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	20210323	知道创宇区块链安全研究团队	V1.0

文档信息

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

声明

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

目录

1. 综述

本次报告有效测试时间是从 2021 年 03 月 22 日开始到 2021 年 03 月 24 日结束，在此期间针对 **VIP 智能合约代码**的安全性和规范性进行审计并以此作为报告统计依据。

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

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

本次智能合约安全审计结果： 通过

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

本次审计的报告信息：

报告编号：

报告查询地址链接:

本次审计的目标信息：

条目	描述
Token 名称	VIP
代码地址	https://bscscan.com/address/0x03f711082adebf90e21cbf71ad60e98b489c66ae#code
代码类型	代币代码、BNB 智能合约代码
代码语言	Solidity

合约文件及哈希：

合约文件	MD5
VipToken.sol	E59CAD781CA1252D1EF01CF8AE88283A
VipPool.sol	9D0F9E61C83F28651731E0BC19A0D911

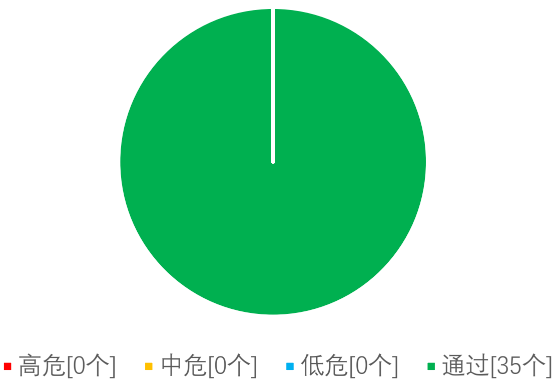
2. 代码漏洞分析

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	35

风险等级分布图



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	VipToken.sol 合约变量及构造函数	通过	经检测，不存在安全问题。
	VipToken.sol 合约 mint 函数	通过	经检测，不存在安全问题。
	VipToken.sol 合约铸币相关函数	通过	经检测，不存在安全问题。
	VipPool.sol 合约变量及构造函数	通过	经检测，不存在安全问题。
	VipPool.sol 合约更新矿池奖励相关函数	通过	经检测，不存在安全问题。
	VipToken.sol 合约存入代币相关函数	通过	经检测，不存在安全问题。
	VipToken.sol 合约从矿池提现相关函数	通过	经检测，不存在安全问题。
	VipToken.sol 合约 emergencyWithdrawVip 函数	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.origin 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. VipToken.sol 合约变量及构造函数【通过】

审计分析：VipToken.sol 合约变量定义及构造函数设计合理。

```
contract VIPToken is DelegateERC20, Ownable {
    uint256 private constant preMineSupply = 5250000 * 1e18;
    uint256 private constant maxSupply = 21000000 * 1e18;     // the total supply

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _minters;

    constructor() public ERC20("Vipswap Token", "VIP"){
        _mint(msg.sender, preMineSupply);
    }
    ……
}
```

安全建议：无。

3.2. VipToken.sol 合约 mint 函数【通过】

审计分析：合约 VipToken.sol 的 mint 函数用于铸币。经审计，该处功能设计合理，权限控制正确。

```
    function mint(address _to, uint256 _amount) public onlyMinter returns (bool) {
        if (_amount.add(totalSupply()) > maxSupply) {//knownsec 判断铸币量是否超过最大值
            return false;
        }
        _mint(_to, _amount);
        return true;
    }
```

安全建议：无。

3.3. VipToken.sol 合约铸币相关函数【通过】

审计分析：合约 VipToken.sol 铸币相关函数，addMinter 用于添加铸币权限，delMinter 用于删除铸币权限，getMinterLength 用于获取铸币人数，isMinter 用于判断是否有铸币权限。经审计，该处功能设计合理，权限控制正确。

```
function addMinter(address _addMinter) public onlyOwner returns (bool) {
        require(_addMinter != address(0), "VIPToken: _addMinter is the zero address");
        return EnumerableSet.add(_minters, _addMinter);
    }

    function delMinter(address _delMinter) public onlyOwner returns (bool) {
        require(_delMinter != address(0), "VIPToken: _delMinter is the zero address");
        return EnumerableSet.remove(_minters, _delMinter);
    }

    function getMinterLength() public view returns (uint256) {
        return EnumerableSet.length(_minters);
    }

    function isMinter(address account) public view returns (bool) {
        return EnumerableSet.contains(_minters, account);
    }

    function getMinter(uint256 _index) public view onlyOwner returns (address){
        require(_index <= getMinterLength() - 1, "VIPToken: index out of bounds");
        return EnumerableSet.at(_minters, _index);
    }

    // modifier for mint function
    modifier onlyMinter() {
        require(isMinter(msg.sender), "caller is not the minter");
        _;
    }
```

安全建议：无。

3.4. VipPool.sol 合约变量及构造函数【通过】

审计分析：合约 VipPool.sol 合约变量定义及构造函数设计合理。

```
contract BscPool is Ownable {
    using SafeMath for uint256;
    using SafeERC20 for IERC20;

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

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

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

```
}

// The VIP Token!
IVIP public vip;
// VIP tokens created per block.
uint256 public vipPerBlock;
// Info of each pool.
PoolInfo[] public poolInfo;
// Info of each user that stakes LP tokens.
mapping(uint256 => mapping(address => UserInfo)) public userInfo;
// Corresponding to the pid of the multLP pool
mapping(uint256 => uint256) public poolCorrespond;
// pid corresponding address
mapping(address => uint256) public LpOfPid;
// Control mining
bool public paused = false;
// Total allocation points. Must be the sum of all allocation points in all pools.
uint256 public totalAllocPoint = 0;
// The block number when VIP mining starts.
uint256 public startBlock;
// multLP MasterChef
address public multLpChef;
// multLP Token
address public multLpToken;
// How many blocks are halved
uint256 public halvingPeriod = 5256000;

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

constructor(
    IVIP _vip,
        uint256 _vipPerBlock,
        uint256 _startBlock
    ) public {
        vip = _vip;
        vipPerBlock = _vipPerBlock;
        startBlock = _startBlock;
    }
    ……
}
```

安全建议：无。

3.5. VipPool.sol 合约更新矿池奖励相关函数【通过】

审计分析：合约 VipPool.sol 的 setVipPerBlock 函数用于设置 VipToken 的产出效率，massUpdatePools 函数用于遍历更新矿池，updatePool 函数用于具体实现更新矿池的细节。经审计，该处功能设计合理，权限控制正确。

```
    function setVipPerBlock(uint256 _newPerBlock) public onlyOwner {
        massUpdatePools();
        vipPerBlock = _newPerBlock;
    }

    function massUpdatePools() public {
        uint256 length = poolInfo.length;
        for (uint256 pid = 0; pid < length; ++pid) {
            updatePool(pid);
        }
    }

    function updatePool(uint256 _pid) public {
            PoolInfo storage pool = poolInfo[_pid];
            if (block.number <= pool.lastRewardBlock) {//knownsec 判断是否为最新的块
            return;
        }
        uint256 lpSupply;
        if (isMultLP(address(pool.lpToken))) {
            if (pool.totalAmount == 0) {//konwonsec 如果矿池存款总额为 0
                pool.lastRewardBlock = block.number;//knownsec 矿池最新奖励块为
当前块
                return;
            }
            lpSupply = pool.totalAmount;
        } else {
            lpSupply = pool.lpToken.balanceOf(address(this));
            if (lpSupply == 0) {
                pool.lastRewardBlock = block.number;
                return;
            }
        }
        uint256 blockReward = getVipBlockReward(pool.lastRewardBlock);
        if (blockReward <= 0) {
            return;
        }
        uint256 vipReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
        bool minRet = vip.mint(address(this), vipReward);
        if (minRet) {
            pool.accVipPerShare                                                =
pool.accVipPerShare.add(vipReward.mul(1e12).div(lpSupply));
        }
        pool.lastRewardBlock = block.number;
    }
```

安全建议：无。

3.6. VipPool.sol 合约存入代币相关函数【通过】

审计分析：合约 VipPool.sol 的 deposit 用于存入代币。depositVipAndToken 函数用于分情况奖励代币，depositVip 函数用于分情况奖励 VIPToken。根据挖矿情况经审计，该处功能设计合理，权限控制正确。

```
function deposit(uint256 _pid, uint256 _amount) public notPause {
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            depositVipAndToken(_pid, _amount, msg.sender);
        } else {
            depositVip(_pid, _amount, msg.sender);
        }
    }

function depositVipAndToken(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256                    pendingAmount                    =
user.amount.mul(pool.accVipPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeVipTransfer(_user, pendingAmount);
            }
            uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
            IMasterChefBsc(multLpChef).deposit(poolCorrespond[_pid], 0);
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare                                       =
pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
            uint256                    tokenPending                     =
user.amount.mul(pool.accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
```

```
                if (tokenPending > 0) {
                    IERC20(multLpToken).safeTransfer(_user, tokenPending);
                }
            }
            if (_amount > 0) {
                pool.lpToken.safeTransferFrom(_user, address(this), _amount);
                if (pool.totalAmount == 0) {
                    IMasterChefBsc(multLpChef).deposit(poolCorrespond[_pid], _amount);
                    user.amount = user.amount.add(_amount);
                    pool.totalAmount = pool.totalAmount.add(_amount);
                } else {
                    uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
                    IMasterChefBsc(multLpChef).deposit(poolCorrespond[_pid], _amount);
                    uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
                    pool.accMultLpPerShare                                    =
pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
                    user.amount = user.amount.add(_amount);
                    pool.totalAmount = pool.totalAmount.add(_amount);
                }
            }
            user.rewardDebt = user.amount.mul(pool.accVipPerShare).div(1e12);
            user.multLpRewardDebt = user.amount.mul(pool.accMultLpPerShare).div(1e12);
            emit Deposit(_user, _pid, _amount);
        }

        function depositVip(uint256 _pid, uint256 _amount, address _user) private {
            PoolInfo storage pool = poolInfo[_pid];
            UserInfo storage user = userInfo[_pid][_user];
            updatePool(_pid);
            if (user.amount > 0) {
                uint256                          pendingAmount                     =
user.amount.mul(pool.accVipPerShare).div(1e12).sub(user.rewardDebt);
                if (pendingAmount > 0) {
```

```
                safeVipTransfer(_user, pendingAmount);
            }
        }
        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(_user, address(this), _amount);
            user.amount = user.amount.add(_amount);
            pool.totalAmount = pool.totalAmount.add(_amount);
        }
        user.rewardDebt = user.amount.mul(pool.accVipPerShare).div(1e12);
        emit Deposit(_user, _pid, _amount);
    }
```

安全建议：无。

3.7. VipPool.sol 合约从矿池提现相关函数【通过】

审计分析：合约 VipPool.sol 的 withdraw 函数从矿池提取 VioToken。经审计，该处功能设计合理，权限控制正确。

```
function withdraw(uint256 _pid, uint256 _amount) public notPause {
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            withdrawVipAndToken(_pid, _amount, msg.sender);
        } else {
            withdrawVip(_pid, _amount, msg.sender);
        }
    }

    function withdrawVipAndToken(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        require(user.amount >= _amount, "withdrawVipAndToken: not good");
        updatePool(_pid);
        uint256 pendingAmount = user.amount.mul(pool.accVipPerShare).div(1e12).sub(user.rewardDebt);
        if (pendingAmount > 0) {
            safeVipTransfer(_user, pendingAmount);
        }
        if (_amount > 0) {
            uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
            IMasterChefBsc(multLpChef).withdraw(poolCorrespond[_pid], _amount);
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare = pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
            uint256 tokenPending = user.amount.mul(pool.accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
            if (tokenPending > 0) {
                IERC20(multLpToken).safeTransfer(_user, tokenPending);
            }
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            pool.lpToken.safeTransfer(_user, _amount);
        }
        user.rewardDebt = user.amount.mul(pool.accVipPerShare).div(1e12);
        user.multLpRewardDebt = user.amount.mul(pool.accMultLpPerShare).div(1e12);
        emit Withdraw(_user, _pid, _amount);
    }

    function withdrawVip(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        require(user.amount >= _amount, "withdrawVip: not good");
        updatePool(_pid);
        uint256 pendingAmount = user.amount.mul(pool.accVipPerShare).div(1e12).sub(user.rewardDebt);
        if (pendingAmount > 0) {
            safeVipTransfer(_user, pendingAmount);
        }
        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            pool.lpToken.safeTransfer(_user, _amount);
        }
        user.rewardDebt = user.amount.mul(pool.accVipPerShare).div(1e12);
        emit Withdraw(_user, _pid, _amount);
    }
```

安全建议：无。

3.8. VipPool.sol 合约 emergencyWithdrawVip 函数【通过】

审计分析：合约 VipPool.sol 的 emergencyWithdrawVip 函数用于紧急提现 VipToken。经审计，该处功能设计合理，权限控制正确。

```
    function emergencyWithdrawVip(uint256 _pid, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        pool.lpToken.safeTransfer(_user, amount);
        pool.totalAmount = pool.totalAmount.sub(amount);
        emit EmergencyWithdraw(_user, _pid, amount);
    }
```

安全建议：无。

4. 代码基本漏洞检测

4.1. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本

检测结果：经检测，智能合约代码中制定了编译器版本 0.6.0 以上，不存在该安全问题。

安全建议：无。

4.2. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.3. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

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

安全建议：无。

4.4. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.5. require/assert 的合理使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.6. fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.7. tx.origin 身份验证【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.8. owner 权限控制【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.9. 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，智能合约代码中不存在该安全问题。

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，但由于该挖矿方式需要增发代币，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，智能合约代码不存在该问题。

安全建议：无。

4.20. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

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

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

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.21. 交易顺序依赖【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.22. 时间戳依赖攻击【通过】

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

检查合约代码实现中是否存在有依赖于时间戳的关键功能

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.23. 拒绝服务攻击【通过】

在区块链的世界中，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】

Solidity 中的 call.value()函数在被用来发送 BNB 的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送 BNB 的操作发生在实际减少发送者账户的余

额之前时，就会存在重入攻击的风险。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

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

检测结果:经检测，智能合约代码中不存在相关漏洞。

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：

VipToken.sol

```
/**
 *Submitted for verification at BscScan.com on 2021-03-16
*/

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


/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
	function _msgSender() internal view virtual returns (address payable) {
		return msg.sender;
	}

	function _msgData() internal view virtual returns (bytes memory) {
		this;
		// silence state mutability warning without generating bytecode - see
https://github.com/ethereum/solidity/issues/2691
		return msg.data;
	}
}


/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable is Context {
	address private _owner;

	event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

	/**
	 * @dev Initializes the contract setting the deployer as the initial owner.
	 */
	constructor () internal {
		address msgSender = _msgSender();
		_owner = msgSender;
		emit OwnershipTransferred(address(0), msgSender);
	}

	/**
	 * @dev Returns the address of the current owner.
	 */
	function owner() public view returns (address) {
		return _owner;
	}

	/**
	 * @dev Throws if called by any account other than the owner.
	 */
	modifier onlyOwner() {
		require(_owner == _msgSender(), "Ownable: caller is not the owner");
		_;
	}

	/**
	 * @dev Leaves the contract without owner. It will not be possible to call
	 * `onlyOwner` functions anymore. Can only be called by the current owner.
```

```
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "Ownable: new owner is the zero address");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}


/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);
    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}


/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
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) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message 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, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        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) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message 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, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}


/**
 * @dev Collection of functions related to the address type
 */
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) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly {codehash := extcodehash(account)}
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-
pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success,) = recipient.call{value : amount}("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
     * If `target` reverts with a revert reason, it is bubbled up by this
     * function (like regular Solidity function calls).
     *
     * Returns the raw returned data. To convert to the expected return value,
     * use https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-
encoding-and-decoding-functions[`abi.decode`].
     *
     * Requirements:
     *
     * - `target` must be a contract.
     * - calling `target` with `data` must not revert.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data) internal returns (bytes memory) {
        return functionCall(target, data, "Address: low-level call failed");
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
     * `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes
memory) {
        return _functionCallWithValue(target, data, 0, errorMessage);
    }

    /**
     * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
     * but also transferring `value` wei to `target`.
     *
     * Requirements:
     *
     * - the calling contract must have an ETH balance of at least `value`.
     * - the called Solidity function must be `payable`.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes
memory) {
        return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
    }

    /**
     * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
     * with `errorMessage` as a fallback revert reason when `target` reverts.
     *
     * _Available since v3.1._
     */
    function functionCallWithValue(address target, bytes memory data, uint256 value, string memory
errorMessage) internal returns (bytes memory) {
        require(address(this).balance >= value, "Address: insufficient balance for call");
        return _functionCallWithValue(target, data, value, errorMessage);
    }

    function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory
errorMessage) private returns (bytes memory) {
        require(isContract(target), "Address: call to non-contract");

        // solhint-disable-next-line avoid-low-level-calls
        (bool success, bytes memory returndata) = target.call{value : weiValue}(data);
        if (success) {
            return returndata;
        } else {
            // Look for revert reason and bubble it up if present
            if (returndata.length > 0) {
                // The easiest way to bubble the revert reason is using memory via assembly

                // solhint-disable-next-line no-inline-assembly
                assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
            } else {
                revert(errorMessage);
            }
        }
    }
}


/**
 * @dev Implementation of the {IERC20} interface.
 *
 * This implementation is agnostic to the way tokens are created. This means
 * that a supply mechanism has to be added in a derived contract using {_mint}.
 * For a generic mechanism see {ERC20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-erc20-supply-mechanisms/226[How
 * to implement supply mechanisms].
 *
 * We have followed general OpenZeppelin guidelines: functions revert instead
 * of returning `false` on failure. This behavior is nonetheless conventional
 * and does not conflict with the expectations of ERC20 applications.
 *
 * Additionally, an {Approval} event is emitted on calls to {transferFrom}.
 * This allows applications to reconstruct the allowance for all accounts just
 * by listening to said events. Other implementations of the EIP may not emit
 * these events, as it isn't required by the specification.
 *
 * Finally, the non-standard {decreaseAllowance} and {increaseAllowance}
 * functions have been added to mitigate the well-known issues around setting
 * allowances. See {IERC20-approve}.
 */
contract ERC20 is Context, IERC20 {
    using SafeMath for uint256;
    using Address for address;

    mapping(address => uint256) private _balances;

    mapping(address => mapping(address => uint256)) private _allowances;

    uint256 private _totalSupply;

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

```
/**
 * @dev Sets the values for {name} and {symbol}, initializes {decimals} with
 * a default value of 18.
 *
 * To select a different value for {decimals}, use {_setupDecimals}.
 *
 * All three of these values are immutable: they can only be set once during
 * construction.
 */
constructor (string memory name, string memory symbol) public {
        _name = name;
        _symbol = symbol;
        _decimals = 18;
}

/**
 * @dev Returns the name of the token.
 */
function name() public view returns (string memory) {
        return _name;
}

/**
 * @dev Returns the symbol of the token, usually a shorter version of the
 * name.
 */
function symbol() public view returns (string memory) {
        return _symbol;
}

/**
 * @dev Returns the number of decimals used to get its user representation.
 * For example, if `decimals` equals `2`, a balance of `505` tokens should
 * be displayed to a user as `5,05` (`505 / 10 ** 2`).
 *
 * Tokens usually opt for a value of 18, imitating the relationship between
 * Ether and Wei. This is the value {ERC20} uses, unless {_setupDecimals} is
 * called.
 *
 * NOTE: This information is only used for _display_ purposes: it in
 * no way affects any of the arithmetic of the contract, including
 * {IERC20-balanceOf} and {IERC20-transfer}.
 */
function decimals() public view returns (uint8) {
        return _decimals;
}

/**
 * @dev See {IERC20-totalSupply}.
 */
function totalSupply() public view override returns (uint256) {
        return _totalSupply;
}

/**
 * @dev See {IERC20-balanceOf}.
 */
function balanceOf(address account) public view override returns (uint256) {
        return _balances[account];
}

/**
 * @dev See {IERC20-transfer}.
 *
 * Requirements:
 *
 * - `recipient` cannot be the zero address.
 * - the caller must have a balance of at least `amount`.
 */
function transfer(address recipient, uint256 amount) public virtual override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
}

/**
 * @dev See {IERC20-allowance}.
 */
function allowance(address owner, address spender) public view virtual override returns (uint256) {
        return _allowances[owner][spender];
}

/**
 * @dev See {IERC20-approve}.
 *
 * Requirements:
 *
```

```
     * - `spender` cannot be the zero address.
     */
    function approve(address spender, uint256 amount) public virtual override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {IERC20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {ERC20};
     *
     * Requirements:
     * - `sender` and `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     * - the caller must have allowance for ``sender``'s tokens of at least
     * `amount`.
     */
    function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool)
{
        _transfer(sender, recipient, amount);
        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer
amount exceeds allowance"));
        return true;
    }

    /**
     * @dev Atomically increases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }

    /**
     * @dev Atomically decreases the allowance granted to `spender` by the caller.
     *
     * This is an alternative to {approve} that can be used as a mitigation for
     * problems described in {IERC20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     * - `spender` must have allowance for the caller of at least
     * `subtractedValue`.
     */
    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].sub(subtractedValue, "ERC20:
decreased allowance below zero"));
        return true;
    }

    /**
     * @dev Moves tokens `amount` from `sender` to `recipient`.
     *
     * This is internal function is equivalent to {transfer}, and can be used to
     * e.g. implement automatic token fees, slashing mechanisms, etc.
     *
     * Emits a {Transfer} event.
     *
     * Requirements:
     *
     * - `sender` cannot be the zero address.
     * - `recipient` cannot be the zero address.
     * - `sender` must have a balance of at least `amount`.
     */
    function _transfer(address sender, address recipient, uint256 amount) internal virtual {
        require(sender != address(0), "ERC20: transfer from the zero address");
        require(recipient != address(0), "ERC20: transfer to the zero address");

        _beforeTokenTransfer(sender, recipient, amount);

        _balances[sender] = _balances[sender].sub(amount, "ERC20: transfer amount exceeds balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
    }

    /** @dev Creates `amount` tokens and assigns them to `account`, increasing
     * the total supply.
     *
     * Emits a {Transfer} event with `from` set to the zero address.
     *
     * Requirements
     *
     * - `to` cannot be the zero address.
     */
    function _mint(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: mint to the zero address");

        _beforeTokenTransfer(address(0), account, amount);

        _totalSupply = _totalSupply.add(amount);
        _balances[account] = _balances[account].add(amount);
        emit Transfer(address(0), account, amount);
    }

    /**
     * @dev Destroys `amount` tokens from `account`, reducing the
     * total supply.
     *
     * Emits a {Transfer} event with `to` set to the zero address.
     *
     * Requirements
     *
     * - `account` cannot be the zero address.
     * - `account` must have at least `amount` tokens.
     */
    function _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount, "ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

    /**
     * @dev Sets `amount` as the allowance of `spender` over the `owner`s tokens.
     *
     * This is internal function is equivalent to `approve`, and can be used to
     * e.g. set automatic allowances for certain subsystems, etc.
     *
     * Emits an {Approval} event.
     *
     * Requirements:
     *
     * - `owner` cannot be the zero address.
     * - `spender` cannot be the zero address.
     */
    function _approve(address owner, address spender, uint256 amount) internal virtual {
        require(owner != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }

    /**
     * @dev Sets {decimals} to a value other than the default one of 18.
     *
     * WARNING: This function should only be called from the constructor. Most
     * applications that interact with token contracts will not expect
     * {decimals} to ever change, and may work incorrectly if it does.
     */
    function _setupDecimals(uint8 decimals_) internal {
        _decimals = decimals_;
    }

    /**
     * @dev Hook that is called before any transfer of tokens. This includes
     * minting and burning.
     *
     * Calling conditions:
     *
     * - when `from` and `to` are both non-zero, `amount` of ``from``'s tokens
     * will be to transferred to `to`.
     * - when `from` is zero, `amount` tokens will be minted for `to`.
     * - when `to` is zero, `amount` of ``from``'s tokens will be burned.
     * - `from` and `to` are never both zero.
     *
     * To learn more about hooks, head to xref:ROOT:extending-contracts.adoc#using-hooks[Using Hooks].
```

```
        */
        function _beforeTokenTransfer(address from, address to, uint256 amount) internal virtual {}
}


library EnumerableSet {
        // To implement this library for multiple types with as little code
        // repetition as possible, we write it in terms of a generic Set type with
        // bytes32 values.
        // The Set implementation uses private functions, and user-facing
        // implementations (such as AddressSet) are just wrappers around the
        // underlying Set.
        // This means that we can only create new EnumerableSets for types that fit
        // in bytes32.

        struct Set {
                // Storage of set values
                bytes32[] _values;

                // Position of the value in the `values` array, plus 1 because index 0
                // means a value is not in the set.
                mapping (bytes32 => uint256) _indexes;
        }

        /**
         * @dev Add a value to a set. O(1).
         *
         * Returns true if the value was added to the set, that is if it was not
         * already present.
         */
        function _add(Set storage set, bytes32 value) private returns (bool) {
                if (!_contains(set, value)) {
                        set._values.push(value);
                        // The value is stored at length-1, but we add 1 to all indexes
                        // and use 0 as a sentinel value
                        set._indexes[value] = set._values.length;
                        return true;
                } else {
                        return false;
                }
        }

        /**
         * @dev Removes a value from a set. O(1).
         *
         * Returns true if the value was removed from the set, that is if it was
         * present.
         */
        function _remove(Set storage set, bytes32 value) private returns (bool) {
                // We read and store the value's index to prevent multiple reads from the same storage slot
                uint256 valueIndex = set._indexes[value];

                if (valueIndex != 0) { // Equivalent to contains(set, value)
                        // To delete an element from the _values array in O(1), we swap the element to delete with the last
one in
                        // the array, and then remove the last element (sometimes called as 'swap and pop').
                        // This modifies the order of the array, as noted in {at}.

                        uint256 toDeleteIndex = valueIndex - 1;
                        uint256 lastIndex = set._values.length - 1;

                        // When the value to delete is the last one, the swap operation is unnecessary. However, since this
occurs
                        // so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' statement.

                        bytes32 lastvalue = set._values[lastIndex];

                        // Move the last value to the index where the value to delete is
                        set._values[toDeleteIndex] = lastvalue;
                        // Update the index for the moved value
                        set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

                        // Delete the slot where the moved value was stored
                        set._values.pop();

                        // Delete the index for the deleted slot
                        delete set._indexes[value];

                        return true;
                } else {
                        return false;
                }
        }

        /**
         * @dev Returns true if the value is in the set. O(1).
         */
```

```
    function _contains(Set storage set, bytes32 value) private view returns (bool) {
        return set._indexes[value] != 0;
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function _length(Set storage set) private view returns (uint256) {
        return set._values.length;
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function _at(Set storage set, uint256 index) private view returns (bytes32) {
        require(set._values.length > index, "EnumerableSet: index out of bounds");
        return set._values[index];
    }

    // Bytes32Set

    struct Bytes32Set {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _add(set._inner, value);
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(Bytes32Set storage set, bytes32 value) internal returns (bool) {
        return _remove(set._inner, value);
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(Bytes32Set storage set, bytes32 value) internal view returns (bool) {
        return _contains(set._inner, value);
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(Bytes32Set storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function at(Bytes32Set storage set, uint256 index) internal view returns (bytes32) {
        return _at(set._inner, index);
    }

    // AddressSet

    struct AddressSet {
        Set _inner;
    }

    /**
```

```
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(AddressSet storage set, address value) internal returns (bool) {
        return _add(set._inner, bytes32(uint256(value)));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(AddressSet storage set, address value) internal returns (bool) {
        return _remove(set._inner, bytes32(uint256(value)));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(AddressSet storage set, address value) internal view returns (bool) {
        return _contains(set._inner, bytes32(uint256(value)));
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(AddressSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function at(AddressSet storage set, uint256 index) internal view returns (address) {
        return address(uint256(_at(set._inner, index)));
    }


    // UintSet

    struct UintSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(UintSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(UintSet storage set, uint256 value) internal view returns (bool) {
        return _contains(set._inner, bytes32(value));
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }
    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
        return uint256(_at(set._inner, index));
    }
}

pragma solidity ^0.6.0;
pragma experimental ABIEncoderV2;

abstract contract DelegateERC20 is ERC20 {
    // @notice A record of each accounts delegate
    mapping (address => address) internal _delegates;

    /// @notice A checkpoint for marking number of votes from a given block
    struct Checkpoint {
        uint32 fromBlock;
        uint256 votes;
    }

    /// @notice A record of votes checkpoints for each account, by index
    mapping (address => mapping (uint32 => Checkpoint)) public checkpoints;

    /// @notice The number of checkpoints for each account
    mapping (address => uint32) public numCheckpoints;

    /// @notice The EIP-712 typehash for the contract's domain
    bytes32 public constant DOMAIN_TYPEHASH = keccak256("EIP712Domain(string name,uint256
chainId,address verifyingContract)");

    /// @notice The EIP-712 typehash for the delegation struct used by the contract
    bytes32 public constant DELEGATION_TYPEHASH = keccak256("Delegation(address delegatee,uint256
nonce,uint256 expiry)");

    /// @notice A record of states for signing / validating signatures
    mapping (address => uint) public nonces;


    // support delegates mint
    function _mint(address account, uint256 amount) internal override virtual {
        super._mint(account, amount);

        // add delegates to the minter
        _moveDelegates(address(0), _delegates[account], amount);
    }


    function _transfer(address sender, address recipient, uint256 amount) internal override virtual {
        super._transfer(sender, recipient, amount);
        _moveDelegates(_delegates[sender], _delegates[recipient], amount);
    }


    /**
     * @notice Delegate votes from `msg.sender` to `delegatee`
     * @param delegatee The address to delegate votes to
     */
    function delegate(address delegatee) external {
        return _delegate(msg.sender, delegatee);
    }

    /**
      * @notice Delegates votes from signatory to `delegatee`
      * @param delegatee The address to delegate votes to
      * @param nonce The contract state required to match the signature
      * @param expiry The time at which to expire the signature
      * @param v The recovery byte of the signature
      * @param r Half of the ECDSA signature pair
      * @param s Half of the ECDSA signature pair
      */
    function delegateBySig(
        address delegatee,
        uint nonce,
        uint expiry,
        uint8 v,
        bytes32 r,
        bytes32 s
    )
    external
    {
        bytes32 domainSeparator = keccak256(
            abi.encode(
                DOMAIN_TYPEHASH,
                keccak256(bytes(name())),
                getChainId(),
                address(this)
            )
        );

        bytes32 structHash = keccak256(
            abi.encode(
                DELEGATION_TYPEHASH,
                delegatee,
                nonce,
                expiry
            )
        );

        bytes32 digest = keccak256(
            abi.encodePacked(
                "\x19\x01",
                domainSeparator,
                structHash
            )
        );

        address signatory = ecrecover(digest, v, r, s);
        require(signatory != address(0), "VIPToken::delegateBySig: invalid signature");
        require(nonce == nonces[signatory]++, "VIPToken::delegateBySig: invalid nonce");
        require(now <= expiry, "VIPToken::delegateBySig: signature expired");
        return _delegate(signatory, delegatee);
    }

    /**
     * @notice Gets the current votes balance for `account`
     * @param account The address to get votes balance
     * @return The number of current votes for `account`
     */
    function getCurrentVotes(address account)
    external
    view
    returns (uint256)
    {
        uint32 nCheckpoints = numCheckpoints[account];
        return nCheckpoints > 0 ? checkpoints[account][nCheckpoints - 1].votes : 0;
    }

    /**
     * @notice Determine the prior number of votes for an account as of a block number
     * @dev Block number must be a finalized block or else this function will revert to prevent misinformation.
     * @param account The address of the account to check
     * @param blockNumber The block number to get the vote balance at
     * @return The number of votes the account had as of the given block
     */
    function getPriorVotes(address account, uint blockNumber)
    external
    view
    returns (uint256)
    {
        require(blockNumber < block.number, "VIPToken::getPriorVotes: not yet determined");

        uint32 nCheckpoints = numCheckpoints[account];
        if (nCheckpoints == 0) {
            return 0;
        }

        // First check most recent balance
        if (checkpoints[account][nCheckpoints - 1].fromBlock <= blockNumber) {
            return checkpoints[account][nCheckpoints - 1].votes;
        }

        // Next check implicit zero balance
        if (checkpoints[account][0].fromBlock > blockNumber) {
            return 0;
        }

        uint32 lower = 0;
        uint32 upper = nCheckpoints - 1;
        while (upper > lower) {
            uint32 center = upper - (upper - lower) / 2; // ceil, avoiding overflow
            Checkpoint memory cp = checkpoints[account][center];
            if (cp.fromBlock == blockNumber) {
                return cp.votes;
            } else if (cp.fromBlock < blockNumber) {
                lower = center;
                } else {
                    upper = center - 1;
                }
            }
            return checkpoints[account][lower].votes;
        }

        function _delegate(address delegator, address delegatee)
        internal
        {
            address currentDelegate = _delegates[delegator];
            uint256 delegatorBalance = balanceOf(delegator); // balance of underlying balances (not scaled);
            _delegates[delegator] = delegatee;

            _moveDelegates(currentDelegate, delegatee, delegatorBalance);

            emit DelegateChanged(delegator, currentDelegate, delegatee);
        }

        function _moveDelegates(address srcRep, address dstRep, uint256 amount) internal {
            if (srcRep != dstRep && amount > 0) {
                if (srcRep != address(0)) {
                    // decrease old representative
                    uint32 srcRepNum = numCheckpoints[srcRep];
                    uint256 srcRepOld = srcRepNum > 0 ? checkpoints[srcRep][srcRepNum - 1].votes : 0;
                    uint256 srcRepNew = srcRepOld.sub(amount);
                    _writeCheckpoint(srcRep, srcRepNum, srcRepOld, srcRepNew);
                }

                if (dstRep != address(0)) {
                    // increase new representative
                    uint32 dstRepNum = numCheckpoints[dstRep];
                    uint256 dstRepOld = dstRepNum > 0 ? checkpoints[dstRep][dstRepNum - 1].votes : 0;
                    uint256 dstRepNew = dstRepOld.add(amount);
                    _writeCheckpoint(dstRep, dstRepNum, dstRepOld, dstRepNew);
                }
            }
        }

        function _writeCheckpoint(
            address delegatee,
            uint32 nCheckpoints,
            uint256 oldVotes,
            uint256 newVotes
        )
        internal
        {
            uint32 blockNumber = safe32(block.number, "VIPToken::_writeCheckpoint: block number exceeds 32
bits");

            if (nCheckpoints > 0 && checkpoints[delegatee][nCheckpoints - 1].fromBlock == blockNumber) {
                checkpoints[delegatee][nCheckpoints - 1].votes = newVotes;
            } else {
                checkpoints[delegatee][nCheckpoints] = Checkpoint(blockNumber, newVotes);
                numCheckpoints[delegatee] = nCheckpoints + 1;
            }

            emit DelegateVotesChanged(delegatee, oldVotes, newVotes);
        }

        function safe32(uint n, string memory errorMessage) internal pure returns (uint32) {
            require(n < 2**32, errorMessage);
            return uint32(n);
        }

        function getChainId() internal pure returns (uint) {
            uint256 chainId;
            assembly { chainId := chainid() }

            return chainId;
        }

        /// @notice An event thats emitted when an account changes its delegate
        event DelegateChanged(address indexed delegator, address indexed fromDelegate, address indexed
toDelegate);

        /// @notice An event thats emitted when a delegate account's vote balance changes
        event DelegateVotesChanged(address indexed delegate, uint previousBalance, uint newBalance);

}

contract VIPToken is DelegateERC20, Ownable {
        uint256 private constant preMineSupply = 5250000 * 1e18;
        uint256 private constant maxSupply = 21000000 * 1e18;        // the total supply

        using EnumerableSet for EnumerableSet.AddressSet;
        EnumerableSet.AddressSet private _minters;
```

```
    constructor() public ERC20("Vipswap Token", "VIP"){
        _mint(msg.sender, preMineSupply);
    }

    // mint with max supply
    function mint(address _to, uint256 _amount) public onlyMinter returns (bool) {
        if (_amount.add(totalSupply()) > maxSupply) {
            return false;
        }
        _mint(_to, _amount);
        return true;
    }

    function addMinter(address _addMinter) public onlyOwner returns (bool) {
        require(_addMinter != address(0), "VIPToken: _addMinter is the zero address");
        return EnumerableSet.add(_minters, _addMinter);
    }

    function delMinter(address _delMinter) public onlyOwner returns (bool) {
        require(_delMinter != address(0), "VIPToken: _delMinter is the zero address");
        return EnumerableSet.remove(_minters, _delMinter);
    }

    function getMinterLength() public view returns (uint256) {
        return EnumerableSet.length(_minters);
    }

    function isMinter(address account) public view returns (bool) {
        return EnumerableSet.contains(_minters, account);
    }

    function getMinter(uint256 _index) public view onlyOwner returns (address){
        require(_index <= getMinterLength() - 1, "VIPToken: index out of bounds");
        return EnumerableSet.at(_minters, _index);
    }

    // modifier for mint function
    modifier onlyMinter() {
        require(isMinter(msg.sender), "caller is not the minter");
        _;
    }

}


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

/**
 * @dev Interface of the ERC20 standard as defined in the EIP.
 */
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender` will be
     * allowed to spend on behalf of `owner` through {transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are called.
     */
    function allowance(address owner, address spender) external view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this method brings the risk
     * that someone may use both the old and the new allowance by unfortunate
     * transaction ordering. One possible solution to mitigate this race
     * condition is to first reduce the spender's allowance to 0 and set the
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}


pragma solidity ^0.6.0;

/**
 * @dev Wrappers over Solidity's arithmetic operations with added overflow
 * checks.
 *
 * Arithmetic operations in Solidity wrap on overflow. This can easily result
 * in bugs, because programmers usually assume that an overflow raises an
 * error, which is the standard behavior in high level programming languages.
 * `SafeMath` restores this intuition by reverting the transaction when an
 * operation overflows.
 *
 * Using this library instead of the unchecked operations eliminates an entire
 * class of bugs, so it's recommended to use it always.
 */
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) {
        return div(a, b, "SafeMath: division by zero");
    }

    /**
     * @dev Returns the integer division of two unsigned integers. Reverts with custom message 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, string memory errorMessage) internal pure returns (uint256) {
        require(b > 0, errorMessage);
        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) {
        return mod(a, b, "SafeMath: modulo by zero");
    }

    /**
     * @dev Returns the remainder of dividing two unsigned integers. (unsigned integer modulo),
     * Reverts with custom message 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, string memory errorMessage) internal pure returns (uint256) {
        require(b != 0, errorMessage);
        return a % b;
    }
}


pragma solidity ^0.6.2;

/**
 * @dev Collection of functions related to the address type
 */
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) {
        // According to EIP-1052, 0x0 is the value returned for not-yet created accounts
        // and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

    /**
     * @dev Replacement for Solidity's `transfer`: sends `amount` wei to
     * `recipient`, forwarding all available gas and reverting on errors.
     *
     * https://eips.ethereum.org/EIPS/eip-1884[EIP1884] increases the gas cost
     * of certain opcodes, possibly making contracts go over the 2300 gas limit
     * imposed by `transfer`, making them unable to receive funds via
     * `transfer`. {sendValue} removes this limitation.
     *
     * https://diligence.consensys.net/posts/2019/09/stop-using-soliditys-transfer-now/[Learn more].
     *
     * IMPORTANT: because control is transferred to `recipient`, care must be
     * taken to not create reentrancy vulnerabilities. Consider using
     * {ReentrancyGuard} or the
     * https://solidity.readthedocs.io/en/v0.5.11/security-considerations.html#use-the-checks-effects-interactions-
  pattern[checks-effects-interactions pattern].
     */
    function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls, avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have reverted");
    }

    /**
     * @dev Performs a Solidity function call using a low level `call`. A
     * plain`call` is an unsafe replacement for a function call: use this
     * function instead.
     *
```

```
         * If `target` reverts with a revert reason, it is bubbled up by this
         * function (like regular Solidity function calls).
         *
         * Returns the raw returned data. To convert to the expected return value,
         *   use   https://solidity.readthedocs.io/en/latest/units-and-global-variables.html?highlight=abi.decode#abi-
    encoding-and-decoding-functions[`abi.decode`].
         *
         * Requirements:
         *
         * - `target` must be a contract.
         * - calling `target` with `data` must not revert.
         *
         * _Available since v3.1._
         */
      function functionCall(address target, bytes memory data) internal returns (bytes memory) {
            return functionCall(target, data, "Address: low-level call failed");
      }

      /**
         * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`], but with
         * `errorMessage` as a fallback revert reason when `target` reverts.
         *
         * _Available since v3.1._
         */
      function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes
    memory) {
            return _functionCallWithValue(target, data, 0, errorMessage);
      }

      /**
         * @dev Same as {xref-Address-functionCall-address-bytes-}[`functionCall`],
         * but also transferring `value` wei to `target`.
         *
         * Requirements:
         *
         * - the calling contract must have an ETH balance of at least `value`.
         * - the called Solidity function must be `payable`.
         *
         * _Available since v3.1._
         */
      function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes
    memory) {
            return functionCallWithValue(target, data, value, "Address: low-level call with value failed");
      }

      /**
         * @dev Same as {xref-Address-functionCallWithValue-address-bytes-uint256-}[`functionCallWithValue`], but
         * with `errorMessage` as a fallback revert reason when `target` reverts.
         *
         * _Available since v3.1._
         */
      function functionCallWithValue(address target, bytes memory data, uint256 value, string memory
    errorMessage) internal returns (bytes memory) {
            require(address(this).balance >= value, "Address: insufficient balance for call");
            return _functionCallWithValue(target, data, value, errorMessage);
      }

      function _functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory
    errorMessage) private returns (bytes memory) {
            require(isContract(target), "Address: call to non-contract");

            // solhint-disable-next-line avoid-low-level-calls
            (bool success, bytes memory returndata) = target.call{value : weiValue}(data);
            if (success) {
                  return returndata;
            } else {
                  // Look for revert reason and bubble it up if present
                  if (returndata.length > 0) {
                        // The easiest way to bubble the revert reason is using memory via assembly

                        // solhint-disable-next-line no-inline-assembly
                        assembly {
                              let returndata_size := mload(returndata)
                              revert(add(32, returndata), returndata_size)
                        }
                  } else {
                        revert(errorMessage);
                  }
            }
      }
}


pragma solidity ^0.6.0;

/**
```

```
 * @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 IERC20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeERC20 {
        using SafeMath for uint256;
        using Address for address;

        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));
        }

        /**
          * @dev Deprecated. This function has issues similar to the ones found in
          * {IERC20-approve}, and its usage is discouraged.
          *
          * Whenever possible, use {safeIncreaseAllowance} and
          * {safeDecreaseAllowance} instead.
          */
        function safeApprove(IERC20 token, address spender, uint256 value) internal {
                // safeApprove should only be called when setting an initial allowance,
                // or when resetting it to zero. To increase and decrease it, use
                // 'safeIncreaseAllowance' and 'safeDecreaseAllowance'
                // solhint-disable-next-line max-line-length
                require((value == 0) || (token.allowance(address(this), spender) == 0),
                        "SafeERC20: approve from non-zero to non-zero allowance"
                );
                _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
        }

        function safeIncreaseAllowance(IERC20 token, address spender, uint256 value) internal {
                uint256 newAllowance = token.allowance(address(this), spender).add(value);
                _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
        }

        function safeDecreaseAllowance(IERC20 token, address spender, uint256 value) internal {
                uint256 newAllowance = token.allowance(address(this), spender).sub(value, "SafeERC20: decreased
    allowance below zero");
                _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, newAllowance));
        }

        /**
          * @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. We use {Address.functionCall} to perform this call, which verifies that
                // the target address contains contract code and also asserts for success in the low-level call.

                bytes memory returndata = address(token).functionCall(data, "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");
                }
        }
}


pragma solidity ^0.6.0;

/*
 * @dev Provides information about the current execution context, including the
 * sender of the transaction and its data. While these are generally available
 * via msg.sender and msg.data, they should not be accessed in such a direct
 * manner, since when dealing with GSN meta-transactions the account sending and
 * paying for execution may not be the actual sender (as far as an application
 * is concerned).
 *
 * This contract is only required for intermediate, library-like contracts.
 */
abstract contract Context {
        function _msgSender() internal view virtual returns (address payable) {
                return msg.sender;
        }

        function _msgData() internal view virtual returns (bytes memory) {
            this;
            // silence state mutability warning without generating bytecode - see
    https://github.com/ethereum/solidity/issues/2691
            return msg.data;
        }
}


pragma solidity ^0.6.0;

/**
  * @dev Contract module which provides a basic access control mechanism, where
  * there is an account (an owner) that can be granted exclusive access to
  * specific functions.
  *
  * By default, the owner account will be the one that deploys the contract. This
  * can later be changed with {transferOwnership}.
  *
  * This module is used through inheritance. It will make available the modifier
  * `onlyOwner`, which can be applied to your functions to restrict their use to
  * the owner.
  */
contract Ownable is Context {
        address private _owner;

        event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

        /**
          * @dev Initializes the contract setting the deployer as the initial owner.
          */
        constructor () internal {
                address msgSender = _msgSender();
                _owner = msgSender;
                emit OwnershipTransferred(address(0), msgSender);
        }

        /**
          * @dev Returns the address of the current owner.
          */
        function owner() public view returns (address) {
                return _owner;
        }

        /**
          * @dev Throws if called by any account other than the owner.
          */
        modifier onlyOwner() {
                require(_owner == _msgSender(), "Ownable: caller is not the owner");
                _;
        }

        /**
          * @dev Leaves the contract without owner. It will not be possible to call
          * `onlyOwner` functions anymore. Can only be called by the current owner.
          *
          * NOTE: Renouncing ownership will leave the contract without an owner,
          * thereby removing any functionality that is only available to the owner.
          */
        function renounceOwnership() public virtual onlyOwner {
                emit OwnershipTransferred(_owner, address(0));
                _owner = address(0);
        }

        /**
          * @dev Transfers ownership of the contract to a new account (`newOwner`).
          * Can only be called by the current owner.
          */
        function transferOwnership(address newOwner) public virtual onlyOwner {
                require(newOwner != address(0), "Ownable: new owner is the zero address");
                emit OwnershipTransferred(_owner, newOwner);
                _owner = newOwner;
        }
}


pragma solidity ^0.6.0;

/**
  * @dev Library for managing
  * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
  * types.
  *
  * Sets have the following properties:
  *
  * - Elements are added, removed, and checked for existence in constant time
  * (O(1)).
  * - Elements are enumerated in O(n). No guarantees are made on the ordering.
```

```
 *
 * ```
 * contract Example {
 *       // Add the library methods
 *       using EnumerableSet for EnumerableSet.AddressSet;
 *
 *       // Declare a set state variable
 *       EnumerableSet.AddressSet private mySet;
 * }
 * ```
 *
 * As of v3.0.0, only sets of type `address` (`AddressSet`) and `uint256`
 * (`UintSet`) are supported.
 */
library EnumerableSet {
    // To implement this library for multiple types with as little code
    // repetition as possible, we write it in terms of a generic Set type with
    // bytes32 values.
    // The Set implementation uses private functions, and user-facing
    // implementations (such as AddressSet) are just wrappers around the
    // underlying Set.
    // This means that we can only create new EnumerableSets for types that fit
    // in bytes32.

    struct Set {
        // Storage of set values
        bytes32[] _values;

        // Position of the value in the `values` array, plus 1 because index 0
        // means a value is not in the set.
        mapping(bytes32 => uint256) _indexes;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function _add(Set storage set, bytes32 value) private returns (bool) {
        if (!_contains(set, value)) {
            set._values.push(value);
            // The value is stored at length-1, but we add 1 to all indexes
            // and use 0 as a sentinel value
            set._indexes[value] = set._values.length;
            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function _remove(Set storage set, bytes32 value) private returns (bool) {
        // We read and store the value's index to prevent multiple reads from the same storage slot
        uint256 valueIndex = set._indexes[value];

        if (valueIndex != 0) {// Equivalent to contains(set, value)
            // To delete an element from the _values array in O(1), we swap the element to delete with the last
 one in
            // the array, and then remove the last element (sometimes called as 'swap and pop').
            // This modifies the order of the array, as noted in {at}.

            uint256 toDeleteIndex = valueIndex - 1;
            uint256 lastIndex = set._values.length - 1;

            // When the value to delete is the last one, the swap operation is unnecessary. However, since this
 occurs
            // so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' statement.

            bytes32 lastvalue = set._values[lastIndex];

            // Move the last value to the index where the value to delete is
            set._values[toDeleteIndex] = lastvalue;
            // Update the index for the moved value
            set._indexes[lastvalue] = toDeleteIndex + 1;
            // All indexes are 1-based

            // Delete the slot where the moved value was stored
            set._values.pop();

            // Delete the index for the deleted slot
            delete set._indexes[value];
```

```
                return true;
            } else {
                return false;
            }
        }

        /**
         * @dev Returns true if the value is in the set. O(1).
         */
        function _contains(Set storage set, bytes32 value) private view returns (bool) {
            return set._indexes[value] != 0;
        }

        /**
         * @dev Returns the number of values on the set. O(1).
         */
        function _length(Set storage set) private view returns (uint256) {
            return set._values.length;
        }

        /**
         * @dev Returns the value stored at position `index` in the set. O(1).
         *
         * Note that there are no guarantees on the ordering of values inside the
         * array, and it may change when more values are added or removed.
         *
         * Requirements:
         *
         * - `index` must be strictly less than {length}.
         */
        function _at(Set storage set, uint256 index) private view returns (bytes32) {
            require(set._values.length > index, "EnumerableSet: index out of bounds");
            return set._values[index];
        }

        // AddressSet

        struct AddressSet {
            Set _inner;
        }

        /**
         * @dev Add a value to a set. O(1).
         *
         * Returns true if the value was added to the set, that is if it was not
         * already present.
         */
        function add(AddressSet storage set, address value) internal returns (bool) {
            return _add(set._inner, bytes32(uint256(value)));
        }

        /**
         * @dev Removes a value from a set. O(1).
         *
         * Returns true if the value was removed from the set, that is if it was
         * present.
         */
        function remove(AddressSet storage set, address value) internal returns (bool) {
            return _remove(set._inner, bytes32(uint256(value)));
        }

        /**
         * @dev Returns true if the value is in the set. O(1).
         */
        function contains(AddressSet storage set, address value) internal view returns (bool) {
            return _contains(set._inner, bytes32(uint256(value)));
        }

        /**
         * @dev Returns the number of values in the set. O(1).
         */
        function length(AddressSet storage set) internal view returns (uint256) {
            return _length(set._inner);
        }

        /**
         * @dev Returns the value stored at position `index` in the set. O(1).
         *
         * Note that there are no guarantees on the ordering of values inside the
         * array, and it may change when more values are added or removed.
         *
         * Requirements:
         *
         * - `index` must be strictly less than {length}.
         */
        function at(AddressSet storage set, uint256 index) internal view returns (address) {
```

```
        return address(uint256(_at(set._inner, index)));
    }


    // UintSet

    struct UintSet {
        Set _inner;
    }

    /**
     * @dev Add a value to a set. O(1).
     *
     * Returns true if the value was added to the set, that is if it was not
     * already present.
     */
    function add(UintSet storage set, uint256 value) internal returns (bool) {
        return _add(set._inner, bytes32(value));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(UintSet storage set, uint256 value) internal view returns (bool) {
        return _contains(set._inner, bytes32(value));
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
     * Note that there are no guarantees on the ordering of values inside the
     * array, and it may change when more values are added or removed.
     *
     * Requirements:
     *
     * - `index` must be strictly less than {length}.
     */
    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
        return uint256(_at(set._inner, index));
    }
}
interface IVIP is IERC20 {
    function mint(address to, uint256 amount) external returns (bool);
}
interface IMasterChefBsc {
    function pending(uint256 pid, address user) external view returns (uint256);

    function deposit(uint256 pid, uint256 amount) external;

    function withdraw(uint256 pid, uint256 amount) external;

    function emergencyWithdraw(uint256 pid) external;
}

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

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

    // Info of each user.
    struct UserInfo {
        uint256 amount;          // How many LP tokens the user has provided.
        uint256 rewardDebt; // Reward debt.
        uint256 multLpRewardDebt; //multLp Reward debt.
    }
    // Info of each pool.
    struct PoolInfo {
        IERC20 lpToken;                  // Address of LP token contract.
        uint256 allocPoint;           // How many allocation points assigned to this pool. VIPs to distribute per
block.
        uint256 lastRewardBlock;    // Last block number that VIPs distribution occurs.
        uint256 accVipPerShare; // Accumulated VIPs per share, times 1e12.
        uint256 accMultLpPerShare; //Accumulated multLp per share
        uint256 totalAmount;        // Total amount of current pool deposit.
    }

    // The VIP Token!
    IVIP public vip;
    // VIP tokens created per block.
    uint256 public vipPerBlock;
    // Info of each pool.
    PoolInfo[] public poolInfo;
    // Info of each user that stakes LP tokens.
    mapping(uint256 => mapping(address => UserInfo)) public userInfo;
    // Corresponding to the pid of the multLP pool
    mapping(uint256 => uint256) public poolCorrespond;
    // pid corresponding address
    mapping(address => uint256) public LpOfPid;
    // Control mining
    bool public paused = false;
    // Total allocation points. Must be the sum of all allocation points in all pools.
    uint256 public totalAllocPoint = 0;
    // The block number when VIP mining starts.
    uint256 public startBlock;
    // multLP MasterChef
    address public multLpChef;
    // multLP Token
    address public multLpToken;
    // How many blocks are halved
    uint256 public halvingPeriod = 5256000;

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

    constructor(
        IVIP _vip,
        uint256 _vipPerBlock,
        uint256 _startBlock
    ) public {
        vip = _vip;
        vipPerBlock = _vipPerBlock;
        startBlock = _startBlock;
    }

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

    // Set the number of vip produced by each block
    function setVipPerBlock(uint256 _newPerBlock) public onlyOwner {
        massUpdatePools();
        vipPerBlock = _newPerBlock;
    }

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

    function addMultLP(address _addLP) public onlyOwner returns (bool) {
        require(_addLP != address(0), "LP is the zero address");
        IERC20(_addLP).approve(multLpChef, uint256(- 1));
        return EnumerableSet.add(_multLP, _addLP);
    }

    function isMultLP(address _LP) public view returns (bool) {
        return EnumerableSet.contains(_multLP, _LP);
    }

    function getMultLPLength() public view returns (uint256) {
        return EnumerableSet.length(_multLP);
    }

    function getMultLPAddress(uint256 _pid) public view returns (address){
        require(_pid <= getMultLPLength() - 1, "not find this multLP");
        return EnumerableSet.at(_multLP, _pid);
    }

    function setPause() public onlyOwner {
        paused = !paused;
    }
```

```
function setMultLP(address _multLpToken, address _multLpChef) public onlyOwner {
        require(_multLpToken != address(0) && _multLpChef != address(0), "is the zero address");
        multLpToken = _multLpToken;
        multLpChef = _multLpChef;
}

function replaceMultLP(address _multLpToken, address _multLpChef) public onlyOwner {
        require(_multLpToken != address(0) && _multLpChef != address(0), "is the zero address");
        require(paused == true, "No mining suspension");
        multLpToken = _multLpToken;
        multLpChef = _multLpChef;
        uint256 length = getMultLPLength();
        while (length > 0) {
                address dAddress = EnumerableSet.at(_multLP, 0);
                uint256 pid = LpOfPid[dAddress];
                IMasterChefBsc(multLpChef).emergencyWithdraw(poolCorrespond[pid]);
                EnumerableSet.remove(_multLP, dAddress);
                length--;
        }
}

// Add a new lp to the pool. Can only be called by the owner.
// XXX DO NOT add the same LP token more than once. Rewards will be messed up if you do.
function add(uint256 _allocPoint, IERC20 _lpToken, bool _withUpdate) public onlyOwner {
        require(address(_lpToken) != address(0), "_lpToken is the zero address");
        if (_withUpdate) {
                massUpdatePools();
        }
        uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
        poolInfo.push(PoolInfo({
        lpToken : _lpToken,
        allocPoint : _allocPoint,
        lastRewardBlock : lastRewardBlock,
        accVipPerShare : 0,
        accMultLpPerShare : 0,
        totalAmount : 0
        }));
        LpOfPid[address(_lpToken)] = poolLength() - 1;
}

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

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

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

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

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

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

    // Update reward variables of the given pool to be up-to-date.
    function updatePool(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        if (block.number <= pool.lastRewardBlock) {
            return;
        }
        uint256 lpSupply;
        if (isMultLP(address(pool.lpToken))) {
            if (pool.totalAmount == 0) {
                pool.lastRewardBlock = block.number;
                return;
            }
            lpSupply = pool.totalAmount;
        } else {
            lpSupply = pool.lpToken.balanceOf(address(this));
            if (lpSupply == 0) {
                pool.lastRewardBlock = block.number;
                return;
            }
        }
        uint256 blockReward = getVipBlockReward(pool.lastRewardBlock);
        if (blockReward <= 0) {
            return;
        }
        uint256 vipReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
        bool minRet = vip.mint(address(this), vipReward);
        if (minRet) {
            pool.accVipPerShare = pool.accVipPerShare.add(vipReward.mul(1e12).div(lpSupply));
        }
        pool.lastRewardBlock = block.number;
    }

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

    function pendingVipAndToken(uint256 _pid, address _user) private view returns (uint256, uint256){
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accVipPerShare = pool.accVipPerShare;
        uint256 accMultLpPerShare = pool.accMultLpPerShare;
        if (user.amount > 0) {
            uint256      TokenPending      =      IMasterChefBsc(multLpChef).pending(poolCorrespond[_pid],
address(this));
            accMultLpPerShare = accMultLpPerShare.add(TokenPending.mul(1e12).div(pool.totalAmount));
            uint256                              userPending                              =
user.amount.mul(accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
            if (block.number > pool.lastRewardBlock) {
                uint256 blockReward = getVipBlockReward(pool.lastRewardBlock);
                uint256 vipReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
                accVipPerShare = accVipPerShare.add(vipReward.mul(1e12).div(pool.totalAmount));
                return (user.amount.mul(accVipPerShare).div(1e12).sub(user.rewardDebt), userPending);
            }
            if (block.number == pool.lastRewardBlock) {
                return (user.amount.mul(accVipPerShare).div(1e12).sub(user.rewardDebt), userPending);
            }
        }
        return (0, 0);
    }

    function pendingVip(uint256 _pid, address _user) private view returns (uint256){
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 accVipPerShare = pool.accVipPerShare;
        uint256 lpSupply = pool.lpToken.balanceOf(address(this));
        if (user.amount > 0) {
            if (block.number > pool.lastRewardBlock) {
                uint256 blockReward = getVipBlockReward(pool.lastRewardBlock);
                uint256 vipReward = blockReward.mul(pool.allocPoint).div(totalAllocPoint);
                accVipPerShare = accVipPerShare.add(vipReward.mul(1e12).div(lpSupply));
                return user.amount.mul(accVipPerShare).div(1e12).sub(user.rewardDebt);
            }
            if (block.number == pool.lastRewardBlock) {
                return user.amount.mul(accVipPerShare).div(1e12).sub(user.rewardDebt);
            }
        }
        return 0;
    }

    // Deposit LP tokens to BscPool for VIP allocation.
    function deposit(uint256 _pid, uint256 _amount) public notPause {
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            depositVipAndToken(_pid, _amount, msg.sender);
        } else {
            depositVip(_pid, _amount, msg.sender);
        }
    }

    function depositVipAndToken(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pendingAmount = user.amount.mul(pool.accVipPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeVipTransfer(_user, pendingAmount);
            }
            uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
            IMasterChefBsc(multLpChef).deposit(poolCorrespond[_pid], 0);
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare                                                    =
pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
            uint256                              tokenPending                         =
user.amount.mul(pool.accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
            if (tokenPending > 0) {
                IERC20(multLpToken).safeTransfer(_user, tokenPending);
            }
        }
        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(_user, address(this), _amount);
            if (pool.totalAmount == 0) {
                IMasterChefBsc(multLpChef).deposit(poolCorrespond[_pid], _amount);
                user.amount = user.amount.add(_amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
            } else {
                uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
                IMasterChefBsc(multLpChef).deposit(poolCorrespond[_pid], _amount);
                uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
                pool.accMultLpPerShare                                                =
pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
                user.amount = user.amount.add(_amount);
                pool.totalAmount = pool.totalAmount.add(_amount);
            }
        }
        user.rewardDebt = user.amount.mul(pool.accVipPerShare).div(1e12);
        user.multLpRewardDebt = user.amount.mul(pool.accMultLpPerShare).div(1e12);
        emit Deposit(_user, _pid, _amount);
    }

    function depositVip(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        updatePool(_pid);
        if (user.amount > 0) {
            uint256 pendingAmount = user.amount.mul(pool.accVipPerShare).div(1e12).sub(user.rewardDebt);
            if (pendingAmount > 0) {
                safeVipTransfer(_user, pendingAmount);
            }
        }
        if (_amount > 0) {
            pool.lpToken.safeTransferFrom(_user, address(this), _amount);
            user.amount = user.amount.add(_amount);
            pool.totalAmount = pool.totalAmount.add(_amount);
        }
        user.rewardDebt = user.amount.mul(pool.accVipPerShare).div(1e12);
        emit Deposit(_user, _pid, _amount);
    }

    // Withdraw LP tokens from BscPool.
    function withdraw(uint256 _pid, uint256 _amount) public notPause {
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            withdrawVipAndToken(_pid, _amount, msg.sender);
        } else {
            withdrawVip(_pid, _amount, msg.sender);
        }
    }

    function withdrawVipAndToken(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        require(user.amount >= _amount, "withdrawVipAndToken: not good");
        updatePool(_pid);
        uint256 pendingAmount = user.amount.mul(pool.accVipPerShare).div(1e12).sub(user.rewardDebt);
        if (pendingAmount > 0) {
            safeVipTransfer(_user, pendingAmount);
        }
        if (_amount > 0) {
            uint256 beforeToken = IERC20(multLpToken).balanceOf(address(this));
            IMasterChefBsc(multLpChef).withdraw(poolCorrespond[_pid], _amount);
            uint256 afterToken = IERC20(multLpToken).balanceOf(address(this));
            pool.accMultLpPerShare                                                          =
pool.accMultLpPerShare.add(afterToken.sub(beforeToken).mul(1e12).div(pool.totalAmount));
            uint256                                  tokenPending                           =
user.amount.mul(pool.accMultLpPerShare).div(1e12).sub(user.multLpRewardDebt);
            if (tokenPending > 0) {
                IERC20(multLpToken).safeTransfer(_user, tokenPending);
            }
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            pool.lpToken.safeTransfer(_user, _amount);
        }
        user.rewardDebt = user.amount.mul(pool.accVipPerShare).div(1e12);
        user.multLpRewardDebt = user.amount.mul(pool.accMultLpPerShare).div(1e12);
        emit Withdraw(_user, _pid, _amount);
    }

    function withdrawVip(uint256 _pid, uint256 _amount, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        require(user.amount >= _amount, "withdrawVip: not good");
        updatePool(_pid);
        uint256 pendingAmount = user.amount.mul(pool.accVipPerShare).div(1e12).sub(user.rewardDebt);
        if (pendingAmount > 0) {
            safeVipTransfer(_user, pendingAmount);
        }
        if (_amount > 0) {
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            pool.lpToken.safeTransfer(_user, _amount);
        }
        user.rewardDebt = user.amount.mul(pool.accVipPerShare).div(1e12);
        emit Withdraw(_user, _pid, _amount);
    }

    // Withdraw without caring about rewards. EMERGENCY ONLY.
    function emergencyWithdraw(uint256 _pid) public notPause {
        PoolInfo storage pool = poolInfo[_pid];
        if (isMultLP(address(pool.lpToken))) {
            emergencyWithdrawVipAndToken(_pid, msg.sender);
        } else {
            emergencyWithdrawVip(_pid, msg.sender);
        }
    }

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

    function emergencyWithdrawVip(uint256 _pid, address _user) private {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][_user];
        uint256 amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        pool.lpToken.safeTransfer(_user, amount);
        pool.totalAmount = pool.totalAmount.sub(amount);
        emit EmergencyWithdraw(_user, _pid, amount);
    }

    // Safe VIP transfer function, just in case if rounding error causes pool to not have enough VIPs.
    function safeVipTransfer(address _to, uint256 _amount) internal {
        uint256 vipBal = vip.balanceOf(address(this));
        if (_amount > vipBal) {
                vip.transfer(_to, vipBal);
            } else {
                vip.transfer(_to, _amount);
            }
        }

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

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 BNB 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 BNB 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 BNB 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

6.1 Manticore

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

6.2 Oyente

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

6.3 securify.sh

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

6.4 Echidna

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

6.5 MAIAN

MAIAN 是一个用于查找以太坊智能合约漏洞的自动化工具，Maian 处理合

约的字节码，并尝试建立一系列交易以找出并确认错误。

6.6 ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

6.7 ida-evm

ida-evm 是一个针对以太坊虚拟机（EVM）的 IDA 处理器模块。

6.8 Remix-ide

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

6.9 知道创宇区块链安全审计人员专用工具包

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

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话	+86(10)400 060 9587
邮　　箱	sec@knownsec.com
官　　网	www.knownsec.com
地　　址	北京市 朝阳区 望京 SOHO T2-B座-2509