(_records[tokenIn].bound, "ERR_NOT_BOUND");
        require(_records[tokenOut].bound, "ERR_NOT_BOUND");
        Record storage inRecord = _records[tokenIn];
        Record storage outRecord = _records[tokenOut];
        uint _swapFee = _getSwapFee();
        return  calcSpotPrice(inRecord.balance,  inRecord.denorm,  outRecord.balance,  outRecord.denorm,
_swapFee);
    }


    function  getSpotPriceSansFee(address  tokenIn,  address  tokenOut)  external  view  _viewlock_  returns  (uint
spotPrice){
        require(_records[tokenIn].bound, "ERR_NOT_BOUND");
        require(_records[tokenOut].bound, "ERR_NOT_BOUND");
        Record storage inRecord = _records[tokenIn];
        Record storage outRecord = _records[tokenOut];
        return calcSpotPrice(inRecord.balance, inRecord.denorm, outRecord.balance, outRecord.denorm, 0);
    }


    function joinPool(uint poolAmountOut, uint[] calldata maxAmountsIn) external _lock_ {
        require(_finalized, "ERR_NOT_FINALIZED");

        uint poolTotal = totalSupply();
        uint ratio = bdiv(poolAmountOut, poolTotal);
        require(ratio != 0, "ERR_MATH_APPROX");

        for (uint i = 0; i < _tokens.length; i++) {
            address t = _tokens[i];
            uint bal = _records[t].balance;
            uint tokenAmountIn = bmul(ratio, bal);
            require(tokenAmountIn != 0, "ERR_MATH_APPROX");
            require(tokenAmountIn <= maxAmountsIn[i], "ERR_LIMIT_IN");
            _records[t].balance = badd(_records[t].balance, tokenAmountIn);
            emit LOG_JOIN(msg.sender, t, tokenAmountIn);
            _pullUnderlying(t, msg.sender, tokenAmountIn);
        }
        _mintPoolShare(poolAmountOut);
        _pushPoolShare(msg.sender, poolAmountOut);
    }

    function exitPool(uint poolAmountIn, uint[] calldata minAmountsOut) external _lock_ {
        require(_finalized, "ERR_NOT_FINALIZED");

        uint poolTotal = totalSupply();
        uint _exitFee = IFactoryManager(factoryManager).exitFee();
        uint _exitAmount = bmul(poolAmountIn, _exitFee);
        uint pAiAfterExitFee = bsub(poolAmountIn, _exitAmount);
        uint ratio = bdiv(pAiAfterExitFee, poolTotal);
        require(ratio != 0, "ERR_MATH_APPROX");

        _pullPoolShare(msg.sender, poolAmountIn);
        _pushPoolShare(factoryManager, _exitAmount);
        _burnPoolShare(pAiAfterExitFee);

        for (uint i = 0; i < _tokens.length; i++) {
            address t = _tokens[i];
            uint bal = _records[t].balance;
            uint tokenAmountOut = bmul(ratio, bal);
            require(tokenAmountOut != 0, "ERR_MATH_APPROX");
            require(tokenAmountOut >= minAmountsOut[i], "ERR_LIMIT_OUT");
            _records[t].balance = bsub(_records[t].balance, tokenAmountOut);
            emit LOG_EXIT(msg.sender, t, tokenAmountOut);
            _pushUnderlying(t, msg.sender, tokenAmountOut);
        }

    }


    function swapExactAmountIn(
        address tokenIn,
        uint tokenAmountIn,
        address tokenOut,
        uint minAmountOut,
        uint maxPrice
    ) external _lock_ returns (uint tokenAmountOut, uint spotPriceAfter){

        require(_records[tokenIn].bound, "ERR_NOT_BOUND");
        require(_records[tokenOut].bound, "ERR_NOT_BOUND");
        require(_publicSwap, "ERR_SWAP_NOT_PUBLIC");

        Record storage inRecord = _records[address(tokenIn)];
        Record storage outRecord = _records[address(tokenOut)];
        uint _swapFee = _getSwapFee();
        require(tokenAmountIn <= bmul(inRecord.balance, MAX_IN_RATIO), "ERR_MAX_IN_RATIO");

        uint spotPriceBefore = calcSpotPrice(
            inRecord.balance,
            inRecord.denorm,
            outRecord.balance,
            outRecord.denorm,
            _swapFee
        );
        require(spotPriceBefore <= maxPrice, "ERR_BAD_LIMIT_PRICE");

        tokenAmountOut = calcOutGivenIn(
            inRecord.balance,
            inRecord.denorm,
            outRecord.balance,
            outRecord.denorm,
            tokenAmountIn,
            _swapFee
        );
        require(tokenAmountOut >= minAmountOut, "ERR_LIMIT_OUT");

        _chargeFee(tokenIn, tokenAmountIn);

        inRecord.balance = badd(inRecord.balance, tokenAmountIn);
        outRecord.balance = bsub(outRecord.balance, tokenAmountOut);

        spotPriceAfter = calcSpotPrice(
            inRecord.balance,
            inRecord.denorm,
            outRecord.balance,
            outRecord.denorm,
            _swapFee
        );
        require(spotPriceAfter >= spotPriceBefore, "ERR_MATH_APPROX");
        require(spotPriceAfter <= maxPrice, "ERR_LIMIT_PRICE");
        require(spotPriceBefore <= bdiv(tokenAmountIn, tokenAmountOut), "ERR_MATH_APPROX");

        emit LOG_SWAP(msg.sender, tokenIn, tokenOut, tokenAmountIn, tokenAmountOut);

        _pullUnderlying(tokenIn, msg.sender, tokenAmountIn);
        _pushUnderlying(tokenOut, msg.sender, tokenAmountOut);

        return (tokenAmountOut, spotPriceAfter);
    }

    function swapExactAmountOut(
        address tokenIn,
        uint maxAmountIn,
        address tokenOut,
        uint tokenAmountOut,
        uint maxPrice
    ) external _lock_ returns (uint tokenAmountIn, uint spotPriceAfter){
        require(_records[tokenIn].bound, "ERR_NOT_BOUND");
        require(_records[tokenOut].bound, "ERR_NOT_BOUND");
        require(_publicSwap, "ERR_SWAP_NOT_PUBLIC");

        uint _swapFee = _getSwapFee();
        Record storage inRecord = _records[address(tokenIn)];
        Record storage outRecord = _records[address(tokenOut)];

        require(tokenAmountOut          <=          bmul(outRecord.balance,          MAX_OUT_RATIO),
"ERR_MAX_OUT_RATIO");

        uint spotPriceBefore = calcSpotPrice(
            inRecord.balance,
            inRecord.denorm,
            outRecord.balance,
            outRecord.denorm,
            _swapFee
        );
        require(spotPriceBefore <= maxPrice, "ERR_BAD_LIMIT_PRICE");

        tokenAmountIn = calcInGivenOut(
            inRecord.balance,
            inRecord.denorm,
            outRecord.balance,
            outRecord.denorm,
            tokenAmountOut,
            _swapFee
        );
        require(tokenAmountIn <= maxAmountIn, "ERR_LIMIT_IN");
        _chargeFee(tokenIn, tokenAmountIn);

        inRecord.balance = badd(inRecord.balance, tokenAmountIn);
        outRecord.balance = bsub(outRecord.balance, tokenAmountOut);

        spotPriceAfter = calcSpotPrice(
            inRecord.balance,
            inRecord.denorm,
            outRecord.balance,
            outRecord.denorm,
            _swapFee
        );
        require(spotPriceAfter >= spotPriceBefore, "ERR_MATH_APPROX");
        require(spotPriceAfter <= maxPrice, "ERR_LIMIT_PRICE");
        require(spotPriceBefore <= bdiv(tokenAmountIn, tokenAmountOut), "ERR_MATH_APPROX");

        emit LOG_SWAP(msg.sender, tokenIn, tokenOut, tokenAmountIn, tokenAmountOut);

        _pullUnderlying(tokenIn, msg.sender, tokenAmountIn);
        _pushUnderlying(tokenOut, msg.sender, tokenAmountOut);

        return (tokenAmountIn, spotPriceAfter);
    }


    function joinswapExternAmountIn(
        address tokenIn, uint tokenAmountIn, uint minPoolAmountOut
    ) external _lock_ returns (uint poolAmountOut){
        require(_finalized, "ERR_NOT_FINALIZED");
        require(_records[tokenIn].bound, "ERR_NOT_BOUND");
        require(tokenAmountIn          <=          bmul(_records[tokenIn].balance,          MAX_IN_RATIO),
"ERR_MAX_IN_RATIO");
        uint _swapFee = _getSwapFee();
        Record storage inRecord = _records[tokenIn];

        poolAmountOut = calcPoolOutGivenSingleIn(
            inRecord.balance,
            inRecord.denorm,
            _totalSupply,
            _totalWeight,
            tokenAmountIn,
            _swapFee
        );

        require(poolAmountOut >= minPoolAmountOut, "ERR_LIMIT_OUT");
        _chargeFee(tokenIn, tokenAmountIn);
        inRecord.balance = badd(inRecord.balance, tokenAmountIn);

        emit LOG_JOIN(msg.sender, tokenIn, tokenAmountIn);

        _mintPoolShare(poolAmountOut);
        _pushPoolShare(msg.sender, poolAmountOut);
        _pullUnderlying(tokenIn, msg.sender, tokenAmountIn);

        return poolAmountOut;
    }

    function joinswapPoolAmountOut(
        address tokenIn, uint poolAmountOut, uint maxAmountIn
    ) external _lock_ returns (uint tokenAmountIn){
        require(_finalized, "ERR_NOT_FINALIZED");
        require(_records[tokenIn].bound, "ERR_NOT_BOUND");

        Record storage inRecord = _records[tokenIn];
        uint _swapFee = _getSwapFee();
        tokenAmountIn = calcSingleInGivenPoolOut(
            inRecord.balance,
            inRecord.denorm,
            _totalSupply,
            _totalWeight,
            poolAmountOut,
            _swapFee
        );

        require(tokenAmountIn != 0, "ERR_MATH_APPROX");
        require(tokenAmountIn <= maxAmountIn, "ERR_LIMIT_IN");

        require(tokenAmountIn          <=          bmul(_records[tokenIn].balance,          MAX_IN_RATIO),
"ERR_MAX_IN_RATIO");

        _chargeFee(tokenIn, tokenAmountIn);

        inRecord.balance = badd(inRecord.balance, tokenAmountIn);
        emit LOG_JOIN(msg.sender, tokenIn, tokenAmountIn);

        _mintPoolShare(poolAmountOut);
        _pushPoolShare(msg.sender, poolAmountOut);
        _pullUnderlying(tokenIn, msg.sender, tokenAmountIn);

        return tokenAmountIn;
    }

    function exitswapPoolAmountIn(
        address tokenOut, uint poolAmountIn, uint minAmountOut
    ) external _lock_ returns (uint tokenAmountOut){
        require(_finalized, "ERR_NOT_FINALIZED");
        require(_records[tokenOut].bound, "ERR_NOT_BOUND");

        uint _swapFee = _getSwapFee();
        Record storage outRecord = _records[tokenOut];
        uint _exitFee = IFactoryManager(factoryManager).exitFee();
        tokenAmountOut = calcSingleOutGivenPoolIn(
            outRecord.balance,
            outRecord.denorm,
            _totalSupply,
            _totalWeight,
            poolAmountIn,
            _swapFee,
            _exitFee
        );

        require(tokenAmountOut >= minAmountOut, "ERR_LIMIT_OUT");

        require(tokenAmountOut          <=          bmul(_records[tokenOut].balance,          MAX_OUT_RATIO),
"ERR_MAX_OUT_RATIO");
        _chargeFee(tokenOut, tokenAmountOut);
        outRecord.balance = bsub(outRecord.balance, tokenAmountOut);
        uint _exitAmount = bmul(poolAmountIn, _exitFee);

        emit LOG_EXIT(msg.sender, tokenOut, tokenAmountOut);

        _pullPoolShare(msg.sender, poolAmountIn);
        _burnPoolShare(bsub(poolAmountIn, _exitAmount));
        _pushPoolShare(factoryManager, _exitAmount);
        _pushUnderlying(tokenOut, msg.sender, tokenAmountOut);

        return tokenAmountOut;
    }

    function exitswapExternAmountOut(
        address tokenOut, uint tokenAmountOut, uint maxPoolAmountIn
    ) external _lock_ returns (uint poolAmountIn){
        require(_finalized, "ERR_NOT_FINALIZED");
        require(_records[tokenOut].bound, "ERR_NOT_BOUND");
        require(tokenAmountOut          <=          bmul(_records[tokenOut].balance,          MAX_OUT_RATIO),
"ERR_MAX_OUT_RATIO");

        uint _swapFee = _getSwapFee();
        Record storage outRecord = _records[tokenOut];
        uint _exitFee = IFactoryManager(factoryManager).exitFee();
        poolAmountIn = calcPoolInGivenSingleOut(
            outRecord.balance,
            outRecord.denorm,
            _totalSupply,
            _totalWeight,
            tokenAmountOut,
            _swapFee,
            _exitFee
        );

        require(poolAmountIn != 0, "ERR_MATH_APPROX");
        require(poolAmountIn <= maxPoolAmountIn, "ERR_LIMIT_IN");
        _chargeFee(tokenOut, tokenAmountOut);
        outRecord.balance = bsub(outRecord.balance, tokenAmountOut);


        uint _exitAmount = bmul(poolAmountIn, _exitFee);

        emit LOG_EXIT(msg.sender, tokenOut, tokenAmountOut);

        _pullPoolShare(msg.sender, poolAmountIn);
        _burnPoolShare(bsub(poolAmountIn, _exitAmount));
        _pushPoolShare(factoryManager, _exitAmount);
        _pushUnderlying(tokenOut, msg.sender, tokenAmountOut);

        return poolAmountIn;
    }
    // ==
    // 'Underlying' token-manipulation functions make external calls but are NOT locked
    // You must `_lock_` or otherwise ensure reentry-safety

    function _pullUnderlying(address erc20, address from, uint amount) internal {
        IERC20(erc20).transferFrom(from, address(this), amount);
    }

    function _pushUnderlying(address erc20, address to, uint amount) internal {
        IERC20(erc20).transfer(to, amount);
    }

    function _pullPoolShare(address from, uint amount) internal {
        _pull(from, amount);
    }

    function _pushPoolShare(address to, uint amount) internal {
        _push(to, amount);
    }

    function _mintPoolShare(uint amount) internal {
        _mint(amount);
    }

    function _burnPoolShare(uint amount) internal {
        _burn(amount);
    }

    function _chargeFee(address _token, uint _amount) internal {
        uint foxFund = bmul(_amount, IFactoryManager(factoryManager).swapFeeForDex());//knownsec// 交
易所手续费分成
        tokenFundForFox[_token] = badd(tokenFundForFox[_token], foxFund);
        uint buildFund = bmul(_amount, _swapFeeForBuilder);//knownsec// 创建者手续费分成
        tokenFundForBuilder[_token] = badd(tokenFundForBuilder[_token], buildFund);
    }

    function swapFee() external view returns (uint){
        return _getSwapFee();
    }

    function getExitFee() external view returns (uint){
        return IFactoryManager(factoryManager).exitFee();
    }

    function _getSwapFee() internal view returns (uint){
        uint _fee = badd(_swapFeeForLp, _swapFeeForBuilder);
        uint _feeForDex = IFactoryManager(factoryManager).swapFeeForDex();
        return badd(_fee, _feeForDex);
    }

    function claimFactoryFund() public {//knownsec// 提取工厂合约管理员收益,仅工厂合约管理员调用
        require(msg.sender == factoryManager, "!factory manager");
        for (uint i = 0; i < _tokens.length; i++) {
            address token = _tokens[i];
            uint amount = tokenFundForFox[token];
            if (amount > 0) {
                IERC20(token).transfer(factoryManager, amount);
                tokenFundForFox[token] = 0;
            }
        }

    }

    function withdrawToken(address token) external {
        require(msg.sender == factoryManager, "!factory manager");
        require(!_records[token].bound, "token is bound");
        uint _amount = IERC20(token).balanceOf(address(this));
        IERC20(token).transfer(factoryManager, _amount);
    }

    function claimBuildFund() public {//knownsec// 提取创建者收益,仅 controller 调用
        require(msg.sender == _controller, "!controller");
        for (uint i = 0; i < _tokens.length; i++) {
            address token = _tokens[i];
            uint amount = tokenFundForBuilder[token];
            if (amount > 0) {
                IERC20(token).transfer(_controller, amount);
                tokenFundForBuilder[token] = 0;
            }
        }
    }


}
```

FactoryManager.sol

```
pragma solidity ^0.5.9;

import "./BParam.sol";
import "./BPool.sol";
import "./BMath.sol";
import "./BFactory.sol";
import "@openzeppelin/contracts/ownership/Ownable.sol";

contract FactoryManager is BParam, BMath, Ownable {

	uint256 public goverFundDivRate = 3 * CENTI_FEE;
	address public governaddr;

	uint256 public burnRate = 15 * CENTI_FEE;

	uint public    exitFee = 0;
	uint public swapFeeForDex = 0;
	address public controller;
	mapping(address => address) swapPools;

	address public dexToken;
	address public baseToken;
	address public basePool;

	address public factory;

	uint256 private setFactoryManagerCount = 0;
	uint256 private setBasePoolCount = 0;

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

	function setController(address _controller) onlyOwner external {//knownsec// 设置 controller 地址，仅
controller 调用
		controller = _controller;
	}

	function setFactory(address _factory) external {//knownsec// 设置 facotry 地址,最多调用 2 次,仅 controller
调用
		require(msg.sender == controller, "!controller");
		require(setFactoryManagerCount < 2, "limit factory address");
		factory = _factory;
		setFactoryManagerCount = badd(setFactoryManagerCount, 1);
	}

	function getSwapPool(address token) public view returns (address){//knownsec// 指定 token 查找交易池地
址
		return swapPools[token];
	}

	function addSwapPools(address token, address pool) public {//knownsec// 添加交易池,仅 controller 调用
		require(msg.sender == controller, "!controller");
		require(factory != address(0), "factory=0");
		require(BFactory(factory).isBPool(pool), "!pool");
		swapPools[token] = pool;

	}

	function removeSwapPools(address token) public {//knownsec// 移除交易池,仅 controller 调用
		require(msg.sender == controller, "!controller");
		swapPools[token] = address(0);

	}

	function setExitFee(uint _exitFee) public {
		require(msg.sender == controller, "!controller");
		require(_exitFee <= CENTI_FEE, "ERR_MAX_EXIT_FEE");
		exitFee = _exitFee;
	}

	function setSwapFeeForDex(uint _fee) public {//knownsec// 设置交易所的交易手续费分成,仅 controller 调
用
		require(msg.sender == controller, "!controller");
		require(_fee <= CENTI_FEE, "ERR_MAX_EXIT_FEE");
		swapFeeForDex = _fee;
	}

	function claimToken(address token) public {//knownsec// 将指定 token 全部转换为 baseToken,仅 controller
调用
		require(msg.sender == controller, "!controller");
		address _pool = swapPools[token];
		require(_pool != address(0), "not set pool");
		require(baseToken != address(0), "not set base token");
		swapToken(_pool, token, baseToken);
	}
```

```
    function swapToken(//knownsec//  指定交易池将 tokenIn 转换为 tokenOut, 仅 controller 调用
        address poolAddress, address tokenIn, address tokenOut
    ) public returns (uint){
        require(msg.sender == controller, "!controller");
        require(factory != address(0), "factory=0");
        require(BFactory(factory).isBPool(poolAddress), "!pool");
        BPool pool = BPool(poolAddress);
        uint _tokenInAmount = IERC20(tokenIn).balanceOf(address(this));
        IERC20(tokenIn).approve(poolAddress, _tokenInAmount);
        (uint tokenAmountOut,) = pool.swapExactAmountIn(tokenIn, _tokenInAmount, tokenOut, 0, MAX);
        return tokenAmountOut;
    }

    function addFoxPoolLiquidity() external {//knownsec// baseToken 扣除管理费3%, 销毁15%(转为dexToken),
剩余加入池中, 仅 controller 调用
        require(msg.sender == controller, "!controller");
        uint _amount = IERC20(baseToken).balanceOf(address(this));
        uint256 _goverFund = bmul(_amount, goverFundDivRate);
        IERC20(baseToken).transfer(governaddr, _goverFund);

        uint _tokenInAmount = bmul(_amount, burnRate);

        IERC20(baseToken).approve(basePool, _tokenInAmount);
        BPool(basePool).swapExactAmountIn(baseToken, _tokenInAmount, dexToken, 0, MAX);

        _amount = bsub(_amount, _goverFund);
        _amount = bsub(_amount, _tokenInAmount);

        IERC20(baseToken).approve(basePool, _amount);
        BPool(basePool).joinswapExternAmountIn(baseToken, _amount, 0);
    }

    function collect(BPool pool) public {//knownsec//  提取指定池收益
        require(msg.sender == controller, "!controller");
        pool.claimFactoryFund();
        removeLiquidity(pool);

    }

    function removeLiquidity(BPool pool) public {//knownsec//  移除指定池的所有流动性
        require(msg.sender == controller, "!controller");
        uint _amount = IERC20(pool).balanceOf(address(this));
        if (_amount > 0) {
            uint[] memory amountOuts = new uint[](pool.getNumTokens());
            pool.exitPool(_amount, amountOuts);
        }
    }

    function burnToken() public {//knownsec//  销毁本合约所有 dexToken, 无权限校验
        uint _amount = IERC20(dexToken).balanceOf(address(this));
        if (_amount > 0) {
            IERC20(dexToken).transfer(address(0), _amount);
        }
    }


    function setBasePoolToken(address _pool, address _dexToken, address _baseToken) public {
        require(msg.sender == controller, "!controller");
        require(factory != address(0), "factory is 0");
        require(BFactory(factory).isBPool(_pool), "!pool");
        require(BPool(_pool).isBound(_dexToken), "not bound");
        require(BPool(_pool).isBound(_baseToken), "not bound");
        require(setBasePoolCount < 2, "limit set base pool");
        setBasePoolCount = badd(setBasePoolCount, 1);
        basePool = _pool;
        dexToken = _dexToken;
        baseToken = _baseToken;

    }

    function gover(address _goveraddr) public {
        require(msg.sender == controller, "!controller");
        governaddr = _goveraddr;
    }

    function setBurnRate(uint _rate) public {
        require(msg.sender == controller, "!controller");
        require(_rate < 50 * CENTI_FEE, "< MAX_FEE");
        burnRate = _rate;
    }

    function setGoverFundDivRate(uint256 _goverFundDivRate) public {
        require(msg.sender == controller, "!controller");
        require(_goverFundDivRate < MAX_FEE, "< MAX_FEE");
        goverFundDivRate = _goverFundDivRate;
    }

    function claimOtherToken(address _pool, address _token) external {
        require(msg.sender == controller, "!controller");
        BPool(_pool).withdrawToken(_token);

    }

}
```

IFactoryManager.sol

```
pragma solidity ^0.5.9;

interface IFactoryManager {
    function exitFee() external view returns (uint);

    function swapFeeForDex() external view returns (uint);

}
```

Migrations.sol

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

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

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

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

PoolView.sol

```
pragma solidity ^0.5.9;

import "./BPool.sol";
import "./BMath.sol";


contract PoolView is BMath {

    BPool public pool;

    constructor(address _poolAddress) public {
        pool = BPool(_poolAddress);
    }

    function getInGivenOut(
        address tokenIn, address tokenOut, uint256 tokenInAmount
    ) external view returns (uint tokenOutAmount){
        require(pool.isBound(tokenIn), "ERR_NOT_BOUND");
        require(pool.isBound(tokenOut), "ERR_NOT_BOUND");

        uint inRecordBalance = pool.getBalance(tokenIn);
        uint inRecordDenorm = pool.getDenormalizedWeight(tokenIn);
        uint outRecordBalance = pool.getBalance(tokenOut);
        uint outRecordDenorm = pool.getDenormalizedWeight(tokenOut);

        uint _swapFee = pool.swapFee();

        tokenOutAmount = calcOutGivenIn(
            inRecordBalance,
            inRecordDenorm,
            outRecordBalance,
            outRecordDenorm,
            tokenInAmount,
            _swapFee
        );
        return tokenOutAmount;
    }

    function getTokenInGivenPoolOut(
        address[]    calldata tokens, uint[] calldata amounts
    ) external view returns (uint poolAmountOut){
        require(pool.isFinalized(), "ERR_NOT_FINALIZED");
        require(tokens.length == amounts.length, "!length");
        uint poolTotal = pool.totalSupply();
        uint minRatio = MAX;
        for (uint i = 0; i < tokens.length; i++) {
            address t = tokens[i];
            uint bal = pool.getBalance(t);
            uint ratio = bdiv(amounts[i], poolTotal);
            if (ratio <= minRatio) {
                minRatio = ratio;
            }
        }
        poolAmountOut = bmul(minRatio, poolTotal);
        return poolAmountOut;
    }

    function getSingleTokenInGivenPoolOut(
        address tokenIn, uint256 tokenAmountIn
    ) external view returns (uint poolAmountOut){
        require(pool.isFinalized(), "ERR_NOT_FINALIZED");
        require(pool.isBound(tokenIn), "ERR_NOT_BOUND");
        require(tokenAmountIn              <=            bmul(pool.getBalance(tokenIn),            MAX_IN_RATIO),
"ERR_MAX_IN_RATIO");

        uint _swapFee = pool.swapFee();
        uint inRecordBalance = pool.getBalance(tokenIn);
        uint inRecordDenorm = pool.getDenormalizedWeight(tokenIn);
        uint _totalSupply = pool.totalSupply();
        uint _totalWeight = pool.getTotalDenormalizedWeight();
        poolAmountOut = calcPoolOutGivenSingleIn(
            inRecordBalance,
            inRecordDenorm,
            _totalSupply,
            _totalWeight,
            tokenAmountIn,
            _swapFee
        );
        return poolAmountOut;
    }


    function getPoolInGivenTokenOut(
        address tokenOut, uint poolAmountIn, uint minAmountOut
    ) external view returns (uint tokenAmountOut){
        require(pool.isFinalized(), "ERR_NOT_FINALIZED");
        require(pool.isBound(tokenOut), "ERR_NOT_BOUND");

        uint _swapFee = pool.swapFee();
        uint outRecordBalance = pool.getBalance(tokenOut);
        uint outRecordDenorm = pool.getDenormalizedWeight(tokenOut);
        uint _totalSupply = pool.totalSupply();
        uint _totalWeight = pool.getTotalDenormalizedWeight();
        uint _exitFee = pool.getExitFee();
        tokenAmountOut = calcSingleOutGivenPoolIn(
            outRecordBalance,
            outRecordDenorm,
            _totalSupply,
            _totalWeight,
            poolAmountIn,
            _swapFee,
            _exitFee
        );
        return tokenAmountOut;
    }


}
```

TokenBase.sol

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.    See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program.    If not, see <http://www.gnu.org/licenses/>.

pragma solidity ^0.5.9;
```

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

contract TokenBase is BNum {

        mapping(address => uint)                         internal _balance;
        mapping(address => mapping(address=>uint)) internal _allowance;
        uint internal _totalSupply;

        event Approval(address indexed src, address indexed dst, uint amt);
        event Transfer(address indexed src, address indexed dst, uint amt);

        function _mint(uint amt) internal {
                _balance[address(this)] = badd(_balance[address(this)], amt);
                _totalSupply = badd(_totalSupply, amt);
                emit Transfer(address(0), address(this), amt);
        }

        function _burn(uint amt) internal {
                require(_balance[address(this)] >= amt, "ERR_INSUFFICIENT_BAL");
                _balance[address(this)] = bsub(_balance[address(this)], amt);
                _totalSupply = bsub(_totalSupply, amt);
                emit Transfer(address(this), address(0), amt);
        }

        function _move(address src, address dst, uint amt) internal {
                require(_balance[src] >= amt, "ERR_INSUFFICIENT_BAL");
                _balance[src] = bsub(_balance[src], amt);
                _balance[dst] = badd(_balance[dst], amt);
                emit Transfer(src, dst, amt);
        }

        function _push(address to, uint amt) internal {
                _move(address(this), to, amt);
        }

        function _pull(address from, uint amt) internal {
                _move(from, address(this), amt);
        }
}

contract BToken is TokenBase, IERC20 {

        string   private _name         = "Balancer Pool Token";
        string   private _symbol       = "BPT";
        uint8     private _decimals = 18;

        function name() public view returns (string memory) {
                return _name;
        }

        function symbol() public view returns (string memory) {
                return _symbol;
        }

        function decimals() public view returns(uint8) {
                return _decimals;
        }

        function allowance(address src, address dst) external view returns (uint) {
                return _allowance[src][dst];
        }

        function balanceOf(address whom) external view returns (uint) {
                return _balance[whom];
        }

        function totalSupply() public view returns (uint) {
                return _totalSupply;
        }

        function approve(address dst, uint amt) external returns (bool) {
                _allowance[msg.sender][dst] = amt;
                emit Approval(msg.sender, dst, amt);
                return true;
        }

        function increaseApproval(address dst, uint amt) external returns (bool) {
                _allowance[msg.sender][dst] = badd(_allowance[msg.sender][dst], amt);
                emit Approval(msg.sender, dst, _allowance[msg.sender][dst]);
                return true;
        }

        function decreaseApproval(address dst, uint amt) external returns (bool) {
                uint oldValue = _allowance[msg.sender][dst];
                if (amt > oldValue) {
```

```
                _allowance[msg.sender][dst] = 0;
            } else {
                _allowance[msg.sender][dst] = bsub(oldValue, amt);
            }
            emit Approval(msg.sender, dst, _allowance[msg.sender][dst]);
            return true;
        }

        function transfer(address dst, uint amt) external returns (bool) {
            _move(msg.sender, dst, amt);
            return true;
        }

        function transferFrom(address src, address dst, uint amt) external returns (bool) {
            require(msg.sender == src || amt <= _allowance[src][msg.sender], "ERR_BTOKEN_BAD_CALLER");
            _move(src, dst, amt);
            if (msg.sender != src && _allowance[src][msg.sender] != uint256(-1)) {
                _allowance[src][msg.sender] = bsub(_allowance[src][msg.sender], amt);
                emit Approval(msg.sender, dst, _allowance[src][msg.sender]);
            }
            return true;
        }
}
```

MockMath.sol

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.

pragma solidity ^0.5.9;

import "../BMath.sol";
import "../BNum.sol";

// Contract to wrap internal functions for testing

contract MockMath is BMath {

        function calc_btoi(uint a) external pure returns (uint) {
            return btoi(a);
        }

        function calc_bfloor(uint a) external pure returns (uint) {
            return bfloor(a);
        }

        function calc_badd(uint a, uint b) external pure returns (uint) {
            return badd(a, b);
        }

        function calc_bsub(uint a, uint b) external pure returns (uint) {
            return bsub(a, b);
        }

        function calc_bsubSign(uint a, uint b) external pure returns (uint, bool) {
            return bsubSign(a, b);
        }

        function calc_bmul(uint a, uint b) external pure returns (uint) {
            return bmul(a, b);
        }

        function calc_bdiv(uint a, uint b) external pure returns (uint) {
            return bdiv(a, b);
        }

        function calc_bpowi(uint a, uint n) external pure returns (uint) {
            return bpowi(a, n);
        }

        function calc_bpow(uint base, uint exp) external pure returns (uint) {
            return bpow(base, exp);
        }

        function calc_bpowApprox(uint base, uint exp, uint precision) external pure returns (uint) {
            return bpowApprox(base, exp, precision);
```

```
    }
}
```

MockToken.sol

```
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.

// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.   See the
// GNU General Public License for more details.

// You should have received a copy of the GNU General Public License
// along with this program.   If not, see <http://www.gnu.org/licenses/>.

pragma solidity ^0.5.9;

// Test Token

contract MockToken {

    string private _name;
    string private _symbol;
    uint8      private _decimals;

    address private _owner;

    uint internal _totalSupply;

    mapping(address => uint)                        private _balance;
    mapping(address => mapping(address=>uint)) private _allowance;

    modifier _onlyOwner_() {
        require(msg.sender == _owner, "ERR_NOT_OWNER");
        _;
    }

    event Approval(address indexed src, address indexed dst, uint amt);
    event Transfer(address indexed src, address indexed dst, uint amt);

    // Math
    function add(uint a, uint b) internal pure returns (uint c) {
        require((c = a + b) >= a);
    }
    function sub(uint a, uint b) internal pure returns (uint c) {
        require((c = a - b) <= a);
    }

    constructor(
        string memory name,
        string memory symbol,
        uint8 decimals
    ) public {
        _name = name;
        _symbol = symbol;
        _decimals = decimals;
        _owner = msg.sender;
    }

    function name() public view returns (string memory) {
        return _name;
    }

    function symbol() public view returns (string memory) {
        return _symbol;
    }

    function decimals() public view returns(uint8) {
        return _decimals;
    }

    function _move(address src, address dst, uint amt) internal {
        require(_balance[src] >= amt, "ERR_INSUFFICIENT_BAL");
        _balance[src] = sub(_balance[src], amt);
        _balance[dst] = add(_balance[dst], amt);
        emit Transfer(src, dst, amt);
    }

    function _push(address to, uint amt) internal {
        _move(address(this), to, amt);
    }

    function _pull(address from, uint amt) internal {
        _move(from, address(this), amt);
    }

    function _mint(address dst, uint amt) internal {
        _balance[dst] = add(_balance[dst], amt);
        _totalSupply = add(_totalSupply, amt);
        emit Transfer(address(0), dst, amt);
    }

    function allowance(address src, address dst) external view returns (uint) {
        return _allowance[src][dst];
    }

    function balanceOf(address whom) external view returns (uint) {
        return _balance[whom];
    }

    function totalSupply() public view returns (uint) {
        return _totalSupply;
    }

    function approve(address dst, uint amt) external returns (bool) {
        _allowance[msg.sender][dst] = amt;
        emit Approval(msg.sender, dst, amt);
        return true;
    }

    function mint(address dst, uint256 amt) public _onlyOwner_ returns (bool) {
        _mint(dst, amt);
        return true;
    }

    function burn(uint amt) public returns (bool) {
        require(_balance[address(this)] >= amt, "ERR_INSUFFICIENT_BAL");
        _balance[address(this)] = sub(_balance[address(this)], amt);
        _totalSupply = sub(_totalSupply, amt);
        emit Transfer(address(this), address(0), amt);
        return true;
    }

    function transfer(address dst, uint amt) external returns (bool) {
        _move(msg.sender, dst, amt);
        return true;
    }

    function transferFrom(address src, address dst, uint amt) external returns (bool) {
        require(msg.sender == src || amt <= _allowance[src][msg.sender], "ERR_BTOKEN_BAD_CALLER");
        _move(src, dst, amt);
        if (msg.sender != src && _allowance[src][msg.sender] != uint256(-1)) {
            _allowance[src][msg.sender] = sub(_allowance[src][msg.sender], amt);
            emit Approval(msg.sender, dst, _allowance[src][msg.sender]);
        }
        return true;
    }
}
```

FoxToken.sol

```
pragma solidity ^0.5.9;


import "@openzeppelin/contracts/token/ERC20/ERC20.sol";
import "@openzeppelin/contracts/token/ERC20/ERC20Detailed.sol";
import "@openzeppelin/contracts/ownership/Ownable.sol";


contract FoxToken is ERC20, ERC20Detailed, Ownable {
    uint256 public MAX = 31 * 1e6 * 1e18;
    constructor () public ERC20Detailed("foxdex", "fox", 18) {
    }

    function mint(address _to) public onlyOwner {//knownsec// 仅代币总量为 0 时 owner 可调用
        require(totalSupply() == 0, "mint one");
        _mint(_to, MAX);
    }

}
```

MasterChef.sol

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

pragma solidity ^0.5.9;

import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
import "@openzeppelin/contracts/utils/EnumerableSet.sol";
import "@openzeppelin/contracts/math/SafeMath.sol";
import "@openzeppelin/contracts/ownership/Ownable.sol";
import "./FoxToken.sol";

// MasterChef was the master of dex. He now governs over dex. He can make dex and he is a fair guy.
//
// Note that it's ownable and the owner wields tremendous power. The ownership
// will be transferred to a governance smart contract once dex is sufficiently
// distributed and the community can show to govern itself.
//
// Have fun reading it. Hopefully it's bug-free. God bless.
contract MasterChef is Ownable {
        using SafeMath for uint256;
        using EnumerableSet for EnumerableSet.AddressSet;

        // Info of each user.
        struct UserInfo {
                uint256 amount; // How many LP tokens the user has provided.
                uint256 rewardDebt; // Reward debt. See explanation below.
                //
                // We do some fancy math here. Basically, any point in time, the amount of dex
                // entitled to a user but is pending to be distributed is:
                //
                //      pending reward = (user.amount * pool.accPicklePerShare) - user.rewardDebt
                //
                // Whenever a user deposits or withdraws LP tokens to a pool. Here's what happens:
                //      1. The pool's `accPicklePerShare` (and `lastRewardBlock`) gets updated.
                //      2. User receives the pending reward sent to his/her address.
                //      3. User's `amount` gets updated.
                //      4. User's `rewardDebt` gets updated.
        }

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


        // 7 days
        uint256 public duration = 7 * 28800;//knownsec// 周期 7 天

        FoxToken public foxToken;
        // Dev fund (25%, initially)
        uint256 public devFundDivRate = 4;//knownsec// 开发者奖励分成比率,1/4=25%
        // gover fund (5%, initially)
        uint256 public goverFundDivRate = 20;//knownsec// 治理奖励分成比率,1/20=5%
        // insurance fund (5%, initially)
        uint256 public insuranceFundDivRate = 20;//knownsec// 保险奖励分成比率,1/20=5%
        uint256 public tokenPerBlock = 3 * 1e18;//knownsec// 区块奖励 token 量
        // tokens created per block.
        uint256 public tokenPerBlockForReward;//knownsec// 扣除各费后用于分发的区块奖励 token 量

        address public controller;
        EnumerableSet.AddressSet private devaddrs;

        address public governaddr;
        address public insuranceaddr;

        uint256 public devFund;
        uint256 public goverFund;
        uint256 public insuranceFund;



        // Info of each pool.
        PoolInfo[] public poolInfo;
        // Info of each user that stakes LP tokens.
        mapping(uint256 => mapping(address => UserInfo)) public userInfo;
        // Total allocation points. Must be the sum of all allocation points in all pools.
        uint256 public totalAllocPoint = 0;
        uint256 decrement = 0;
        // The block number when PICKLE mining starts.
        uint256 public startBlock;
        uint256 public periodEndBlock;

        // Events
        event Recovered(address token, uint256 amount);
        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(
                FoxToken _fox,
                uint256 _startBlock,
                uint256 _duration
        ) public {
                foxToken = _fox;
                startBlock = _startBlock;
                duration = _duration;
                periodEndBlock = _startBlock.add(duration);
                tokenPerBlockForReward = calTokenPerBlock(tokenPerBlock);
                controller = msg.sender;//knownsec// 合约部署者为控制器地址
        }


        function calTokenPerBlock(uint256 _blockToken) view internal returns (uint256){//knownsec// 计算扣除各
费后的区块奖励
                uint256 _devFund = _blockToken.div(devFundDivRate);//knownsec// 25%
                uint256 _goverFund = _blockToken.div(goverFundDivRate);//knownsec// 5%
                uint256 _insuranceFund = _blockToken.div(insuranceFundDivRate);//knownsec// 5%

                return _blockToken.sub(_devFund).sub(_goverFund).sub(_insuranceFund);
        }

        function mintToken() onlyOwner public {//knownsec// 铸币,仅 owner 调用
                foxToken.mint(address(this));
        }

        function setController(address _controller) external {//knownsec// 转移 controller,仅 controller 外部调用
                require(msg.sender == controller, "!controller");
                controller = _controller;
        }

        function poolLength() external view returns (uint256) {
                return poolInfo.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(//knownsec// 仅 controller 调用
                uint256 _allocPoint,
                IERC20 _lpToken,
                bool _withUpdate
        ) public {
                require(msg.sender == controller, "!controller");
                if (_withUpdate) {
                    massUpdatePools();
                }
                uint256 lastRewardBlock = block.number > startBlock
                ? block.number
                : startBlock;
                totalAllocPoint = totalAllocPoint.add(_allocPoint);
                poolInfo.push(
                    PoolInfo(
                    {
                    lpToken : _lpToken,
                    allocPoint : _allocPoint,
                    lastRewardBlock : lastRewardBlock,
                    accFoxPerShare : 0,
                    perBlockToken : tokenPerBlockForReward
                    }
                    )
                );
        }

        // Update the given pool's PICKLE allocation point. Can only be called by the owner.
        function set(//knownsec// 仅 controller 调用
                uint256 _pid,
                uint256 _allocPoint,
                bool _withUpdate
        ) public {
                require(msg.sender == controller, "!controller");
                if (_withUpdate) {
                    massUpdatePools();
                }
                totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(
                    _allocPoint
                );
                poolInfo[_pid].allocPoint = _allocPoint;
        }

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

        // 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);
                }
        }

        function getBlockNum() public view returns (uint){
                return block.number;
        }
        // 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 = block.number - pool.lastRewardBlock;
                uint256 foxReward = multiplier
                .mul(calPerBlockToken(pool.perBlockToken))
                .mul(pool.allocPoint)
                .div(totalAllocPoint);
                pool.accFoxPerShare = pool.accFoxPerShare.add(
                    foxReward.mul(1e12).div(lpSupply)
                );
                pool.lastRewardBlock = block.number;
                if (block.number >= periodEndBlock) {//knownsec// 若周期结束
                    // Calculate the fund of the period
                    uint256 durationReward = tokenPerBlock.mul(duration);

                    uint256 _devFund = durationReward.div(devFundDivRate);
                    uint256 _goverFund = durationReward.div(goverFundDivRate);
                    uint256 _insuranceFund = durationReward.div(insuranceFundDivRate);

                    devFund = devFund.add(_devFund);
                    goverFund = goverFund.add(_goverFund);
                    insuranceFund = insuranceFund.add(_insuranceFund);

                    tokenPerBlock = tokenPerBlock.mul(98).div(100);//knownsec// 下一周期区块奖励调为 98%
                    tokenPerBlockForReward = calTokenPerBlock(tokenPerBlock);

                    pool.perBlockToken = tokenPerBlockForReward;
                    periodEndBlock = block.number.add(duration);//knownsec// 开启下一周期


                }
        }


        function calPerBlockToken(uint256 perBlockTokenOfPool) public view returns (uint256){
                if (perBlockTokenOfPool > tokenPerBlockForReward) {
                    return tokenPerBlockForReward;
                } else {
                    return perBlockTokenOfPool;
                }
        }

        // Deposit LP tokens to MasterChef for PICKLE allocation.
        function deposit(uint256 _pid, uint256 _amount) public {
                PoolInfo storage pool = poolInfo[_pid];
                UserInfo storage user = userInfo[_pid][msg.sender];
                updatePool(_pid);
                if (user.amount > 0) {
                    uint256 pending = user
                    .amount
                    .mul(pool.accFoxPerShare)
                    .div(1e12)
                    .sub(user.rewardDebt);
                    if (pending > 0) {
                        safeFoxTransfer(msg.sender, pending);
                    }
                }
                if (_amount > 0) {
                    pool.lpToken.transferFrom(
                        address(msg.sender),
                        address(this),
                        _amount
                    );
                    user.amount = user.amount.add(_amount);
                }
                user.rewardDebt = user.amount.mul(pool.accFoxPerShare).div(1e12);
                emit Deposit(msg.sender, _pid, _amount);
        }

        // Withdraw LP tokens from MasterChef.
        function withdraw(uint256 _pid, uint256 _amount) public {
                PoolInfo storage pool = poolInfo[_pid];
                UserInfo storage user = userInfo[_pid][msg.sender];
                require(user.amount >= _amount, "withdraw: not good");
                updatePool(_pid);
                uint256 pending = user.amount.mul(pool.accFoxPerShare).div(1e12).sub(
                    user.rewardDebt
                );
                safeFoxTransfer(msg.sender, pending);
                user.amount = user.amount.sub(_amount);
                user.rewardDebt = user.amount.mul(pool.accFoxPerShare).div(1e12);
                pool.lpToken.transfer(address(msg.sender), _amount);
                emit Withdraw(msg.sender, _pid, _amount);
        }

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

        // Safe pickle transfer function, just in case if rounding error causes pool to not have enough dex.
        function safeFoxTransfer(address _to, uint256 _amount) internal {
                uint256 pickleBal = foxToken.balanceOf(address(this));
                if (_amount > pickleBal) {
                    foxToken.transfer(_to, pickleBal);
                } else {
                    foxToken.transfer(_to, _amount);
                }
        }

        function claimOtherFund() public {//knownsec// 提取管理及保险的奖励,仅 controller 调用
                require(msg.sender == controller, "!controller");
                foxToken.transfer(governaddr, goverFund);
                foxToken.transfer(insuranceaddr, insuranceFund);
                goverFund = 0;
                insuranceFund = 0;
        }


        function claimDevFund() public {//knownsec// 提取开发者奖励,仅 controller 调用
                require(msg.sender == controller, "!controller");
                uint256 perDevFund = devFund.div(devaddrs.length());
                for (uint256 i = 0; i < devaddrs.length() - 1; i++) {
                    foxToken.transfer(devaddrs.get(i), perDevFund);
                }
                uint256 remainFund = devFund.sub(perDevFund.mul(devaddrs.length() - 1));
                foxToken.transfer(devaddrs.get(devaddrs.length() - 1), remainFund);
                devFund = 0;


        }

        function addDev(address _devaddr) public {//knownsec// 添加开发者地址,仅 controller 调用
                require(msg.sender == controller, "!controller");
                require(devaddrs.length() < 100, "less 100");
                devaddrs.add(_devaddr);
        }

        function removeDev(address _devaddr) public {//knownsec// 移除开发者地址,仅 controller 调用
```

```
        require(msg.sender == controller, "!controller");
        devaddrs.remove(_devaddr);
    }

    function contains(address _dev) public view returns (bool){
        return devaddrs.contains(_dev);
    }

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

    function setDevFundDivRate(uint256 _devFundDivRate) public {//knownsec// 设置开发者奖励分成比率,仅
controller 调用
        require(msg.sender == controller, "!controller");
        require(_devFundDivRate >= 3, "!devFundDivRate-0");
        devFundDivRate = _devFundDivRate;
    }

    function insurance(address _insuranceaddr) public {//knownsec// 设置保险地址,仅 controller 调用
        require(msg.sender == controller, "!controller");
        insuranceaddr = _insuranceaddr;
    }

    function setInsuranceFundDivRate(uint256 _insuranceFundDivRate) public {//knownsec// 设置保险奖励分
成比率,仅 controller 调用
        require(msg.sender == controller, "!controller");
        require(_insuranceFundDivRate >= 15, "!devFundDivRate-0");
        insuranceFundDivRate = _insuranceFundDivRate;
    }

    function gover(address _goveraddr) public {//knownsec// 设置 gover 地址,仅 controller 调用
        require(msg.sender == controller, "!controller");
        governaddr = _goveraddr;
    }

    function setGoverFundDivRate(uint256 _goverFundDivRate) public {//knownsec// 设置治理奖励分成比率,
仅 controller 调用
        require(msg.sender == controller, "!controller");
        require(_goverFundDivRate >= 15, "!devFundDivRate-0");
        goverFundDivRate = _goverFundDivRate;
    }

}
```

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 TRX 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 TRX 导致的拒绝服务漏洞、因 energy 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 TRX 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 energy 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

知道创宇安全审计人员专用工具包，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