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	20210120	知道创宇区块链安全研究团队	V2.0

文档信息

文档名称	文档版本号	报告编号	保密级别
HotpotFunds 智能合约审计报告	V2.0	31d3b7e9512a4c6abdd0c5d1e35e3c2e	项目组公开

声明

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

目录

1. 综述

本次报告有效测试时间是从 2021 年 1 月 14 日开始到 2021 年 1 月 20 日结束，在此期间针对 **HotpotFunds 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进行了全面的分析，未发现相关安全风险，故综合评定为**通过**。

本次智能合约安全审计结果： 通过

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

本次审计的报告信息：

报告编号：31d3b7e9512a4c6abdd0c5d1e35e3c2e

报告查询地址链接:

https://attest.im/attestation/searchresult?qurey=31d3b7e9512a4c6abdd0c5d1e35e3c2e

本次审计的目标信息：

条目	描述
Token 名称	HotpotFunds
代码类型	以太坊智能合约代码
代码语言	solidity
代码地址	https://github.com/HotpotFunds/HotpotFunds

合约文件及哈希：

合约文件	MD5
ICurve.sol	6DAE87A0E2EA44ADADFD6E2F19EE4823

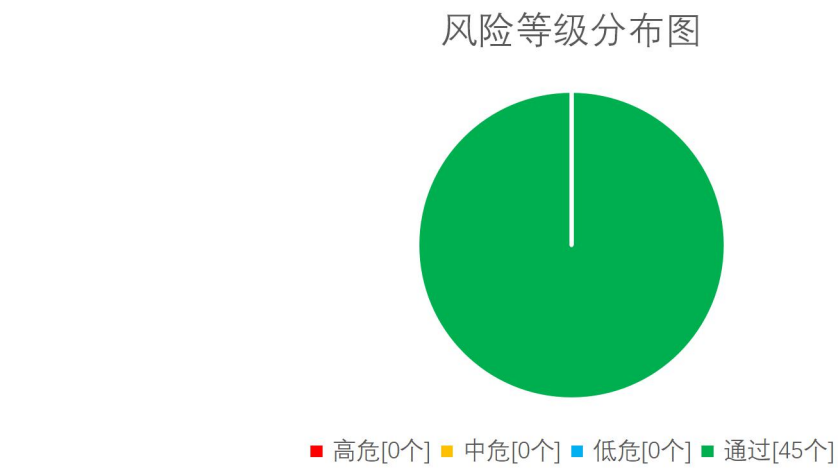
IERC20.sol	77E7A67AA796D62581AB860A1C6A3A13
IHotPot.sol	5C5FC8C5BB6CD6CA8945F34FBF343A0B
IHotPotController.sol	066A8D203E9AD43319CA9E8A3318DC41
IHotPotFund.sol	91F61936ED0700204834755EA93C899D
IHOtPotFundETH.sol	ED681BE3DD5341CB422A5ED9B1713500
IStakingRewards.sol	EDDAB49BC64D738DA064D5C774DE9690
IUniswapV2Factory.sol	58C36B20DF58A94EE9D608A7F6BCBCA2
IUniswapV2Pair.sol	55A3F9995D7034E6AE1767EE8B35E0A3
IUniswapV2Router.sol	961656556EEFE168442AD711CE0707F4
IWETH.sol	1E7D807A12A993F59BC5E889E45BC7AA
Address.sol	0A985EE077853BBE89F3651980AA9448
SafeERC20.sol	05F6EA9BA7FC8A7AE0B8A2CDB017AE09
SafeMath.sol	D052A711357716EF2569AF5AA730EEBE
HotPot.sol	65E12BD7B0B680FDD282689E4DF9EAD1
HotPotController.sol	985C923DECAFAEC7932C7CF6F3091DF8
HotPotFund.sol	1F50DC110D14E945DD387C78B14FCAF3
HotPotFundERC20.sol	D6A75BEEB6A2E1586B2A29830FA79CEC
HotPotFundETH.sol	F8F45CB9E4AD86A260A34879F32C51B4
ReentrancyGuard.sol	D9C6BBC0AEAFCE6F09465417DF7566B8
StakingRewards.sol	14127EA052DCA0C2DF5A59A86DFB0335

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	45



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	Transfer 逻辑设计	通过	经检测，不存在安全问题。
	TransferFrom 逻辑设计	通过	经检测，不存在安全问题。
	Burn 逻辑设计	通过	经检测，不存在安全问题。
	BurnFrom 逻辑设计	通过	经检测，不存在安全问题。
	Harvest 逻辑设计	通过	经检测，不存在安全问题。
	Stake 逻辑设计	通过	经检测，不存在安全问题。
	Withdraw 逻辑设计	通过	经检测，不存在安全问题。
	GetReward 逻辑设计	通过	经检测，不存在安全问题。
	NotifyRewardAmount 逻辑设计	通过	经检测，不存在安全问题。
	Deposit 逻辑设计	通过	经检测，不存在安全问题。
	Invest 逻辑设计	通过	经检测，不存在安全问题。
	SetUNIPool 逻辑设计	通过	经检测，不存在安全问题。
	MineUNI 逻辑设计	通过	经检测，不存在安全问题。
	MineUNIALL 逻辑设计	通过	经检测，不存在安全问题。
	Withdraw2 逻辑设计	通过	经检测，不存在安全问题。
	addPair 逻辑设计	通过	经检测，不存在安全问题。
	reBalance 逻辑设计	通过	经检测，不存在安全问题。
	RemovePair 逻辑设计	通过	经检测，不存在安全问题。
代码基本漏洞检测	编译器版本安全	通过	经检测，不存在该安全问题。
	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。
	require/assert 的合理使用	通过	经检测，不存在该安全问题。

	fallback 函数安全	通过	经检测，不存在该安全问题。
	tx.orgin 身份验证	通过	经检测，不存在该安全问题。
	owner 权限控制	通过	经检测，不存在该安全问题。
	gas 消耗检测	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	增发代币漏洞	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。
	错误使用随机数检测	通过	经检测，不存在该安全问题。
	不安全的接口使用	通过	经检测，不存在该安全问题。
	变量覆盖	通过	经检测，不存在该安全问题。
	未初始化的存储指针	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	交易顺序依赖检测	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	通过	经检测，不存在该安全问题。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	假充值漏洞检测	通过	经检测，不存在该安全问题。
	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	重排攻击检测	通过	经检测，不存在该安全问题。

3. 业务安全性检测

3.1. Transfer 逻辑设计【通过】

对 transfer 逻辑进行审计，查看是否有参数进行检查以及是否有溢出校验或使用 SafeMath 函数进行数值运算等。

检测结果：经审计，transfer 函数中有对参数进行检查，并使用 SafeMath 进行数值运算操作，未发现存在相关安全风险。

```
function transfer(address to, uint value) external returns (bool) {
        _transfer(msg.sender, to, value);//knownsec//调用 transfer 进行转账操作
        return true;
    }
    function _transfer(address from, address to, uint value) private {
        require(from != address(0), "ERC20: transfer from the zero address");//knownsec//对地址参数是否为空进行检查
        require(to != address(0), "ERC20: transfer to the zero address");//knownsec//对地址参数是否为空进行检查
        balanceOf[from] = balanceOf[from].sub(value);//knownsec//更新账户资产
        balanceOf[to] = balanceOf[to].add(value);//knownsec//更新账户资产
        emit Transfer(from, to, value);
    }
```

安全建议：无。

3.2. transferFrom 逻辑设计【通过】

对 transferFrom 逻辑进行审计，查看是否有参数进行检查以及是否有溢出校验或使用 SafeMath 函数进行数值运算，以及逻辑设计的合理性等。

检测结果：经审计，transferFrom 函数中有对参数进行检查，并使用 SafeMath

进行数值运算操作，逻辑设计合理，未发现存在相关安全风险。

```
function transferFrom(address from, address to, uint value) external returns (bool) {
    allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);//knownsec//更新授权转账额度
    _transfer(from, to, value);//knownsec//调用_transfer 进行转账
    return true;
}
function _transfer(address from, address to, uint value) private {
    require(from != address(0), "ERC20: transfer from the zero address");//knownsec//对地址参数是否为空进行检查
    require(to != address(0), "ERC20: transfer to the zero address");//knownsec//对地址参数是否为空进行检查
    balanceOf[from] = balanceOf[from].sub(value);//knownsec//更新账户资产
    balanceOf[to] = balanceOf[to].add(value);//knownsec//更新账户资产
    emit Transfer(from, to, value);
}
```

安全建议：无。

3.3. Burn 逻辑设计【通过】

对 burn 代币销毁逻辑进行审计，查看权限设计是否合理，以及是否有对参数进行合法性检查等。

审计结果：代币销毁只能销毁函数调用者自己的代币，且对参数有检查，未发现相关安全风险。

```
function burn(uint value) external returns (bool) {
    _burn(msg.sender, value);//knownsec//调用_burn 函数销毁代币
    return true;
}
function _burn(address from, uint value) internal {
    require(from != address(0), "ERC20: burn from the zero address");
```

```
            balanceOf[from] = balanceOf[from].sub(value);//knownsec//更新用户代币数量
            totalSupply = totalSupply.sub(value);//knownsec//更新代币总量
            emit Transfer(from, address(0), value);
        }
```

安全建议：无

3.4. burnFrom 逻辑设计【通过】

对 burnFrom 授权代为销毁代币逻辑设计进行审计，查看权限和逻辑设计的合理性以及是否有对参数进行检查等。

审计结果：未发现相关安全风险。

```
    function burnFrom(address from, uint value) external returns (bool) {
        allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);//knownsec//更新授权额度
        _burn(from, value);//knownsec//调用_burn 函数进行销毁代币处理
        return true;
    }
    function _burn(address from, uint value) internal {
            require(from != address(0), "ERC20: burn from the zero address");
            balanceOf[from] = balanceOf[from].sub(value);//knownsec//更新用户代币数量
            totalSupply = totalSupply.sub(value);//knownsec//更新代币总量
            emit Transfer(from, address(0), value);
        }
```

安全建议：无。

3.5. harvest 逻辑设计【通过】

对 harvest 业务逻辑进行审计，检查是否有对参数进行检查以及业务逻辑设计的合理性等。

审计结果：未发现相关安全风险。

```
function harvest(
    address token,
    uint amount
) public nonReentrant returns(uint burned) {
    uint value = amount <= IERC20(token).balanceOf(address(this)) ? amount :
IERC20(token).balanceOf(address(this));
    address pair = IUniswapV2Factory(UNISWAP_FACTORY).getPair(token, hotpot);
    require(pair != address(0), 'Pair not exist.');//knownsec//检查是否存在
    IERC20(token).safeApprove(UNISWAP_V2_ROUTER, value);
    address[] memory path = new address[](2);
    path[0] = token;
    path[1] = hotpot;
    uint[] memory amounts =
IUniswapV2Router(UNISWAP_V2_ROUTER).swapExactTokensForTokens(
        value,
        0,
        path,
        address(this), block.timestamp);
    IHotPot(hotpot).burn(amounts[1]);
    return amounts[1];
}
```

安全建议：无

3.6. stake 逻辑设计【通过】

对 stake 逻辑进行审计，检查是否有对参数进行检查以及业务逻辑设的合理性。

审计结果：未发现相关安全风险。

```
function stake(uint256 amount) external nonReentrant updateReward(msg.sender) {
    require(amount > 0, "Cannot stake 0"); //knownsec//数值检查
    _totalSupply = _totalSupply.add(amount);//knownsec//更新总量
    _balances[msg.sender] = _balances[msg.sender].add(amount);
```

```
        IERC20(stakingToken).safeTransferFrom(msg.sender, address(this), amount);
        emit Staked(msg.sender, amount);
    }
```

安全建议：无

3.7. Withdraw1 逻辑设计【通过】

对 Withdraw 逻辑进行审计，检查是否有对参数进行检查以及业务逻辑设的合理性。

审计结果：未发现相关安全风险。

```
        function withdraw(uint256 amount) public nonReentrant updateReward(msg.sender) {
        require(amount > 0, "Cannot withdraw 0");//knownsec//数值检查
        _totalSupply = _totalSupply.sub(amount);//knownsec//更新总量
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
        IERC20(stakingToken).safeTransfer(msg.sender, amount);
        emit Withdrawn(msg.sender, amount);
    }
```

安全建议：无

3.8. getReward 逻辑设计【通过】

对 getReward 逻辑进行审计，检查是否有对参数进行检查以及业务逻辑设计的合理性。

审计结果：未发现相关安全风险。

```
    function getReward() public nonReentrant updateReward(msg.sender) {
        uint256 reward = rewards[msg.sender];
        if (reward > 0) {//knownsec//判断 Reward 是否大于 0
            rewards[msg.sender] = 0;
            IERC20(rewardsToken).safeTransfer(msg.sender,
reward);//knownsec//提取 reward
            emit RewardPaid(msg.sender, reward);
        }
    }
```

安全建议：无

3.9. notifyRewardAmount【通过】

对 notifyRewardAmount 逻辑进行审计，检查是否有对参数进行检查以及权限和业务逻辑设计的合理性。

审计结果：未发现相关安全风险。

```
    function notifyRewardAmount(uint256 reward) external onlyRewardsDistribution updateReward(address(0)) {
        if (block.timestamp >= periodFinish) {//knownsec//timestamp 检查
            rewardRate = reward.div(rewardsDuration);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(rewardsDuration);
        }

        // Ensure the provided reward amount is not more than the balance in the contract.
        // This keeps the reward rate in the right range, preventing overflows due to
        // very high values of rewardRate in the earned and rewardsPerToken functions;
        // Reward + leftover must be less than 2^256 / 10^18 to avoid overflow.
        uint balance = IERC20(rewardsToken).balanceOf(address(this));
        require(rewardRate <= balance.div(rewardsDuration), "Provided reward too high");
       //knownsec//更新
        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(rewardsDuration);
        emit RewardAdded(reward);
    }
```

安全建议：无。

3.10. Deposit 逻辑设计【通过】

对 deposit 逻辑设计进行审计，检查是否有对参数进行检查以及业务逻辑设计的合理性。

审计结果：经审计，未发现相关安全风险，遂评定为通过。

```
function deposit(uint amount) public nonReentrant returns(uint share) {
    require(amount > 0, 'Are you kidding me?');//knownsec//数值检查
    uint _total_assets = totalAssets();
    IERC20(token).safeTransferFrom(msg.sender, address(this), amount);

    if(totalSupply == 0){
        share = amount;
    }
    else{
        share = amount.mul(totalSupply).div(_total_assets);
        // user uni debt
        uint debt = share.mul(totalDebts.add(totalUNIRewards())).div(totalSupply);
        if(debt > 0){
            debtOf[msg.sender] = debtOf[msg.sender].add(debt);
            totalDebts = totalDebts.add(debt);
        }
    }
    //knwonsec//更新操作
    investmentOf[msg.sender] = investmentOf[msg.sender].add(amount);
    totalInvestment = totalInvestment.add(amount);
    _mint(msg.sender, share);//knownsec//铸币操作
    emit Deposit(msg.sender, amount, share);
}
//knownsec//资产总量
 function totalAssets() public view returns(uint _assets) {
        address token0 = token;
```

```
            for(uint i=0; i<pairs.length; i++){//knownsec//循环整个交易对数组
                address token1 = pairs[i];
                IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0, token1));
                (uint reserve0, uint reserve1, ) = pair.getReserves();
                uint liquidity = pair.balanceOf(address(this)).add(stakingLPOf(address(pair)));
                if( pair.token0() == token0 )
                    _assets = _assets.add((reserve0 << 1).mul(liquidity).div(pair.totalSupply()));
                else // pair.token1() == token0
                    _assets = _assets.add((reserve1 << 1).mul(liquidity).div(pair.totalSupply()));
            }
            _assets = _assets.add(IERC20(token0).balanceOf(address(this)));
        }

        function stakingLPOf(address pair) public view returns(uint liquidity){
            if(uniPool[pair] != address(0)){
                liquidity = IStakingRewards(uniPool[pair]).balanceOf(address(this));
            }
        }
        function totalUNIRewards() public view returns(uint amount){
            amount = IERC20(UNI).balanceOf(address(this));
            for(uint i = 0; i < pairs.length; i++){
                IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token, pairs[i]));
                address stakingRewardAddr = uniPool[address(pair)];
                if(stakingRewardAddr != address(0)){
                    amount = amount.add(IStakingRewards(stakingRewardAddr).earned(address(this)));
                }
            }
```

```
    }
```

安全建议：无

3.11. invest 逻辑设计【通过】

对 invest 逻辑设计进行审计，检查是否有对参数进行检测以及权限检查和业务逻辑设计的合理性。

审计结果：未发现相关安全风险。

```
    function invest(uint amount, uint[] calldata proportions) external onlyController {
        uint len = pairs.length;
        require(len>0, 'Pairs is empty.');//knownsec//检查 Pair 是否存在
        address token0 = token;
        require(amount <= IERC20(token0).balanceOf(address(this)), "Not enough balance.");//knownsec//Token 数量检查
        require(proportions.length == pairs.length, 'Proportions index out of range.');

        uint _whole;
        for(uint i=0; i<len; i++){
            if(proportions[i] == 0) continue;
            _whole = _whole.add(proportions[i]);

            uint amount0 = (amount.mul(proportions[i]).div(DIVISOR)) >> 1;
            if(amount0 == 0) continue;

            address token1 = pairs[i];
            uint amount1 = _swap(token0, token1, amount0);
            (,uint amountB,) = IUniswapV2Router(UNISWAP_V2_ROUTER).addLiquidity(
                token0, token1,
                amount0, amount1,
                0, 0,
                address(this), block.timestamp
            );
```

```
                if(amount1 > amountB) _swap(token1, token0, amount1.sub(amountB));
            }
            require(_whole == DIVISOR, 'Error proportion.');//knownsec//比例检查
        }
```

安全建议：无

3.12 setUNIPool 逻辑设计【通过】

对 setUNIPool 逻辑进行审计，检查其权限设计是否合理，参数是否有做校验，逻辑设计是否存在错误等。

审计结果：未发现相关安全风险。

```
        function setUNIPool(address pair, address _uniPool) external onlyController {
            require(pair!= address(0) && _uniPool!= address(0), "Invalid address.");//knownsec//非 0 地址检查
            if(uniPool[pair] != address(0)){
                _withdrawStaking(IUniswapV2Pair(pair), totalSupply);
            }
            IERC20(pair).approve(_uniPool, 2**256-1);
            uniPool[pair] = _uniPool;
        }
        //knownsec//提取质押资产
        function _withdrawStaking(IUniswapV2Pair pair, uint share) internal returns(uint liquidity){
            address stakingRewardAddr = uniPool[address(pair)];
            if(stakingRewardAddr != address(0)){
                liquidity = IStakingRewards(stakingRewardAddr).balanceOf(address(this)).mul(share).div(totalSupply);
                if(liquidity > 0){
                    IStakingRewards(stakingRewardAddr).withdraw(liquidity);
                    IStakingRewards(stakingRewardAddr).getReward();
                }
            }
```

```
    }
```

安全建议：无

3.13 MineUNI 逻辑设计【通过】

对 mineUNI 业务逻辑进行审计，检查其权限和业务逻辑设计的合理性。

审计结果：未发现相关安全风险。

```
    function mineUNI(address pair) public onlyController {
        address stakingRewardAddr = uniPool[pair];
        if(stakingRewardAddr != address(0)){
            uint liquidity = IUniswapV2Pair(pair).balanceOf(address(this));
            if(liquidity > 0){
                IStakingRewards(stakingRewardAddr).stake(liquidity);
            }
        }
    }
```

安全建议：无

3.14 mineUNIAll 逻辑设计【通过】

对 mineUNIAll 业务逻辑进行审计，检查其权限和业务逻辑设计的合理性。

审计结果：未发现相关安全风险。

```
    function mineUNIAll() external onlyController {
        for(uint i = 0; i < pairs.length; i++){
            IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token, pairs[i]));
            address stakingRewardAddr = uniPool[address(pair)];
            if(stakingRewardAddr != address(0)){
                uint liquidity = pair.balanceOf(address(this));
                if(liquidity > 0){
                    IStakingRewards(stakingRewardAddr).stake(liquidity);
                }
            }
```

```
        }
    }
```

安全建议：无

3.15 Withdraw2 逻辑设计【通过】

对 Withdraw 业务逻辑进行审计，检查其权限和业务逻辑设计的合理性。

审计结果：未发现相关安全风险。

```
    function withdraw(uint share) public nonReentrant returns(uint amount) {
        require(share > 0 && share <= balanceOf[msg.sender], 'Not enough balance.');//knownsec//参数检查
        uint _investment;
        (amount, _investment) = _withdraw(msg.sender, share);//knownsec//提取逻辑
        investmentOf[msg.sender] = investmentOf[msg.sender].sub(_investment);//knownsec//更新操作
        totalInvestment = totalInvestment.sub(_investment);
        _burn(msg.sender, share);//knownsec//销毁
        IERC20(token).safeTransfer(msg.sender, amount);
        emit Withdraw(msg.sender, amount, share);
    }
    //knownsec//提款逻辑
    function _withdraw(
        address user,
        uint share
    ) internal returns (uint amount, uint investment) {
        address token0 = token;
        amount = IERC20(token0).balanceOf(address(this)).mul(share).div(totalSupply);
        for(uint i = 0; i < pairs.length; i++) {//knownsec//循环操作
            address token1 = pairs[i];
            IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0, token1));
            uint liquidity = pair.balanceOf(address(this)).mul(share).div(totalSupply);
            liquidity    = liquidity.add(_withdrawStaking(pair, share));
```

```
                if(liquidity == 0) continue;

                (uint amount0, uint amount1) = IUniswapV2Router(UNISWAP_V2_ROUTER).removeLiquidity(
                    token0, token1,
                    liquidity,
                    0, 0,
                    address(this), block.timestamp
                );
                amount = amount.add(amount0).add(_swap(token1, token0, amount1));
            }

            //withdraw UNI reward
            uint uniAmount = IERC20(UNI).balanceOf(address(this));
            uint totalAmount = totalDebts.add(uniAmount).mul(share).div(totalSupply);
            if(totalAmount > 0){
                uint debt = debtOf[user].mul(share).div(balanceOf[user]);
                debtOf[user] = debtOf[user].sub(debt);
                totalDebts = totalDebts.sub(debt);
                uint reward = totalAmount.sub(debt);
                if(reward > uniAmount) reward = uniAmount;
                if(reward > 0) IERC20(UNI).transfer(user, reward);
            }

            investment = investmentOf[user].mul(share).div(balanceOf[user]);
            if(amount > investment){
                uint _fee = (amount.sub(investment)).mul(FEE).div(DIVISOR);
                amount = amount.sub(_fee);
                IERC20(token0).safeTransfer(controller, _fee);
            }
            else {
                investment = amount;
            }
```

```
    }
```

安全建议：无

3.16 addPair 逻辑设计【通过】

对 addPair 逻辑设计进行审计，检查是否有对参数进行校验以及权限进行检查和业务逻辑设计的合理性。

审计结果：未发现相关安全风险。

```
    function addPair(address _token) external onlyController {
        address pair = IUniswapV2Factory(UNISWAP_FACTORY).getPair(token,
_token);
        require(pair != address(0), 'Pair not exist.');

        //approve for add liquidity and swap.
        IERC20(_token).safeApprove(UNISWAP_V2_ROUTER, 2**256-1);
        //approve for remove liquidity
        IUniswapV2Pair(pair).approve(UNISWAP_V2_ROUTER, 2**256-1);

        for(uint i = 0; i < pairs.length; i++) {
            require(pairs[i] != _token, 'Pair existed.');
        }
        pairs.push(_token);
    }
```

安全建议：无

3.17 reBalance 逻辑设计【通过】

对 reBalance 逻辑设计进行审计，检查是否有对参数进行检查以及逻辑设计的合理性等。

审计结果：未发现相关安全风险。

```
function reBalance(
    uint add_index,
    uint remove_index,
    uint liquidity
) external onlyController {
    require(remove_index < pairs.length, 'Pair index out of range.');

    //撤出&兑换
    address token0 = token;
    address token1 = pairs[remove_index];
    IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0, token1));

    uint stakingLP = stakingLPOf(address(pair));
    if(stakingLP > 0) IStakingRewards(uniPool[address(pair)]).exit();

    require(liquidity <= pair.balanceOf(address(this)) && liquidity > 0, 'Not enough liquidity.');

    (uint amount0, uint amount1) = IUniswapV2Router(UNISWAP_V2_ROUTER).removeLiquidity(
        token0, token1,
        liquidity,
        0, 0,
        address(this), block.timestamp
    );
    amount0 = amount0.add(_swap(token1, token0, amount1));
```

```
        //Only remove liquidity
        if(add_index >= pairs.length || add_index == remove_index) return;

        //兑换&投入
        token1 = pairs[add_index];
        amount0 = amount0 >> 1;
        amount1 = _swap(token0, token1, amount0);

        (,uint amountB,) = IUniswapV2Router(UNISWAP_V2_ROUTER).addLiquidity(
            token0, token1,
            amount0, amount1,
            0, 0,
            address(this), block.timestamp
        );

        //处理 dust. 如果有的话
        if(amount1 > amountB) _swap(token1, token0, amount1.sub(amountB));
    }
```

安全建议：无

3.18 removePair 逻辑设计【通过】

对 removePair 逻辑进行审计，检查是否有对参数进行检查以及权限进行检查，同时对业务逻辑的合理性进行审计。

审计结果：未发现相关安全风险。

```
    function removePair(uint index) external onlyController {
        require(index < pairs.length, 'Pair index out of range.');

        //撤出&兑换
        address token0 = token;
        address token1 = pairs[index];
```

```
                IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0, token1));
                _withdrawStaking(pair, totalSupply);
                uint liquidity = pair.balanceOf(address(this));

                if(liquidity > 0){
                    (uint amount0, uint amount1) = IUniswapV2Router(UNISWAP_V2_ROUTER).removeLiquidity(
                        token0, token1,
                        liquidity,
                        0, 0,
                        address(this), block.timestamp
                    );
                    amount0 = amount0.add(_swap(token1, token0, amount1));
                }
                IERC20(token1).safeApprove(UNISWAP_V2_ROUTER, 0);

                for (uint i = index; i < pairs.length-1; i++){
                    pairs[i] = pairs[i+1];
                }
                pairs.pop();
            }
        function _swap(address tokenIn, address tokenOut, uint amount)  private returns(uint){
                address pool = tokenIn == token ? curvePool[tokenOut] : curvePool[tokenIn];
                if(pool != address(0)){
                    int128 N_COINS = CURVE_N_COINS[pool];
                    int128 idxIn = N_COINS;
                    int128 idxOut = N_COINS;
                    for(int128 i=0; i<N_COINS; i++){
                        address coin = ICurve(pool).coins(uint(i));
                        if(coin == tokenIn) {idxIn = i; continue;}
                        if(coin == tokenOut) idxOut = i;
```

```
                }
                if(idxIn != N_COINS && idxOut != N_COINS){
                    uint amountBefore = IERC20(tokenOut).balanceOf(address(this));
                    ICurve(pool).exchange(idxIn, idxOut, amount, 0);
                    return (IERC20(tokenOut).balanceOf(address(this))).sub(amountBefore);
                }
            }
            address[] memory path = new address[](2);
            path[0] = tokenIn;
            path[1] = tokenOut;
            uint[] memory amounts = IUniswapV2Router(UNISWAP_V2_ROUTER).swapExactTokensForTokens(
                amount, 0, path, address(this), block.timestamp);
            return amounts[1];
        }
```

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，经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.2. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.3. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

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

安全建议：无。

4.4. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.5. require/assert 的合理使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.6. fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.7. tx.origin 身份验证【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.8. owner 权限控制【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.9. gas 消耗检测【通过】

检查 gas 的消耗是否超过区块最大限制

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.10. call 注入攻击【通过】

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

检测结果：经检测，智能合约未使用 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，但由于流动性挖矿需要增发代币，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 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；传递所有可用 gas 进行调用（可通过传入 gas_value 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继

续执行后面的代码，可能由于 Ether 发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.21. 交易顺序依赖【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

```
function deposit() public {//knownsec//  流动性挖矿
        uint _want = IERC20(want).balanceOf(address(this));
        address _controller = For(fortube).controller();
        if (_want > 0) {
            //knownsec//  由于 HBTC 合约不能设置授权额为 0
            // IERC20(want).safeApprove(_controller, 0);
            IERC20(want).safeApprove(_controller, _want);
            For(fortube).deposit(want, _want);
        }
    }
```

安全建议：无。

4.22. 时间戳依赖攻击【通过】

数据块的时间戳通常来说都是使用矿工的本地时间，而这个时间大约能有

900 秒的范围波动，当其他节点接受一个新区块时，只需要验证时间戳是否晚于之前的区块并且与本地时间误差在 900 秒以内。一个矿工可以通过设置区块的时间戳来尽可能满足有利于他的条件来从中获利。

检查合约代码实现中是否存在有依赖于时间戳的关键功能

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.23. 拒绝服务攻击【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.24. 假充值漏洞【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.25. 重入攻击检测【通过】

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

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

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

检测结果：经检测，智能合约未使用 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：

ICurve.sol:

```
pragma solidity >=0.5.0;

interface ICurve {
    function exchange(int128 i, int128 j, uint256 dx, uint256 min_dy) external;
    function coins(uint256) external view returns (address coin);
}
```

IERC20.sol:

```
pragma solidity >=0.5.0;

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

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

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

IHotPot.sol:

```
pragma solidity >=0.5.0;

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

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

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

IHotPotController.sol:

```
pragma solidity >=0.5.0;

interface IHotPotController {
    function hotpot() external view returns (address);
    function manager() external view returns (address);
    function governance() external view returns (address);
    function trustedToken(address token) external view returns (bool);

    function harvest(address token, uint amount) external returns(uint burned);

    function invest(address fund, uint amount, uint[] calldata proportions) external;
    function addPair(address fund, address token) external;
    function removePair(address fund, uint index) external;
    function reBalance(address fund, uint add_index, uint remove_index, uint liquidity) external;
    function mineUNI(address fund, address pair) external;
    function mineUNIAll(address fund) external;

    function setManager(address account) external;
    function setGovernance(address account) external;

    function setUNIPool(address fund, address pair, address uniPool) external;
    function setTrustedToken(address token, bool isTrusted) external;
    function setCurvePool(address fund, address token, address curvePool, int128 N_COINS) external;
}
```

IHotPotFund.sol

```
pragma solidity >=0.5.0;

interface IHotPotFund {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);
    event Deposit(address indexed owner, uint amount, uint share);
    event Withdraw(address indexed owner, uint amount, uint share);

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

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

    function token() external view returns (address);
    function controller() external view returns (address);
    function assets(uint index) external view returns(uint);
    function totalAssets() external view returns (uint);
    function investmentOf(address owner) external view returns (uint);

    function totalDebts() external view returns (uint);
    function debtOf(address owner) external view returns (uint256);
    function uniPool(address pair) external view returns (address);

    function pairs(uint index) external view returns (address);
    function pairsLength() external view returns(uint);
    function curvePool(address token) external view returns(address);

    function deposit(uint amount) external returns(uint share);
    function withdraw(uint share) external returns(uint amount);

    function invest(uint amount, uint[] calldata proportions) external;
    function addPair(address _token) external;
    function removePair(uint index) external;
    function reBalance(uint add_index, uint remove_index, uint liquidity) external;
    function setCurvePool(address _token, address _curvePool, int128 N_COINS) external;

    function setUNIPool(address pair, address _uniPool) external;
    function mineUNI(address pair) external;
    function mineUNIAll() external;
}
```

IHotPotFundETH.sol

```
pragma solidity >=0.5.0;

interface IHotPotFundETH {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);
    event Deposit(address indexed owner, uint amount, uint share);
    event Withdraw(address indexed owner, uint amount, uint share);

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

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

    function controller() external view returns (address);
    function assets(uint index) external view returns(uint);
    function totalAssets() external view returns (uint);
    function investmentOf(address owner) external view returns (uint);

    function totalDebts() external view returns (uint);
    function debtOf(address owner) external view returns (uint256);
    function uniPool(address pair) external view returns (address);

    function pairs(uint index) external view returns (address);
    function pairsLength() external view returns(uint);

    function deposit() external payable returns(uint share);
    function withdraw(uint share) external returns(uint amount);
```

```
        function invest(uint amount, uint[] calldata proportions) external;
        function addPair(address _token) external;
        function removePair(uint index) external;
        function reBalance(uint add_index, uint remove_index, uint liquidity) external;

        function setUNIPool(address pair, address _uniPool) external;
        function mineUNI(address pair) external;
        function mineUNIAll() external;
}
```

IStakingRewards.sol

```
pragma solidity >=0.5.0;

interface IStakingRewards {
        // Views
        function lastTimeRewardApplicable() external view returns (uint256);

        function rewardPerToken() external view returns (uint256);

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

        function getRewardForDuration() external view returns (uint256);

        function totalSupply() external view returns (uint256);

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

        // Mutative

        function stake(uint256 amount) external;

        function withdraw(uint256 amount) external;

        function getReward() external;

        function exit() external;
}
```

IUniswapV2Factory.sol

```
pragma solidity >=0.5.0;

interface IUniswapV2Factory {
        function getPair(address tokenA, address tokenB) external view returns (address pair);
}
```

IUniswapV2Pair.sol

```
pragma solidity >=0.5.0;

interface IUniswapV2Pair {
        function totalSupply() external view returns (uint);
        function balanceOf(address owner) external view returns (uint);
        function token0() external view returns (address);
        function token1() external view returns (address);

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

        function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32
blockTimestampLast);
}
```

IUniswapV2Router.sol

```
pragma solidity >=0.5.0;

interface IUniswapV2Router {
        function addLiquidity(
                address tokenA,
                address tokenB,
                uint amountADesired,
                uint amountBDesired,
                uint amountAMin,
                uint amountBMin,
                address to,
                uint deadline
        ) external returns (uint amountA, uint amountB, uint liquidity);
        function addLiquidityETH(
                address token,
                uint amountTokenDesired,
                uint amountTokenMin,
                uint amountETHMin,
                address to,
                uint deadline
        ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
```

```
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountETH);
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
        external
        payable
        returns (uint[] memory amounts);
    function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to,
uint deadline)
        external
        returns (uint[] memory amounts);
}
```

IWETH.sol

```
pragma solidity >=0.5.0;

interface IWETH {
    function deposit() external payable;
    function withdraw(uint) external;
}
```

Address.sol

```
pragma solidity >=0.5.0;

// (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/Address.sol)

library Address {
    /**
     * @dev Returns true if `account` is a contract.
     *
     * [IMPORTANT]
     * ====
     * It is unsafe to assume that an address for which this function returns
     * false is an externally-owned account (EOA) and not a contract.
     *
     * Among others, `isContract` will return false for the following
     * types of addresses:
     *
     *  - an externally-owned account
     *  - a contract in construction
     *  - an address where a contract will be created
     *  - an address where a contract lived, but was destroyed
     * ====
     */
    function isContract(address account) internal view returns (bool) {
        // This method relies in extcodesize, which returns 0 for contracts in
        // construction, since the code is only stored at the end of the
        // constructor execution.

        uint256 size;
        // solhint-disable-next-line no-inline-assembly
        assembly { size := extcodesize(account) }
        return size > 0;
    }
}
```

SafeERC20.sol

```
pragma solidity >=0.5.0;

/**
* https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/SafeERC20.sol
 */
```

```
import './SafeMath.sol';
import '../interfaces/IERC20.sol';

/**
 * @title SafeERC20
 * @dev Wrappers around ERC20 operations that throw on failure (when the token
 * contract returns false). Tokens that return no value (and instead revert or
 * throw on failure) are also supported, non-reverting calls are assumed to be
 * successful.
 * To use this library you can add a `using SafeERC20 for ERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
    using SafeMath for uint256;

    function safeTransfer(IERC20 token, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(IERC20 token, address from, address to, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.transferFrom.selector, from, to, value));
    }

    function safeApprove(IERC20 token, address spender, uint256 value) internal {
        callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    /**
     * @dev Imitates a Solidity high-level call (i.e. a regular function call to a contract), relaxing the
requirement
     * on the return value: the return value is optional (but if data is returned, it must not be false).
     * @param token The token targeted by the call.
     * @param data The call data (encoded using abi.encode or one of its variants).
     */
    function callOptionalReturn(IERC20 token, bytes memory data) private {
        // We need to perform a low level call here, to bypass Solidity's return data size checking mechanism,
since
        // we're implementing it ourselves.

        // A Solidity high level call has two parts:
        //   1. The call itself is made, and success asserted
        //   2. The return value is decoded, which in turn checks the size of the returned data.

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = address(token).call(data);
        require(success, "SafeERC20: low-level call failed");

        if (returndata.length > 0) { // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), "SafeERC20: ERC20 operation did not succeed");
        }
    }
}
```

SafeMath.sol

```
pragma solidity >=0.5.0;

// (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/math/SafeMath.sol)

library SafeMath {
    /**
     * @dev Returns the addition of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `+` operator.
     *
     * Requirements:
     *
     * - Addition cannot overflow.
     */
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
```

```
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }

    /**
     * @dev Returns the subtraction of two unsigned integers, reverting with custom message on
     * overflow (when the result is negative).
     *
     * Counterpart to Solidity's `-` operator.
     *
     * Requirements:
     *
     * - Subtraction cannot overflow.
     */
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }

    /**
     * @dev Returns the multiplication of two unsigned integers, reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        // Gas optimization: this is cheaper than requiring 'a' not being zero, but the
        // benefit is lost if 'b' is also tested.
        // See: https://github.com/OpenZeppelin/openzeppelin-contracts/pull/522
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts on
     * division by zero. The result is rounded towards zero.
     *
     * Counterpart to Solidity's `/` operator. Note: this function uses a
     * `revert` opcode (which leaves remaining gas untouched) while Solidity
     * uses an invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     *
     * - The divisor cannot be zero.
     */
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b > 0, "SafeMath: division by zero");
        uint256 c = a / b;
        // assert(a == b * c + a % b); // There is no case in which this doesn't hold

        return c;
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts when dividing by zero.
     *
     * Counterpart to Solidity's `%` operator. This function uses a `revert`
     * opcode (which leaves remaining gas untouched) while Solidity uses an
     * invalid opcode to revert (consuming all remaining gas).
     *
     * Requirements:
     * - The divisor cannot be zero.
     */
    function mod(uint256 a, uint256 b) internal pure returns (uint256) {
        require(b != 0, "SafeMath: modulo by zero");
        return a % b;
    }

    /**
     * @dev Returns the largest of two numbers.
     */
```

```
    function max(uint256 a, uint256 b) internal pure returns (uint256) {
        return a >= b ? a : b;
    }

    /**
     * @dev Returns the smallest of two numbers.
     */
    function min(uint256 a, uint256 b) internal pure returns (uint256) {
        return a < b ? a : b;
    }

    /**
     * @dev Returns the average of two numbers. The result is rounded towards
     * zero.
     */
    function average(uint256 a, uint256 b) internal pure returns (uint256) {
        // (a + b) / 2 can overflow, so we distribute
        return (a / 2) + (b / 2) + ((a % 2 + b % 2) / 2);
    }
}
```

HotPot.sol

```
pragma solidity >=0.5.0;

import './libraries/SafeMath.sol';

contract HotPot {
    using SafeMath for uint;

    string public constant name = 'Hotpot Funds';
    string public constant symbol = 'HPT';
    uint8 public constant decimals = 18;
    uint public totalSupply = 1000000e18;    // Initial supply 1 million HotPot.

    mapping(address => uint) public balanceOf;

    mapping(address => mapping(address => uint)) public allowance;

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

    constructor(address account) public {
        balanceOf[account] = totalSupply;
        emit Transfer(address(0), account, totalSupply);
    }

    function _burn(address from, uint value) internal {
        require(from != address(0), "ERC20: burn from the zero address");

        balanceOf[from] = balanceOf[from].sub(value);
        totalSupply = totalSupply.sub(value);
        emit Transfer(from, address(0), value);
    }

    function _approve(address owner, address spender, uint value) private {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        allowance[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    function _transfer(address from, address to, uint value) private {
        require(from != address(0), "ERC20: transfer from the zero address");
        require(to != address(0), "ERC20: transfer to the zero address");

        balanceOf[from] = balanceOf[from].sub(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(from, to, value);
    }

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

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

    function transferFrom(address from, address to, uint value) external returns (bool) {
        allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);
        _transfer(from, to, value);
        return true;
```

```
    }

    function burn(uint value) external returns (bool) {
        _burn(msg.sender, value);
        return true;
    }

    function burnFrom(address from, uint value) external returns (bool) {
        allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);
        _burn(from, value);
        return true;
    }
}
```

HotPotController.sol

```
pragma solidity >=0.5.0;

import './interfaces/IHotPotFund.sol';
import './interfaces/IERC20.sol';
import './interfaces/IHotPot.sol';
import './interfaces/IUniswapV2Router.sol';
import './interfaces/IUniswapV2Factory.sol';
import './libraries/SafeERC20.sol';
import './ReentrancyGuard.sol';

contract HotPotController is ReentrancyGuard {
    using SafeERC20 for IERC20;

    address constant UNISWAP_V2_ROUTER = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;
    address constant UNISWAP_FACTORY = 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;

    address public hotpot;
    address public manager;
    address public governance;
    mapping (address => bool) public trustedToken;

    event ChangeTrustedToken(address indexed token, bool isTrusted);

    modifier onlyManager {
        require(msg.sender == manager, 'Only called by Manager.');
        _;
    }

    modifier onlyGovernance{
        require(msg.sender == governance, 'Only called by Governance.');
        _;
    }

    constructor(
        address _hotpot,
        address _manager,
        address _governance
    ) public {
        hotpot = _hotpot;
        manager = _manager;
        governance = _governance;
    }

    function harvest(
        address token,
        uint amount
    ) public nonReentrant returns(uint burned) {
        uint value = amount <= IERC20(token).balanceOf(address(this)) ? amount :
IERC20(token).balanceOf(address(this));

        address pair = IUniswapV2Factory(UNISWAP_FACTORY).getPair(token, hotpot);
        require(pair != address(0), 'Pair not exist.');

        IERC20(token).safeApprove(UNISWAP_V2_ROUTER, value);
        address[] memory path = new address[](2);
        path[0] = token;
        path[1] = hotpot;
        uint[] memory amounts = IUniswapV2Router(UNISWAP_V2_ROUTER).swapExactTokensForTokens(
            value,
            0,
            path,
            address(this), block.timestamp);
        IHotPot(hotpot).burn(amounts[1]);
        return amounts[1];
    }

    function invest(address fund, uint amount, uint[] calldata proportions) external onlyManager {
        IHotPotFund(fund).invest(amount, proportions);
    }
```

```
    function addPair(address fund, address token) external onlyManager{
        require(trustedToken[token], "The token is not trusted.");
        IHotPotFund(fund).addPair(token);
    }

    function removePair(address fund, uint index)   external onlyManager {
        IHotPotFund(fund).removePair(index);
    }

    function reBalance(
        address fund,
        uint add_index,
        uint remove_index,
        uint liquidity
    ) external onlyManager {
        IHotPotFund(fund).reBalance(add_index, remove_index, liquidity);
    }

    function mineUNI(address fund, address pair) external onlyManager {
        IHotPotFund(fund).mineUNI(pair);
    }

    function mineUNIAll(address fund) external onlyManager {
        IHotPotFund(fund).mineUNIAll();
    }

    function setGovernance(address account) onlyGovernance external {
        require(account != address(0), "invalid governance address.");
        governance = account;
    }

    function setManager(address account) onlyGovernance external{
        require(account != address(0), "invalid manager address.");
        manager = account;
    }

    function setUNIPool(address fund, address pair, address uniPool) external onlyGovernance {
        IHotPotFund(fund).setUNIPool(pair, uniPool);
    }

    function setTrustedToken(address token, bool isTrusted) external onlyGovernance {
        trustedToken[token] = isTrusted;
        emit ChangeTrustedToken(token, isTrusted);
    }

    function setCurvePool(address fund, address token, address curvePool, int128 N_COINS) external
onlyGovernance {
        IHotPotFund(fund).setCurvePool(token, curvePool, N_COINS);
    }
}
```

HotPotFund.sol

```
pragma solidity >=0.5.0;

import './interfaces/IERC20.sol';
import './interfaces/IUniswapV2Factory.sol';
import './interfaces/IUniswapV2Router.sol';
import './interfaces/IUniswapV2Pair.sol';
import './interfaces/IStakingRewards.sol';
import './interfaces/ICurve.sol';
import './libraries/SafeMath.sol';
import './libraries/SafeERC20.sol';
import './HotPotFundERC20.sol';
import './ReentrancyGuard.sol';

contract HotPotFund is ReentrancyGuard, HotPotFundERC20 {
    using SafeMath for uint;
    using SafeERC20 for IERC20;

    address constant UNISWAP_FACTORY = 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;
    address constant UNISWAP_V2_ROUTER = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;
    address constant UNI = 0x4047fbe32235E74C1264567D80D28d44A50c1dDb;

    uint constant DIVISOR = 100;
    uint constant FEE = 20;

    address public token;
    address public controller;
    uint public totalInvestment;
    mapping (address => uint) public investmentOf;

    // UNI mining rewards
    uint public totalDebts;
    mapping(address => uint256) public debtOf;
    // UNI mining pool pair->mining pool
```

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

    address[] public pairs;

    //Curve swap pools
    mapping (address => address) public curvePool;
    mapping (address => int128) CURVE_N_COINS;

    modifier onlyController() {
        require(msg.sender == controller, 'Only called by Controller.');
        _;
    }

    event Deposit(address indexed owner, uint amount, uint share);
    event Withdraw(address indexed owner, uint amount, uint share);

    constructor (address _token, address _controller) public {
        //approve for add liquidity and swap. 2**256-1 never used up.
        IERC20(_token).safeApprove(UNISWAP_V2_ROUTER, 2**256-1);

        token = _token;
        controller = _controller;
    }

    function deposit(uint amount) public nonReentrant returns(uint share) {
        require(amount > 0, 'Are you kidding me?');
        uint _total_assets = totalAssets();
        IERC20(token).safeTransferFrom(msg.sender, address(this), amount);

        if(totalSupply == 0){
            share = amount;
        }
        else{
            share = amount.mul(totalSupply).div(_total_assets);
            // user uni debt
            uint debt = share.mul(totalDebts.add(totalUNIRewards())).div(totalSupply);
            if(debt > 0){
                debtOf[msg.sender] = debtOf[msg.sender].add(debt);
                totalDebts = totalDebts.add(debt);
            }
        }

        investmentOf[msg.sender] = investmentOf[msg.sender].add(amount);
        totalInvestment = totalInvestment.add(amount);
        _mint(msg.sender, share);
        emit Deposit(msg.sender, amount, share);
    }

    /**
     * @notice 按照基金设定比例投资流动池，统一操作可以节省用户 gas 消耗.
     * 当合约中还未投入流动池的资金额度较大时，一次性投入会产生较大滑点，可能要分批操作，所以投资行为必须由基金统一操作.
     */
    function invest(uint amount, uint[] calldata proportions) external onlyController {
        uint len = pairs.length;
        require(len>0, 'Pairs is empty.');
        address token0 = token;
        require(amount <= IERC20(token0).balanceOf(address(this)), "Not enough balance.");
        require(proportions.length == pairs.length, 'Proportions index out of range.');

        uint _whole;
        for(uint i=0; i<len; i++){
            if(proportions[i] == 0) continue;
            _whole = _whole.add(proportions[i]);

            uint amount0 = (amount.mul(proportions[i]).div(DIVISOR)) >> 1;
            if(amount0 == 0) continue;

            address token1 = pairs[i];
            uint amount1 = _swap(token0, token1, amount0);

            (,uint amountB,) = IUniswapV2Router(UNISWAP_V2_ROUTER).addLiquidity(
                token0, token1,
                amount0, amount1,
                0, 0,
                address(this), block.timestamp
            );

            if(amount1 > amountB) _swap(token1, token0, amount1.sub(amountB));
        }
        require(_whole == DIVISOR, 'Error proportion.');
    }

    function setUNIPool(address pair, address _uniPool) external onlyController {
        require(pair!= address(0) && _uniPool!= address(0), "Invalid address.");

        if(uniPool[pair] != address(0)){
```

```
            _withdrawStaking(IUniswapV2Pair(pair), totalSupply);
        }
        IERC20(pair).approve(_uniPool, 2**256-1);
        uniPool[pair] = _uniPool;
    }

    function mineUNI(address pair) public onlyController {
        address stakingRewardAddr = uniPool[pair];
        if(stakingRewardAddr != address(0)){
            uint liquidity = IUniswapV2Pair(pair).balanceOf(address(this));
            if(liquidity > 0){
                IStakingRewards(stakingRewardAddr).stake(liquidity);
            }
        }
    }

    function mineUNIAll() external onlyController {
        for(uint i = 0; i < pairs.length; i++){
            IUniswapV2Pair pair =
IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token, pairs[i]));
            address stakingRewardAddr = uniPool[address(pair)];
            if(stakingRewardAddr != address(0)){
                uint liquidity = pair.balanceOf(address(this));
                if(liquidity > 0){
                    IStakingRewards(stakingRewardAddr).stake(liquidity);
                }
            }
        }
    }

    function totalUNIRewards() public view returns(uint amount){
        amount = IERC20(UNI).balanceOf(address(this));
        for(uint i = 0; i < pairs.length; i++){
            IUniswapV2Pair pair =
IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token, pairs[i]));
            address stakingRewardAddr = uniPool[address(pair)];
            if(stakingRewardAddr != address(0)){
                amount = amount.add(IStakingRewards(stakingRewardAddr).earned(address(this)));
            }
        }
    }

    function UNIRewardsOf(address account) public view returns(uint reward){
        if(balanceOf[account] > 0){
            uint uniAmount = totalUNIRewards();
            uint totalAmount = totalDebts.add(uniAmount).mul(balanceOf[account]).div(totalSupply);
            reward = totalAmount.sub(debtOf[account]);
        }
    }

    function stakingLPOf(address pair) public view returns(uint liquidity){
        if(uniPool[pair] != address(0)){
            liquidity = IStakingRewards(uniPool[pair]).balanceOf(address(this));
        }
    }

    function _withdrawStaking(IUniswapV2Pair pair, uint share) internal returns(uint liquidity){
        address stakingRewardAddr = uniPool[address(pair)];
        if(stakingRewardAddr != address(0)){
            liquidity =
IStakingRewards(stakingRewardAddr).balanceOf(address(this)).mul(share).div(totalSupply);
            if(liquidity > 0){
                IStakingRewards(stakingRewardAddr).withdraw(liquidity);
                IStakingRewards(stakingRewardAddr).getReward();
            }
        }
    }

    function withdraw(uint share) public nonReentrant returns(uint amount) {
        require(share > 0 && share <= balanceOf[msg.sender], 'Not enough balance.');

        uint _investment;
        (amount, _investment) = _withdraw(msg.sender, share);
        investmentOf[msg.sender] = investmentOf[msg.sender].sub(_investment);
        totalInvestment = totalInvestment.sub(_investment);
        _burn(msg.sender, share);
        IERC20(token).safeTransfer(msg.sender, amount);
        emit Withdraw(msg.sender, amount, share);
    }

    function _withdraw(
        address user,
        uint share
    ) internal returns (uint amount, uint investment) {
        address token0 = token;
        amount = IERC20(token0).balanceOf(address(this)).mul(share).div(totalSupply);
        for(uint i = 0; i < pairs.length; i++) {
```

```
                address token1 = pairs[i];
                IUniswapV2Pair                                          pair                                    =
IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0, token1));
                uint liquidity = pair.balanceOf(address(this)).mul(share).div(totalSupply);
                liquidity    = liquidity.add(_withdrawStaking(pair, share));
                if(liquidity == 0) continue;

                (uint amount0, uint amount1) = IUniswapV2Router(UNISWAP_V2_ROUTER).removeLiquidity(
                    token0, token1,
                    liquidity,
                    0, 0,
                    address(this), block.timestamp
                );
                amount = amount.add(amount0).add(_swap(token1, token0, amount1));
            }

            //withdraw UNI reward
            uint uniAmount = IERC20(UNI).balanceOf(address(this));
            uint totalAmount = totalDebts.add(uniAmount).mul(share).div(totalSupply);
            if(totalAmount > 0){
                uint debt = debtOf[user].mul(share).div(balanceOf[user]);
                debtOf[user] = debtOf[user].sub(debt);
                totalDebts = totalDebts.sub(debt);
                uint reward = totalAmount.sub(debt);
                if(reward > uniAmount) reward = uniAmount;
                if(reward > 0) IERC20(UNI).transfer(user, reward);
            }

            investment = investmentOf[user].mul(share).div(balanceOf[user]);
            if(amount > investment){
                uint _fee = (amount.sub(investment)).mul(FEE).div(DIVISOR);
                amount = amount.sub(_fee);
                IERC20(token0).safeTransfer(controller, _fee);
            }
            else {
                investment = amount;
            }
        }

        function assets(uint index) public view returns(uint _assets) {
            require(index < pairs.length, 'Pair index out of range.');
            address token0 = token;
            address token1 = pairs[index];
            IUniswapV2Pair  pair  =  IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0,
token1));
            (uint reserve0, uint reserve1, ) = pair.getReserves();

            uint liquidity = pair.balanceOf(address(this)).add(stakingLPOf(address(pair)));
            if( pair.token0() == token0 )
                _assets = (reserve0 << 1).mul(liquidity).div(pair.totalSupply());
            else // pair.token1() == token0
                _assets = (reserve1 << 1).mul(liquidity).div(pair.totalSupply());
        }

        function totalAssets() public view returns(uint _assets) {
            address token0 = token;
            for(uint i=0; i<pairs.length; i++){
                address token1 = pairs[i];
                IUniswapV2Pair                                          pair                                    =
IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0, token1));
                (uint reserve0, uint reserve1, ) = pair.getReserves();
                uint liquidity = pair.balanceOf(address(this)).add(stakingLPOf(address(pair)));
                if( pair.token0() == token0 )
                    _assets = _assets.add((reserve0 << 1).mul(liquidity).div(pair.totalSupply()));
                else // pair.token1() == token0
                    _assets = _assets.add((reserve1 << 1).mul(liquidity).div(pair.totalSupply()));
            }
            _assets = _assets.add(IERC20(token0).balanceOf(address(this)));
        }

        function pairsLength() public view returns(uint) {
            return pairs.length;
        }

        function setCurvePool(address _token, address _curvePool, int128 N_COINS) external onlyController {
            curvePool[_token] = _curvePool;
            if(_curvePool != address(0)) {
                if(IERC20(token).allowance(address(this), _curvePool) == 0){
                    IERC20(token).safeApprove(_curvePool, 2**256-1);
                }
                if(IERC20(_token).allowance(address(this), _curvePool) == 0){
                    IERC20(_token).safeApprove(_curvePool, 2**256-1);
                }
                CURVE_N_COINS[_curvePool] = N_COINS;
            }
        }
```

```
    /**
     * @notice 添加流动池.
     */
    function addPair(address _token) external onlyController {
        address pair = IUniswapV2Factory(UNISWAP_FACTORY).getPair(token, _token);
        require(pair != address(0), 'Pair not exist.');

        //approve for add liquidity and swap.
        IERC20(_token).safeApprove(UNISWAP_V2_ROUTER, 2**256-1);
        //approve for remove liquidity
        IUniswapV2Pair(pair).approve(UNISWAP_V2_ROUTER, 2**256-1);

        for(uint i = 0; i < pairs.length; i++) {
            require(pairs[i] != _token, 'Pair existed.');
        }
        pairs.push(_token);
    }

    /**
     * @notice 调整已投入的流动池.
     * 在调整流动池时，如果金额较大，请多付几笔 gas 费用，分次调整，尽量降低滑点.
     */
    function reBalance(
        uint add_index,
        uint remove_index,
        uint liquidity
    ) external onlyController {
        require(remove_index < pairs.length, 'Pair index out of range.');

        //撤出&兑换
        address token0 = token;
        address token1 = pairs[remove_index];
        IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0,
token1));

        uint stakingLP = stakingLPOf(address(pair));
        if(stakingLP > 0) IStakingRewards(uniPool[address(pair)]).exit();

        require(liquidity <= pair.balanceOf(address(this)) && liquidity > 0, 'Not enough liquidity.');

        (uint amount0, uint amount1) = IUniswapV2Router(UNISWAP_V2_ROUTER).removeLiquidity(
            token0, token1,
            liquidity,
            0, 0,
            address(this), block.timestamp
        );
        amount0 = amount0.add(_swap(token1, token0, amount1));

        //Only remove liquidity
        if(add_index >= pairs.length || add_index == remove_index) return;

        //兑换&投入
        token1 = pairs[add_index];
        amount0 = amount0 >> 1;
        amount1 = _swap(token0, token1, amount0);

        (,uint amountB,) = IUniswapV2Router(UNISWAP_V2_ROUTER).addLiquidity(
            token0, token1,
            amount0, amount1,
            0, 0,
            address(this), block.timestamp
        );

        //处理 dust. 如果有的话
        if(amount1 > amountB) _swap(token1, token0, amount1.sub(amountB));
    }

    /**
     * @notice 移除流动池.
     */
    function removePair(uint index) external onlyController {
        require(index < pairs.length, 'Pair index out of range.');

        //撤出&兑换
        address token0 = token;
        address token1 = pairs[index];
        IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0,
token1));
        _withdrawStaking(pair, totalSupply);
        uint liquidity = pair.balanceOf(address(this));

        if(liquidity > 0){
            (uint amount0, uint amount1) = IUniswapV2Router(UNISWAP_V2_ROUTER).removeLiquidity(
                token0, token1,
                liquidity,
                0, 0,
                address(this), block.timestamp
```

```
                );
                amount0 = amount0.add(_swap(token1, token0, amount1));
            }
            IERC20(token1).safeApprove(UNISWAP_V2_ROUTER, 0);

            for (uint i = index; i < pairs.length-1; i++){
                pairs[i] = pairs[i+1];
            }
            pairs.pop();
        }

        function _swap(address tokenIn, address tokenOut, uint amount)    private returns(uint) {
            address pool = tokenIn == token ? curvePool[tokenOut] : curvePool[tokenIn];
            if(pool != address(0)){
                int128 N_COINS = CURVE_N_COINS[pool];
                int128 idxIn = N_COINS;
                int128 idxOut = N_COINS;
                for(int128 i=0; i<N_COINS; i++){
                    address coin = ICurve(pool).coins(uint(i));
                    if(coin == tokenIn) {idxIn = i; continue;}
                    if(coin == tokenOut) idxOut = i;
                }
                if(idxIn != N_COINS && idxOut != N_COINS){
                    uint amountBefore = IERC20(tokenOut).balanceOf(address(this));
                    ICurve(pool).exchange(idxIn, idxOut, amount, 0);
                    return (IERC20(tokenOut).balanceOf(address(this))).sub(amountBefore);
                }
            }
            address[] memory path = new address[](2);
            path[0] = tokenIn;
            path[1] = tokenOut;
            uint[] memory amounts = IUniswapV2Router(UNISWAP_V2_ROUTER).swapExactTokensForTokens(
                amount, 0, path, address(this), block.timestamp);
            return amounts[1];
        }
}
```

HotPotFundERC20.sol

```
pragma solidity >=0.5.0;

import './libraries/SafeMath.sol';

contract HotPotFundERC20 {
    using SafeMath for uint;

    string public constant name = 'Hotpot V1';
    string public constant symbol = 'HPT-V1';
    uint8 public constant decimals = 18;
    uint public totalSupply;
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint)) public allowance;

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

    constructor() public {
    }

    function _mint(address to, uint value) internal {
        require(to != address(0), "ERC20: mint to the zero address");

        totalSupply = totalSupply.add(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(address(0), to, value);
    }

    function _burn(address from, uint value) internal {
        require(from != address(0), "ERC20: burn from the zero address");

        balanceOf[from] = balanceOf[from].sub(value);
        totalSupply = totalSupply.sub(value);
        emit Transfer(from, address(0), value);
    }

    function _approve(address owner, address spender, uint value) private {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        allowance[owner][spender] = value;
        emit Approval(owner, spender, value);
    }

    function _transfer(address from, address to, uint value) private {
        require(from != address(0), "ERC20: transfer from the zero address");
        require(to != address(0), "ERC20: transfer to the zero address");
```

```
            balanceOf[from] = balanceOf[from].sub(value);
            balanceOf[to] = balanceOf[to].add(value);
            emit Transfer(from, to, value);
        }

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

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

        function transferFrom(address from, address to, uint value) external returns (bool) {
            allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);
            _transfer(from, to, value);
            return true;
        }
}
```

HotPotFundETH.sol

```
pragma solidity >=0.5.0;

import './interfaces/IERC20.sol';
import './interfaces/IUniswapV2Factory.sol';
import './interfaces/IUniswapV2Router.sol';
import './interfaces/IUniswapV2Pair.sol';
import './interfaces/IStakingRewards.sol';
import './interfaces/IWETH.sol';
import './libraries/SafeMath.sol';
import './libraries/SafeERC20.sol';
import './HotPotFundERC20.sol';
import './ReentrancyGuard.sol';

contract HotPotFundETH is ReentrancyGuard, HotPotFundERC20 {
    using SafeMath for uint;
    using SafeERC20 for IERC20;

    address constant UNISWAP_FACTORY = 0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f;
    address constant UNISWAP_V2_ROUTER = 0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D;
    address constant WETH = 0xc778417E063141139Fce010982780140Aa0cD5Ab;
    address constant UNI = 0x4047fbe32235E74C1264567D80D28d44A50c1dDb;

    uint constant DIVISOR = 100;
    uint constant FEE = 20;

    address public controller;
    uint public totalInvestment;
    mapping (address => uint) public investmentOf;

    // UNI mining rewards
    uint public totalDebts;
    mapping(address => uint256) public debtOf;
    //UNI mining pool pair->mining pool
    mapping(address => address) public uniPool;

    address[] public pairs;

    modifier onlyController() {
        require(msg.sender == controller, 'Only called by Controller.');
        _;
    }

    event Deposit(address indexed owner, uint amount, uint share);
    event Withdraw(address indexed owner, uint amount, uint share);

    constructor (address _controller) public {
        //approve for add liquidity and swap. 2**256-1 never used up.
        IERC20(WETH).approve(UNISWAP_V2_ROUTER, 2**256-1);

        controller = _controller;
    }

    function() external payable {
        // 接受 WETH 转账.
        if(msg.sender != WETH) deposit();
    }

    function deposit() public nonReentrant payable returns(uint share) {
        require(msg.value > 0, 'Are you kidding me?');
        uint amount = msg.value;
        uint _total_assets = totalAssets();
        IWETH(WETH).deposit.value(amount)();
```

```
        if(totalSupply == 0){
            share = amount;
        }
        else{
            share = amount.mul(totalSupply).div(_total_assets);
            //user uni debt
            uint debt = share.mul(totalDebts.add(totalUNIRewards())).div(totalSupply);
            if(debt > 0){
                debtOf[msg.sender] = debtOf[msg.sender].add(debt);
                totalDebts = totalDebts.add(debt);
            }
        }

        investmentOf[msg.sender] = investmentOf[msg.sender].add(amount);
        totalInvestment = totalInvestment.add(amount);
        _mint(msg.sender, share);
        emit Deposit(msg.sender, amount, share);
    }

    /**
     * @notice 按照基金设定比例投资流动池，统一操作可以节省用户 gas 消耗.
     * 当合约中还未投入流动池的资金额度较大时，一次性投入会产生较大滑点，可能要分批操作，所以投资行为必须由基金统一操作.
     */
    function invest(uint amount, uint[] calldata proportions) external onlyController {
        uint len = pairs.length;
        require(len > 0, 'Pairs is empty.');
        address token0 = WETH;
        require(amount <= IERC20(token0).balanceOf(address(this)), "Not enough balance.");
        require(proportions.length == pairs.length, 'Proportions index out of range.');

        uint _whole;
        for(uint i=0; i<len; i++){
            if(proportions[i] == 0) continue;
            _whole = _whole.add(proportions[i]);

            uint amount0 = (amount.mul(proportions[i]).div(DIVISOR)) >> 1;
            if(amount0 == 0) continue;

            address token1 = pairs[i];
            uint amount1 = _swap(token0, token1, amount0);
            (,uint amountB,) = IUniswapV2Router(UNISWAP_V2_ROUTER).addLiquidity(
                token0, token1,
                amount0, amount1,
                0, 0,
                address(this), block.timestamp
            );

            if(amount1 > amountB) _swap(token1, token0, amount1.sub(amountB));
        }
        require(_whole == DIVISOR, 'Error proportion.');
    }

    function setUNIPool(address pair, address _uniPool) external onlyController {
        require(pair!= address(0) && _uniPool!= address(0), "Invalid address.");
        if(uniPool[pair] != address(0)){
            _withdrawStaking(IUniswapV2Pair(pair), totalSupply);
        }
        IERC20(pair).approve(_uniPool, 2**256-1);
        uniPool[pair] = _uniPool;
    }

    function mineUNI(address pair) public onlyController {
        address stakingRewardAddr = uniPool[pair];
        if(stakingRewardAddr != address(0)){
            uint liquidity = IUniswapV2Pair(pair).balanceOf(address(this));
            if(liquidity > 0){
                IStakingRewards(stakingRewardAddr).stake(liquidity);
            }
        }
    }

    function mineUNIAll() external onlyController {
        for(uint i = 0; i < pairs.length; i++){
            IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(WETH, pairs[i]));
            address stakingRewardAddr = uniPool[address(pair)];
            if(stakingRewardAddr != address(0)){
                uint liquidity = pair.balanceOf(address(this));
                if(liquidity > 0){
                    IStakingRewards(stakingRewardAddr).stake(liquidity);
                }
            }
        }
    }
```

```
    function totalUNIRewards() public view returns(uint amount){
        amount = IERC20(UNI).balanceOf(address(this));
        for(uint i = 0; i < pairs.length; i++){
            IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(WETH, pairs[i]));
            address stakingRewardAddr = uniPool[address(pair)];
            if(stakingRewardAddr != address(0)){
                amount = amount.add(IStakingRewards(stakingRewardAddr).earned(address(this)));
            }
        }
    }

    function UNIRewardsOf(address account) public view returns(uint reward){
        if(balanceOf[account] > 0){
            uint uniAmount = totalUNIRewards();
            uint totalAmount = totalDebts.add(uniAmount).mul(balanceOf[account]).div(totalSupply);
            reward = totalAmount.sub(debtOf[account]);
        }
    }

    function stakingLPOf(address pair) public view returns(uint liquidity){
        if(uniPool[pair] != address(0)){
            liquidity = IStakingRewards(uniPool[pair]).balanceOf(address(this));
        }
    }

    function _withdrawStaking(IUniswapV2Pair pair, uint share) internal returns(uint liquidity){
        address stakingRewardAddr = uniPool[address(pair)];
        if(stakingRewardAddr != address(0)){
            liquidity = IStakingRewards(stakingRewardAddr).balanceOf(address(this)).mul(share).div(totalSupply);
            if(liquidity > 0){
                IStakingRewards(stakingRewardAddr).withdraw(liquidity);
                IStakingRewards(stakingRewardAddr).getReward();
            }
        }
    }

    function withdraw(uint share) public nonReentrant returns(uint amount) {
        require(share > 0 && share <= balanceOf[msg.sender], 'Not enough balance.');

        uint _investment;
        (amount, _investment) = _withdraw(msg.sender, share);
        _burn(msg.sender, share);
        investmentOf[msg.sender] = investmentOf[msg.sender].sub(_investment);
        totalInvestment = totalInvestment.sub(_investment);
        IWETH(WETH).withdraw(amount);
        msg.sender.transfer(amount);
        emit Withdraw(msg.sender, amount, share);
    }

    function _withdraw(
        address user,
        uint share
    ) internal returns (uint amount, uint investment) {
        address token0 = WETH;
        amount = IERC20(token0).balanceOf(address(this)).mul(share).div(totalSupply);
        for(uint i = 0; i < pairs.length; i++) {
            address token1 = pairs[i];
            IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0, token1));
            uint liquidity = pair.balanceOf(address(this)).mul(share).div(totalSupply);
            liquidity    = liquidity.add(_withdrawStaking(pair, share));

            if(liquidity > 0){
                (uint amount0, uint amount1) = IUniswapV2Router(UNISWAP_V2_ROUTER).removeLiquidity(
                    token0, token1,
                    liquidity,
                    0, 0,
                    address(this), block.timestamp
                );
                amount = amount.add(amount0).add(_swap(token1, token0, amount1));
            }
        }

        //withdraw UNI reward
        uint uniAmount = IERC20(UNI).balanceOf(address(this));
        uint totalAmount = totalDebts.add(uniAmount).mul(share).div(totalSupply);
        if(totalAmount > 0){
            uint debt = debtOf[user].mul(share).div(balanceOf[user]);
            debtOf[user] = debtOf[user].sub(debt);
            totalDebts = totalDebts.sub(debt);
            uint reward = totalAmount.sub(debt);
            if(reward > uniAmount) reward = uniAmount;
            if(reward > 0) IERC20(UNI).transfer(user, reward);
        }
```

```
            investment = investmentOf[user].mul(share).div(balanceOf[user]);
            if(amount > investment){
                uint _fee = (amount.sub(investment)).mul(FEE).div(DIVISOR);
                amount = amount.sub(_fee);
                //给 controller 转的是 WETH.
                IERC20(token0).safeTransfer(controller, _fee);
            }
            else {
                investment = amount;
            }
        }

        function assets(uint index) public view returns(uint _assets) {
            require(index < pairs.length, 'Pair index out of range.');
            address token0 = WETH;
            address token1 = pairs[index];
            IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0,
token1));
            (uint reserve0, uint reserve1, ) = pair.getReserves();

            uint liquidity = pair.balanceOf(address(this)).add(stakingLPOf(address(pair)));
            if( pair.token0() == token0 )
                _assets = (reserve0 << 1).mul(liquidity).div(pair.totalSupply());
            else // pair.token1() == token0
                _assets = (reserve1 << 1).mul(liquidity).div(pair.totalSupply());
        }

        function totalAssets() public view returns(uint _assets) {
            address token0 = WETH;
            for(uint i=0; i<pairs.length; i++){
                address token1 = pairs[i];
                IUniswapV2Pair                                          pair                                          =
IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0, token1));
                (uint reserve0, uint reserve1, ) = pair.getReserves();
                uint liquidity = pair.balanceOf(address(this)).add(stakingLPOf(address(pair)));
                if( pair.token0() == token0 )
                    _assets = _assets.add((reserve0 << 1).mul(liquidity).div(pair.totalSupply()));
                else // pair.token1() == token0
                    _assets = _assets.add((reserve1 << 1).mul(liquidity).div(pair.totalSupply()));
            }
            _assets = _assets.add(IERC20(token0).balanceOf(address(this)));
        }

        function pairsLength() public view returns(uint) {
            return pairs.length;
        }

        /**
        * @notice 添加流动池.
        */
        function addPair(address _token) external onlyController {
            address pair = IUniswapV2Factory(UNISWAP_FACTORY).getPair(WETH, _token);
            require(pair != address(0), 'Pair not exist.');

            //approve for add liquidity and swap
            IERC20(_token).safeApprove(UNISWAP_V2_ROUTER, 2**256-1);
            //approve for remove liquidity
            IUniswapV2Pair(pair).approve(UNISWAP_V2_ROUTER, 2**256-1);

            for(uint i = 0; i < pairs.length; i++) {
                require(pairs[i] != _token, 'Pair existed.');
            }
            pairs.push(_token);
        }

        /**
        * @notice 调整已投入的流动池.
        * 在调整流动池时，如果金额较大，请多付几笔 gas 费用，分次调整，尽量降低滑点.
         */
        function reBalance(
            uint add_index,
            uint remove_index,
            uint liquidity
        ) external onlyController {
            require(remove_index < pairs.length, 'Pair index out of range.');

            //撤出&兑换
            address token0 = WETH;
            address token1 = pairs[remove_index];
            IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0,
token1));

            uint stakingLP = stakingLPOf(address(pair));
            if(stakingLP > 0) IStakingRewards(uniPool[address(pair)]).exit();

            require(liquidity <= pair.balanceOf(address(this)), 'Not enough liquidity.');
```

```
            (uint amount0, uint amount1) = IUniswapV2Router(UNISWAP_V2_ROUTER).removeLiquidity(
                token0, token1,
                liquidity,
                0, 0,
                address(this), block.timestamp
            );
            amount0 = amount0.add(_swap(token1, token0, amount1));

            //Only remove liquidity
            if(add_index >= pairs.length || add_index == remove_index) return;

            //兑换&投入
            token1 = pairs[add_index];
            amount0 = amount0 >> 1;
            amount1 = _swap(token0, token1, amount0);

            (,uint amountB,) = IUniswapV2Router(UNISWAP_V2_ROUTER).addLiquidity(
                token0, token1,
                amount0, amount1,
                0, 0,
                address(this), block.timestamp
            );

            //处理 dust. 如果有的话
            if(amount1 > amountB) _swap(token1, token0, amount1.sub(amountB));
        }

        /**
         * @notice 移除流动池.
         */
        function removePair(uint index) external onlyController {
            require(index < pairs.length, 'Pair index out of range.');

            //撤出&兑换
            address token0 = WETH;
            address token1 = pairs[index];
            IUniswapV2Pair pair = IUniswapV2Pair(IUniswapV2Factory(UNISWAP_FACTORY).getPair(token0,
token1));
            _withdrawStaking(pair, totalSupply);
            uint liquidity = pair.balanceOf(address(this));
            if(liquidity > 0){
                (uint amount0, uint amount1) = IUniswapV2Router(UNISWAP_V2_ROUTER).removeLiquidity(
                    token0, token1,
                    liquidity,
                    0, 0,
                    address(this), block.timestamp
                );
                amount0 = amount0.add(_swap(token1, token0, amount1));
            }
            IERC20(token1).safeApprove(UNISWAP_V2_ROUTER, 0);

            for (uint i = index; i<pairs.length-1; i++){
                pairs[i] = pairs[i+1];
            }
            pairs.pop();
        }

        function _swap(address tokenIn, address tokenOut, uint amount) private returns(uint) {
            address[] memory path = new address[](2);
            path[0] = tokenIn;
            path[1] = tokenOut;
            uint[] memory amounts = IUniswapV2Router(UNISWAP_V2_ROUTER).swapExactTokensForTokens(
                amount,
                0,
                path,
                address(this), block.timestamp);
            return amounts[1];
        }
}
```

ReentrancyGuardl.sol

```
pragma solidity >=0.5.0;

// (https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/utils/ReentrancyGuard.sol)
contract ReentrancyGuard {
    // Booleans are more expensive than uint256 or any type that takes up a full
    // word because each write operation emits an extra SLOAD to first read the
    // slot's contents, replace the bits taken up by the boolean, and then write
    // back. This is the compiler's defense against contract upgrades and
    // pointer aliasing, and it cannot be disabled.

    // The values being non-zero value makes deployment a bit more expensive,
    // but in exchange the refund on every call to nonReentrant will be lower in
```

```
    // amount. Since refunds are capped to a percentage of the total
    // transaction's gas, it is best to keep them low in cases like this one, to
    // increase the likelihood of the full refund coming into effect.
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    constructor () internal {
        _status = _NOT_ENTERED;
    }

    /**
     * @dev Prevents a contract from calling itself, directly or indirectly.
     * Calling a `nonReentrant` function from another `nonReentrant`
     * function is not supported. It is possible to prevent this from happening
     * by making the `nonReentrant` function external, and make it call a
     * `private` function that does the actual work.
     */
    modifier nonReentrant() {
        // On the first call to nonReentrant, _notEntered will be true
        require(_status != _ENTERED, "ReentrancyGuard: reentrant call");

        // Any calls to nonReentrant after this point will fail
        _status = _ENTERED;

        _;

        // By storing the original value once again, a refund is triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _status = _NOT_ENTERED;
    }
}
```

StakingRewards.sol

```
/**
 * https://github.com/Synthetixio/synthetix/blob/develop/contracts/StakingRewards.sol
*/

pragma solidity ^0.5.16;

import './interfaces/IERC20.sol';
import './interfaces/IStakingRewards.sol';
import './libraries/SafeMath.sol';
import './libraries/SafeERC20.sol';
import './libraries/Address.sol';
import './ReentrancyGuard.sol';

contract RewardsDistributionRecipient {
    address public rewardsDistribution;

    function notifyRewardAmount(uint256 reward) external;

    modifier onlyRewardsDistribution() {
        require(msg.sender == rewardsDistribution, "Caller is not RewardsDistribution contract");
        _;
    }
}

contract StakingRewards is IStakingRewards, RewardsDistributionRecipient, ReentrancyGuard {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

    /* ========== STATE VARIABLES ========== */

    address public rewardsToken;
    address public stakingToken;
    uint256 public periodFinish = 0;
    uint256 public rewardRate = 0;
    uint256 public rewardsDuration = 100 days;
    uint256 public lastUpdateTime;
    uint256 public rewardPerTokenStored;

    mapping(address => uint256) public userRewardPerTokenPaid;
    mapping(address => uint256) public rewards;

    uint256 private _totalSupply;
    mapping(address => uint256) private _balances;

    /* ========== CONSTRUCTOR ========== */

    constructor(
        address _rewardsDistribution,
        address _rewardsToken,
        address _stakingToken
```

```
    ) public {
        rewardsToken = _rewardsToken;
        stakingToken = _stakingToken;
        rewardsDistribution = _rewardsDistribution;
    }

    /* ========== VIEWS ========== */

    function totalSupply() external view returns (uint256) {
        return _totalSupply;
    }

    function balanceOf(address account) external view returns (uint256) {
        return _balances[account];
    }

    function lastTimeRewardApplicable() public view returns (uint256) {
        return SafeMath.min(block.timestamp, periodFinish);
    }

    function rewardPerToken() public view returns (uint256) {
        if (_totalSupply == 0) {
            return rewardPerTokenStored;
        }
        return
            rewardPerTokenStored.add(
lastTimeRewardApplicable().sub(lastUpdateTime).mul(rewardRate).mul(1e18).div(_totalSupply)
            );
    }

    function earned(address account) public view returns (uint256) {
        return
_balances[account].mul(rewardPerToken().sub(userRewardPerTokenPaid[account])).div(1e18).add(rewards[acco
unt]);
    }

    function getRewardForDuration() external view returns (uint256) {
        return rewardRate.mul(rewardsDuration);
    }

    /* ========== MUTATIVE FUNCTIONS ========== */

    function stake(uint256 amount) external nonReentrant updateReward(msg.sender) {
        require(amount > 0, "Cannot stake 0");
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);
        IERC20(stakingToken).safeTransferFrom(msg.sender, address(this), amount);
        emit Staked(msg.sender, amount);
    }

    function withdraw(uint256 amount) public nonReentrant updateReward(msg.sender) {
        require(amount > 0, "Cannot withdraw 0");
        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);
        IERC20(stakingToken).safeTransfer(msg.sender, amount);
        emit Withdrawn(msg.sender, amount);
    }

    function getReward() public nonReentrant updateReward(msg.sender) {
        uint256 reward = rewards[msg.sender];
        if (reward > 0) {
            rewards[msg.sender] = 0;
            IERC20(rewardsToken).safeTransfer(msg.sender, reward);
            emit RewardPaid(msg.sender, reward);
        }
    }

    function exit() external {
        withdraw(_balances[msg.sender]);
        getReward();
    }

    /* ========== RESTRICTED FUNCTIONS ========== */

    function notifyRewardAmount(uint256 reward) external onlyRewardsDistribution updateReward(address(0))
{
        if (block.timestamp >= periodFinish) {
            rewardRate = reward.div(rewardsDuration);
        } else {
            uint256 remaining = periodFinish.sub(block.timestamp);
            uint256 leftover = remaining.mul(rewardRate);
            rewardRate = reward.add(leftover).div(rewardsDuration);
        }

        // Ensure the provided reward amount is not more than the balance in the contract.
        // This keeps the reward rate in the right range, preventing overflows due to
```

```
        // very high values of rewardRate in the earned and rewardsPerToken functions;
        // Reward + leftover must be less than 2^256 / 10^18 to avoid overflow.
        uint balance = IERC20(rewardsToken).balanceOf(address(this));
        require(rewardRate <= balance.div(rewardsDuration), "Provided reward too high");

        lastUpdateTime = block.timestamp;
        periodFinish = block.timestamp.add(rewardsDuration);
        emit RewardAdded(reward);
    }

    /* ========== MODIFIERS ========== */

    modifier updateReward(address account) {
        rewardPerTokenStored = rewardPerToken();
        lastUpdateTime = lastTimeRewardApplicable();
        if (account != address(0)) {
            rewards[account] = earned(account);
            userRewardPerTokenPaid[account] = rewardPerTokenStored;
        }
        _;
    }

    /* ========== EVENTS ========== */

    event RewardAdded(uint256 reward);
    event Staked(address indexed user, uint256 amount);
    event Withdrawn(address indexed user, uint256 amount);
    event RewardPaid(address indexed user, uint256 reward);
}
```

6. 附录 B：安全风险评级标准

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

7. 附录 C：智能合约安全审计工具简介

7.1. Manticore

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

7.2. Oyente

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

7.3. securify.sh

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

7.4. Echidna

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

7.5. MAIAN

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

7.6. ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

7.7. ida-evm

ida-evm 是一个针对以太坊虚拟机（EVM）的 IDA 处理器模块。

7.8. Remix-ide

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

7.9. 知道创宇区块链安全审计人员专用工具包

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

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587

邮　　箱 sec@knownsec.com

官　　网 www.knownsec.com

地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509