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	20211124	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
UAT金融智能合约审计报告	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 1 1 月 22 日开始到 2021 年 1 1 月 24 日结束，在此期间针对 **U A T 金 融 智能合约代码**的安全性和规范性进行审计并以此作为报告统计依据。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三章节）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	UTA	
合约地址	A-	-
代码类型	BSC 智能合约代码	
代码语言	Solidity	

合约文件及哈希：

合约文件	MD5
Manageable.sol	e7123ac6b8307cd18b44feada0208b92
Ownable.sol	b612c63ba8080ead7a3ba7c32c9f915c

Context.sol	fa402a594278b4fcef72736b6c926df6
SafeMath.sol	d13139b051128153ec2be17aed05c28f
BEP20.sol	fbe3fcba3385e6ba05b0b00b06fac4de
IBEP20.sol	88c0461ffd9a0da7ffbac2e5cd307a3e
SafeBEP20.sol	246db24d4c276c12a3168a8e333235f0
Address.sol	b1753b371721624d258630bbe5576194
AddressStringUtil.sol	377bb63a7c9babe2a159664b4655738a
Create2.sol	115717a0d7638fb0ae565756eb4f96cc
EnumerableSet.sol	000e09950995544fc13e41f9bf50044c
FixedPoint.sol	4af5a1bceab303505639a95b5a647d31
Memory.sol	31ee1f13813295c1276e192bc520994a
PairNamer.sol	685beba6d936435bb176f1b8455018b3
ReentrancyGuard.sol	fb4b20cfcee6e5cd916ff9b0ed367e85
SafeBEP20Namer.sol	968133996315c8ceb97fc92dd27fdec7
TransferHelper.sol	a2cceb75b0e64101e526e87a73eb35dc
UATMinePool.sol	26489e441c6bc12c5c5b38d18551dea8
UATMiner.sol	bb4113356a8a17c7f97cfcdb62a4d0c0
UATMinerDefine.sol	fff095c344a23c23217d4541f864bbe3
UATtoken.sol	e34b6680b5f89187ad081260457c25f3
LpWallet.sol	314360b136a93253b2193f3a91f374e6

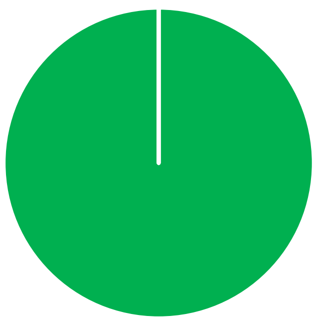
2. 代码漏洞分析

2.1 漏洞等级分布

本次漏洞风险按等级统计：

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	0	31

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	UATMiner 合约 addTradingPool 函数	通过	经检测，不存在安全问题。
	UATMiner 合约 buyVip 函数	通过	经检测，不存在安全问题。
	UATMiner 合约 WithDrawCredit 函数	通过	经检测，不存在安全问题。
	UATMinePool 合约 MineOut 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.orgin 身份验证	通过	经检测，不存在该安全问题。
	owner 权限控制	通过	经检测，不存在该安全问题。
	gas 消耗检测	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	增发代币漏洞	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。

	错误使用随机数检测	通过	经检测，不存在该安全问题。
	不安全的接口使用	通过	经检测，不存在该安全问题。
	变量覆盖	通过	经检测，不存在该安全问题。
	未初始化的存储指针	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	交易顺序依赖检测	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	通过	经检测，不存在该安全问题。
	拒绝服务攻击检测	通过	经检测，不存在该安全问题。
	假充值漏洞检测	通过	经检测，不存在该安全问题。
	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	重排攻击检测	通过	经检测，不存在该安全问题。

3. 业务安全性检测

3.1. UATMiner 合约 addTradingPool 函数【通过】

审计分析：对 UATMiner 合约中 addTradingPool 逻辑审计进行安全审计，检查调用者权限和对传入参数有效性进行检查，以及对 lpPools 更新是否合理

```
function addTradingPool(address tokenAddress,address tradecontract,uint256 rate,uint256
tokendecimals) public returns (bool)
    {
        require(msg.sender==_owner);
        require(rate > 0,"ERROR RATE");
        require(_lpPools[tokenAddress].hashrate==0,"LP EXISTS");

        LpWallet wallet = new LpWallet(tokenAddress,_Lizaddr,_feeowner);
        _lpPools[tokenAddress] = PoolInfo({
          poolwallet:wallet,
          hashrate:rate,
          tokenDecimals:tokendecimals,
          tradeContract:tradecontract
        });
        emit TradingPooladded(tokenAddress);
        return true;
    }
```

安全建议：无。

3.2. UATMiner 合约 buyVip 函数【通过】

审计分析：对 UATMiner 合约中 buyVip 逻辑审计进行安全审计，计算购买新VIP 等级的费用并检查费用有效性，扣除费用和更新 hashdiff、level。

```
function buyVip(uint newlevel) public nonReentrant returns (bool)
```

```
    {
        uint256 costcount=buyVipPrice(msg.sender,newlevel);
        require(costcount>0);
        IBEP20(_Lizaddr).burnFrom(msg.sender, costcount);
        uint256 hashdiff=getHashDiffOnLevelChange(msg.sender,newlevel);
        if(hashdiff > 0)
        {
            UserHashChanged(msg.sender,0,hashdiff,true);
            logCheckPoint(hashdiff,0,true);
        }
        _userInfos[msg.sender].userlevel=newlevel; //knownsec// 更新 level
        emit VipChanged(msg.sender,newlevel);
        return true;
    }
```

安全建议： 无。

3.3. UATMiner 合约 WithDrawCredit 函数【通过】

审计分析： 对 UATMiner 合约中 WithDrawCredit 逻辑审计进行安全审计，获取已产出未领取的 UAT，更新 userInfos 信息，销毁 1%手续费并提现已产出UAT。

```
function WithDrawCredit() public nonReentrant returns (bool)
    {
        uint256 amount = getPendingCoin(msg.sender);
        if(amount < 100) //防止算术溢出
            return true;
        _userInfos[msg.sender].pendingreward=0;
        _userInfos[msg.sender].lastblock=block.number;
        _userInfos[msg.sender].lastcheckpoint= _checkpoints.length -1;
        uint256 fee= amount.div(100);//手续费 1%销毁
        _minepool.MineOut(msg.sender, amount.sub(fee),fee);
        return true;
    }
```

安全建议：无。

3.4. UATMinePool 合约 MineOut 函数【通过】

审计分析：对 UATMinePool 合约中 MineOut 逻辑审计进行安全审计，检查调用者的权限，是否正确提现 amout 和销毁手续费。

```
unction MineOut(address to,uint256 amount,uint256 fee) public returns(bool){
            require(msg.sender==_owner);
            _token.safeTransfer(to, amount); //knownsec//提现 amount
            IBEP20(_token).burn(fee); //knownsec//销毁手续费
            return true;
    }
```

安全建议：无。

4. 代码基本漏洞检测

4.1. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本

检测结果：经检测，智能合约代码中制定了编译器版本 0.5.15 以上，不存在该安全问题。

安全建议：无。

4.2. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.3. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

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

安全建议：无。

4.4. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.5. require/assert 的合理使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.6. fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.7. tx.origin 身份验证【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.8. owner 权限控制【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.9. gas 消耗检测【通过】

检查 gas 的消耗是否超过区块最大限制

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.10. call 注入攻击【通过】

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

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.11. 低级函数安全【通过】

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

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.12. 增发代币漏洞【通过】

检查在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.13. 访问控制缺陷检测【通过】

合约中不同函数应设置合理的权限

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.14. 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢。

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

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.15. 算术精度误差【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.16. 错误使用随机数【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.17. 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.18. 变量覆盖【通过】

检查合约代码实现中是否存在变量覆盖导致的安全问题

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.19. 未初始化的储存指针【通过】

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

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

检测结果：经检测，智能合约代码不存在该问题。

安全建议：无。

4.20. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向某一地址发送 HT，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 B N B 发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.21. 交易顺序依赖【通过】

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

检测结果：经检测，BEP20.sol 中的_approve 函数存在事务顺序依赖攻击风险，代码如下：

```
function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal {
        require(owner != address(0), 'BEP20: approve from the zero address');
        require(spender != address(0), 'BEP20: approve to the zero address');//knownsec//此处存在事务顺序依赖风险,建议添加下面的语句进行判断
        //require((amount == 0) || (_allowance[owner][spender] == 0));
        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
```

可能存在的安全风险描述如下：

1. 用户 A 通过调用 approve 函数允许用户 B 代其转账的数量为 N（N>0）；
2. 经过一段时间后，用户 A 决定将 N 改为 M（M>0），所以再次调用 approve 函数；
3. 用户 B 在第二次调用被矿工处理之前迅速调用 transferFrom 函数转账 N 数量的 token；
4. 用户 A 对 approve 的第二次调用成功后，用户 B 便可再次获得 M 的转账额度，即用户 B 通过交易顺序攻击获得了 N+M 的转账额度。

安全建议：

1. 前端限制，当用户 A 将额度从 N 修改为 M 时，可先从 N 修改为 0，再从 0 修改为 M。
2. 在 approve 函数开头增加如下代码：

require((amount == 0) || (_allowance[owner][spender] == 0));

4.22. 时间戳依赖攻击【通过】

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

检查合约代码实现中是否存在有依赖于时间戳的关键功能

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.23. 拒绝服务攻击【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.24. 假充值漏洞【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.25. 重入攻击检测【通过】

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

前时，就会存在重入攻击的风险。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

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

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.27. 重排攻击检测【通过】

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

检测结果:经检测，智能合约代码中不存在相关漏洞。

安全建议:无。

5. 附录 A：合约代码

本次测试代码来源：

Manageable.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.4.0;

import '../GSN/Context.sol';

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

    event ManagementTransferred(address indexed previousManager, address indexed newManager);

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

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

    /**
     * @dev Throws if called by any account other than the manager.
     */
    modifier onlyManager() {
        require(_manager == _msgSender(), 'Manageable: caller is not the manager');
        _;
    }

    /**
     * @dev Leaves the contract without manager. It will not be possible to call
     * `onlyManager` functions anymore. Can only be called by the current manager.
     *
     * NOTE: Renouncing management will leave the contract without an manager,
     * thereby removing any functionality that is only available to the manager.
     */
    function renounceManagement() public onlyManager {
        emit ManagementTransferred(_manager, address(0));
        _manager = address(0);
    }

    /**
     * @dev Transfers management of the contract to a new account (`newManager`).
     * Can only be called by the current manager.
     */
    function transferManagement(address newManager) public onlyManager {
        _transferManagement(newManager);
    }

    /**
     * @dev Transfers management of the contract to a new account (`newManager`).
     */
    function _transferManagement(address newManager) internal {
        require(newManager != address(0), 'Manageable: new manager is the zero address');
        emit ManagementTransferred(_manager, newManager);
        _manager = newManager;
    }
}
```

Ownable.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.4.0;

import '../GSN/Context.sol';

/**
 * @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() {
        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 Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public onlyOwner {
        _transferOwnership(newOwner);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     */
    function _transferOwnership(address newOwner) internal {
        require(newOwner != address(0), 'Ownable: new owner is the zero address');
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}
```

Context.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.4.0;

/*
 * @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.
 */
contract Context {
	// Empty internal constructor, to prevent people from mistakenly deploying
	// an instance of this contract, which should be used via inheritance.
	constructor() {}

	function _msgSender() internal view returns (address) {
		return msg.sender;
	}

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

SafeMath.sol

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

pragma solidity >=0.4.0;

/**
 * @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;
    }

    function min(uint256 x, uint256 y) internal pure returns (uint256 z) {
        z = x < y ? x : y;
    }

    //                                   babylonian                                   method
(https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_method)
    function sqrt(uint256 y) internal pure returns (uint256 z) {
        if (y > 3) {
            z = y;
            uint256 x = y / 2 + 1;
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }
}
```

BEP20.sol

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

pragma solidity >=0.4.0;

import '../../access/Ownable.sol';
import '../../GSN/Context.sol';
import './IBEP20.sol';
import '../../math/SafeMath.sol';
import '../../utils/Address.sol';

/**
 * @dev Implementation of the {IBEP20} 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 {BEP20PresetMinterPauser}.
 *
 * TIP: For a detailed writeup see our guide
 * https://forum.zeppelin.solutions/t/how-to-implement-BEP20-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 BEP20 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 {IBEP20-approve}.
 */
contract BEP20 is Context, IBEP20, Ownable {
    using SafeMath for uint256;
    using Address for address;
    uint256 _maxsupply;
    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 namea, string memory symbola,uint8 decimalsa) {
        _name = namea;
        _symbol = symbola;
        _decimals = decimalsa;
        _maxsupply = 1000000000 *(10 **_decimals);
}

/**
  * @dev Returns the bep token owner.
  */
function getOwner() external override view returns (address) {
        return owner();
}

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

/**
  * @dev Returns the token decimals.
  */
function decimals() public override view returns (uint8) {
        return _decimals;
}

/**
  * @dev Returns the token symbol.
  */
function symbol() public override view returns (string memory) {
        return _symbol;
}

/**
  * @dev See {BEP20-totalSupply}.
  */
function totalSupply() public override view returns (uint256) {
        return _totalSupply;
}

/**
  * @dev See {BEP20-balanceOf}.
  */
function balanceOf(address account) public override view returns (uint256) {
        return _balances[account];
}

/**
  * @dev See {BEP20-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 override returns (bool) {
        _transfer(_msgSender(), recipient, amount);
        return true;
}

/**
  * @dev See {BEP20-allowance}.
  */
function allowance(address owner, address spender) public override view returns (uint256) {
        return _allowances[owner][spender];
}

/**
  * @dev See {BEP20-approve}.
  *
  * Requirements:
  *
  * - `spender` cannot be the zero address.
```

```
     */
    function approve(address spender, uint256 amount) public override returns (bool) {
        _approve(_msgSender(), spender, amount);
        return true;
    }

    /**
     * @dev See {BEP20-transferFrom}.
     *
     * Emits an {Approval} event indicating the updated allowance. This is not
     * required by the EIP. See the note at the beginning of {BEP20};
     *
     * 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 override returns (bool) {
        _transfer(sender, recipient, amount);
        _approve(
            sender,
            _msgSender(),
            _allowances[sender][_msgSender()].sub(amount, 'BEP20: 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 {BEP20-approve}.
     *
     * Emits an {Approval} event indicating the updated allowance.
     *
     * Requirements:
     *
     * - `spender` cannot be the zero address.
     */
    function increaseAllowance(address spender, uint256 addedValue) public 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 {BEP20-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 returns (bool) {
        _approve(
            _msgSender(),
            spender,
            _allowances[_msgSender()][spender].sub(subtractedValue, 'BEP20: decreased allowance below
zero')
        );
        return true;
    }

    /**
     * @dev Creates `amount` tokens and assigns them to `msg.sender`, increasing
     * the total supply.
     *
     * Requirements
     *
     * - `msg.sender` must be the token owner
     */
    function mint(uint256 amount) public onlyOwner returns (bool) {
        require(_totalSupply.add(amount) <= _maxsupply);
        _mint(_msgSender(), amount);
        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 {
        require(sender != address(0), 'BEP20: transfer from the zero address');
        require(recipient != address(0), 'BEP20: transfer to the zero address');

        _balances[sender] = _balances[sender].sub(amount, 'BEP20: 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 {
        require(account != address(0), 'BEP20: mint to the zero address');
        _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 {
        require(account != address(0), 'BEP20: burn from the zero address');

        _balances[account] = _balances[account].sub(amount, 'BEP20: 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 {
        require(owner != address(0), 'BEP20: approve from the zero address');
        require(spender != address(0), 'BEP20: approve to the zero address');//knownsec//此处存在事务顺序依
赖风险,建议添加下面的语句进行判断
        //require((amount == 0) || (_allowance[owner][spender] == 0));
        _allowances[owner][spender] = amount;
        emit Approval(owner, spender, amount);
    }
    function burn(uint256 amount) public override returns (bool)
    {
        _burn(msg.sender,amount);
        return true;
    }

    function burnFrom(address account, uint256 amount) public override returns (bool)
    {
        _burnFrom(account,amount);
        return true;
    }

    /**
     * @dev Destroys `amount` tokens from `account`.`amount` is then deducted
     * from the caller's allowance.
     *
     * See {_burn} and {_approve}.
     */
    function _burnFrom(address account, uint256 amount) internal {
        _burn(account, amount);
        _approve(
            account,
            _msgSender(),
            _allowances[account][_msgSender()].sub(amount, 'BEP20: burn amount exceeds allowance')
        );
    }
}
```

IBEP20.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.4.0;

interface IBEP20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the token decimals.
     */
    function decimals() external view returns (uint8);

    /**
     * @dev Returns the token symbol.
     */
    function symbol() external view returns (string memory);

    /**
     * @dev Returns the token name.
     */
    function name() external view returns (string memory);

    /**
     * @dev Returns the bep token owner.
     */
    function getOwner() external view returns (address);

    /**
     * @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);

    function burnFrom(address account, uint256 amount) external returns (bool);

    function burn(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);
}
```

SafeBEP20.sol

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

pragma solidity ^0.6.0;

import './IBEP20.sol';
import '../../math/SafeMath.sol';
import '../../utils/Address.sol';

/**
 * @title SafeBEP20
 * @dev Wrappers around BEP20 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 SafeBEP20 for IBEP20;` statement to your contract,
 * which allows you to call the safe operations as `token.safeTransfer(...)`, etc.
 */
library SafeBEP20 {
    using SafeMath for uint256;
    using Address for address;

    function safeTransfer(
        IBEP20 token,
        address to,
        uint256 value
    ) internal {
        _callOptionalReturn(token, abi.encodeWithSelector(token.transfer.selector, to, value));
    }

    function safeTransferFrom(
        IBEP20 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
     * {IBEP20-approve}, and its usage is discouraged.
     *
     * Whenever possible, use {safeIncreaseAllowance} and
     * {safeDecreaseAllowance} instead.
     */
    function safeApprove(
        IBEP20 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),
            'SafeBEP20: approve from non-zero to non-zero allowance'
        );
        _callOptionalReturn(token, abi.encodeWithSelector(token.approve.selector, spender, value));
    }

    function safeIncreaseAllowance(
        IBEP20 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(
        IBEP20 token,
        address spender,
        uint256 value
    ) internal {
        uint256 newAllowance = token.allowance(address(this), spender).sub(
            value,
            'SafeBEP20: 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(IBEP20 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, 'SafeBEP20: low-level call failed');
        if (returndata.length > 0) {
            // Return data is optional
            // solhint-disable-next-line max-line-length
            require(abi.decode(returndata, (bool)), 'SafeBEP20: BEP20 operation did not succeed');
        }
    }
}
```

Address.sol

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

pragma solidity >=0.5.0;

/**
 * @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);
            }
        }
    }
}
```

AddressStringUtil.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.5.0;

library AddressStringUtil {
    // converts an address to the uppercase hex string, extracting only len bytes (up to 20, multiple of 2)
    function toAsciiString(address addr, uint256 len) internal pure returns (string memory) {
        require(len % 2 == 0 && len > 0 && len <= 40, 'AddressStringUtil: INVALID_LEN');

        bytes memory s = new bytes(len);
        uint256 addrNum = uint256(addr);
        for (uint256 i = 0; i < len / 2; i++) {
            // shift right and truncate all but the least significant byte to extract the byte at position 19-i
            uint8 b = uint8(addrNum >> (8 * (19 - i)));
            // first hex character is the most significant 4 bits
            uint8 hi = b >> 4;
            // second hex character is the least significant 4 bits
            uint8 lo = b - (hi << 4);
            s[2 * i] = char(hi);
            s[2 * i + 1] = char(lo);
        }
        return string(s);
    }
```

```
        // hi and lo are only 4 bits and between 0 and 16
        // this method converts those values to the unicode/ascii code point for the hex representation
        // uses upper case for the characters
        function char(uint8 b) private pure returns (bytes1 c) {
                if (b < 10) {
                        return bytes1(b + 0x30);
                } else {
                        return bytes1(b + 0x37);
                }
        }
}
```

Create2.sol

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

pragma solidity >=0.5.16;

/**
  * @dev Helper to make usage of the `CREATE2` EVM opcode easier and safer.
  * `CREATE2` can be used to compute in advance the address where a smart
  * contract will be deployed, which allows for interesting new mechanisms known
  * as 'counterfactual interactions'.
  *
  * See the https://eips.ethereum.org/EIPS/eip-1014#motivation[EIP] for more
  * information.
  */
library Create2 {
        /**
          * @dev Deploys a contract using `CREATE2`. The address where the contract
          * will be deployed can be known in advance via {computeAddress}.
          *
          * The bytecode for a contract can be obtained from Solidity with
          * `type(contractName).creationCode`.
          *
          * Requirements:
          *
          * - `bytecode` must not be empty.
          * - `salt` must have not been used for `bytecode` already.
          * - the factory must have a balance of at least `amount`.
          * - if `amount` is non-zero, `bytecode` must have a `payable` constructor.
          */
        function deploy(
                uint256 amount,
                bytes32 salt,
                bytes memory bytecode
        ) internal returns (address) {
                address addr;
                require(address(this).balance >= amount, 'Create2: insufficient balance');
                require(bytecode.length != 0, 'Create2: bytecode length is zero');
                // solhint-disable-next-line no-inline-assembly
                assembly {
                        addr := create2(amount, add(bytecode, 0x20), mload(bytecode), salt)
                }
                require(addr != address(0), 'Create2: Failed on deploy');
                return addr;
        }

        /**
          * @dev Returns the address where a contract will be stored if deployed via {deploy}. Any change in the
          * `bytecodeHash` or `salt` will result in a new destination address.
          */
        function computeAddress(bytes32 salt, bytes32 bytecodeHash) internal view returns (address) {
                return computeAddress(salt, bytecodeHash, address(this));
        }

        /**
          * @dev Returns the address where a contract will be stored if deployed via {deploy} from a contract located
at
          * `deployer`. If `deployer` is this contract's address, returns the same value as {computeAddress}.
          */
        function computeAddress(
                bytes32 salt,
                bytes32 bytecodeHash,
                address deployer
        ) internal pure returns (address) {
                bytes32 _data = keccak256(abi.encodePacked(bytes1(0xff), deployer, salt, bytecodeHash));
                return address(uint256(_data));
        }
}
```

EnumerableSet.sol

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

```
pragma solidity >=0.5.0;

/**
 * @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));
    }
}
```

FixedPoint.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.4.0;

import '../math/SafeMath.sol';

// a library for handling binary fixed point numbers (https://en.wikipedia.org/wiki/Q_(number_format))
library FixedPoint {
    // range: [0, 2**112 - 1]
    // resolution: 1 / 2**112
    struct uq112x112 {
        uint224 _x;
    }

    // range: [0, 2**144 - 1]
    // resolution: 1 / 2**112
    struct uq144x112 {
        uint256 _x;
    }

    uint8 private constant RESOLUTION = 112;
    uint256 private constant Q112 = uint256(1) << RESOLUTION;
    uint256 private constant Q224 = Q112 << RESOLUTION;

    // encode a uint112 as a UQ112x112
    function encode(uint112 x) internal pure returns (uq112x112 memory) {
        return uq112x112(uint224(x) << RESOLUTION);
    }

    // encodes a uint144 as a UQ144x112
    function encode144(uint144 x) internal pure returns (uq144x112 memory) {
        return uq144x112(uint256(x) << RESOLUTION);
    }

    // divide a UQ112x112 by a uint112, returning a UQ112x112
    function div(uq112x112 memory self, uint112 x) internal pure returns (uq112x112 memory) {
        require(x != 0, 'FixedPoint: DIV_BY_ZERO');
        return uq112x112(self._x / uint224(x));
    }

    // multiply a UQ112x112 by a uint, returning a UQ144x112
    // reverts on overflow
    function mul(uq112x112 memory self, uint256 y) internal pure returns (uq144x112 memory) {
        uint256 z;
        require(y == 0 || (z = uint256(self._x) * y) / y == uint256(self._x), 'FixedPoint:
MULTIPLICATION_OVERFLOW');
        return uq144x112(z);
    }

    // returns a UQ112x112 which represents the ratio of the numerator to the denominator
    // equivalent to encode(numerator).div(denominator)
    function fraction(uint112 numerator, uint112 denominator) internal pure returns (uq112x112 memory) {
        require(denominator > 0, 'FixedPoint: DIV_BY_ZERO');
        return uq112x112((uint224(numerator) << RESOLUTION) / denominator);
    }

    // decode a UQ112x112 into a uint112 by truncating after the radix point
    function decode(uq112x112 memory self) internal pure returns (uint112) {
        return uint112(self._x >> RESOLUTION);
    }

    // decode a UQ144x112 into a uint144 by truncating after the radix point
    function decode144(uq144x112 memory self) internal pure returns (uint144) {
        return uint144(self._x >> RESOLUTION);
    }

    // take the reciprocal of a UQ112x112
    function reciprocal(uq112x112 memory self) internal pure returns (uq112x112 memory) {
        require(self._x != 0, 'FixedPoint: ZERO_RECIPROCAL');
        return uq112x112(uint224(Q224 / self._x));
    }

    // square root of a UQ112x112
    function sqrt(uq112x112 memory self) internal pure returns (uq112x112 memory) {
        return uq112x112(uint224(SafeMath.sqrt(uint256(self._x)) << 56));
    }
}
```

Memory.sol

```
pragma solidity >=0.5.0;

library Memory {

    // Size of a word, in bytes.
    uint internal constant WORD_SIZE = 32;
    // Size of the header of a 'bytes' array.
    uint internal constant BYTES_HEADER_SIZE = 32;
    // Address of the free memory pointer.
    uint internal constant FREE_MEM_PTR = 0x40;

    // Compares the 'len' bytes starting at address 'addr' in memory with the 'len'
    // bytes starting at 'addr2'.
    // Returns 'true' if the bytes are the same, otherwise 'false'.
    function equals(uint addr, uint addr2, uint len) internal pure returns (bool equal) {
        assembly {
            equal := eq(keccak256(addr, len), keccak256(addr2, len))
        }
    }

    // Compares the 'len' bytes starting at address 'addr' in memory with the bytes stored in
    // 'bts'. It is allowed to set 'len' to a lower value then 'bts.length', in which case only
    // the first 'len' bytes will be compared.
    // Requires that 'bts.length >= len'
    function equals(uint addr, uint len, bytes memory bts) internal pure returns (bool equal) {
        require(bts.length >= len);
        uint addr2;
        assembly {
            addr2 := add(bts, /*BYTES_HEADER_SIZE*/32)
        }
        return equals(addr, addr2, len);
    }

    function compareStrings(string memory a, string memory b) internal pure returns (bool) {
        return (keccak256(abi.encodePacked((a))) == keccak256(abi.encodePacked((b))));
    }

    // Copy 'len' bytes from memory address 'src', to address 'dest'.
    // This function does not check the or destination, it only copies
    // the bytes.
    function copy(uint src, uint dest, uint len) internal pure {
        // Copy word-length chunks while possible
        for (; len >= WORD_SIZE; len -= WORD_SIZE) {
            assembly {
                mstore(dest, mload(src))
            }
            dest += WORD_SIZE;
            src += WORD_SIZE;
        }

        // Copy remaining bytes
        uint mask = 256 ** (WORD_SIZE - len) - 1;
        assembly {
            let srcpart := and(mload(src), not(mask))
            let destpart := and(mload(dest), mask)
            mstore(dest, or(destpart, srcpart))
        }
    }

    // Returns a memory pointer to the provided bytes array.
    function ptr(bytes memory bts) internal pure returns (uint addr) {
        assembly {
            addr := bts
        }
    }

    // Returns a memory pointer to the data portion of the provided bytes array.
    function dataPtr(bytes memory bts) internal pure returns (uint addr) {
        assembly {
            addr := add(bts, /*BYTES_HEADER_SIZE*/32)
        }
    }

    // This function does the same as 'dataPtr(bytes memory)', but will also return the
    // length of the provided bytes array.
    function fromBytes(bytes memory bts) internal pure returns (uint addr, uint len) {
        len = bts.length;
        assembly {
            addr := add(bts, /*BYTES_HEADER_SIZE*/32)
        }
    }

    // Creates a 'bytes memory' variable from the memory address 'addr', with the
    // length 'len'. The function will allocate new memory for the bytes array, and
    // the 'len bytes starting at 'addr' will be copied into that new memory.
    function toBytes(uint addr, uint len) internal pure returns (bytes memory bts) {
        bts = new bytes(len);
        uint btsptr;
        assembly {
            btsptr := add(bts, /*BYTES_HEADER_SIZE*/32)
        }
        copy(addr, btsptr, len);
    }

    // Get the word stored at memory address 'addr' as a 'uint'.
    function toUint(uint addr) internal pure returns (uint n) {
        assembly {
            n := mload(addr)
        }
    }

    // Get the word stored at memory address 'addr' as a 'bytes32'.
    function toBytes32(uint addr) internal pure returns (bytes32 bts) {
        assembly {
            bts := mload(addr)
        }
    }
}
```

PairNamer.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.5.0;

import './SafeBEP20Namer.sol';

// produces names for pairs of tokens using Uniswap's naming scheme
library PairNamer {
    string private constant TOKEN_SYMBOL_PREFIX = '🦄';
    string private constant TOKEN_SEPARATOR = ':';

    // produces a pair descriptor in the format of `${prefix}${name0}:${name1}${suffix}`
    function pairName(
        address token0,
        address token1,
        string memory prefix,
        string memory suffix
    ) internal view returns (string memory) {
        return
            string(
                abi.encodePacked(
                    prefix,
                    SafeBEP20Namer.tokenName(token0),
                    TOKEN_SEPARATOR,
                    SafeBEP20Namer.tokenName(token1),
                    suffix
                )
            );
    }

    // produces a pair symbol in the format of `🦄${symbol0}:${symbol1}${suffix}`
    function pairSymbol(
        address token0,
        address token1,
        string memory suffix
    ) internal view returns (string memory) {
        return
            string(
                abi.encodePacked(
                    TOKEN_SYMBOL_PREFIX,
                    SafeBEP20Namer.tokenSymbol(token0),
                    TOKEN_SEPARATOR,
                    SafeBEP20Namer.tokenSymbol(token1),
                    suffix
                )
            );
    }
}
```

ReentrancyGuard.sol

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

pragma solidity >=0.5.0;

/**
 * @dev Contract module that helps prevent reentrant calls to a function.
 *
 * Inheriting from `ReentrancyGuard` will make the {nonReentrant} modifier
 * available, which can be applied to functions to make sure there are no nested
 * (reentrant) calls to them.
 *
 * Note that because there is a single `nonReentrant` guard, functions marked as
 * `nonReentrant` may not call one another. This can be worked around by making
 * those functions `private`, and then adding `external` `nonReentrant` entry
 * points to them.
 *
 * TIP: If you would like to learn more about reentrancy and alternative ways
 * to protect against it, check out our blog post
 * https://blog.openzeppelin.com/reentrancy-after-istanbul/[Reentrancy After Istanbul].
 */
contract ReentrancyGuard {
    // Booleans are more expensive than uint256 or any type that takes up a full
    // word because each write operation emits an extra SLOAD to first read the
    // slot's contents, replace the bits taken up by the boolean, and then write
    // back. This is the compiler's defense against contract upgrades and
    // pointer aliasing, and it cannot be disabled.

    // The values being non-zero value makes deployment a bit more expensive,
    // but in exchange the refund on every call to nonReentrant will be lower in
    // amount. Since refunds are capped to a percentage of the total
    // transaction's gas, it is best to keep them low in cases like this one, to
    // increase the likelihood of the full refund coming into effect.
    uint256 private constant _NOT_ENTERED = 1;
    uint256 private constant _ENTERED = 2;

    uint256 private _status;

    constructor () {
        _status = _NOT_ENTERED;
    }

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

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

        _;

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

SafeBEP20Namer.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.5.0;

import './AddressStringUtil.sol';

// produces token descriptors from inconsistent or absent BEP20 symbol implementations that can return string or
bytes32
// this library will always produce a string symbol to represent the token
library SafeBEP20Namer {
    function bytes32ToString(bytes32 x) private pure returns (string memory) {
        bytes memory bytesString = new bytes(32);
        uint256 charCount = 0;
        for (uint256 j = 0; j < 32; j++) {
            bytes1 char = x[j];
            if (char != 0) {
                bytesString[charCount] = char;
                charCount++;
            }
        }
        bytes memory bytesStringTrimmed = new bytes(charCount);
        for (uint256 j = 0; j < charCount; j++) {
            bytesStringTrimmed[j] = bytesString[j];
        }
        return string(bytesStringTrimmed);
    }

    // assumes the data is in position 2
    function parseStringData(bytes memory b) private pure returns (string memory) {
        uint256 charCount = 0;
        // first parse the charCount out of the data
        for (uint256 i = 32; i < 64; i++) {
            charCount <<= 8;
            charCount += uint8(b[i]);
        }

        bytes memory bytesStringTrimmed = new bytes(charCount);
        for (uint256 i = 0; i < charCount; i++) {
            bytesStringTrimmed[i] = b[i + 64];
        }

        return string(bytesStringTrimmed);
    }

    // uses a heuristic to produce a token name from the address
    // the heuristic returns the full hex of the address string in upper case
    function addressToName(address token) private pure returns (string memory) {
        return AddressStringUtil.toAsciiString(token, 40);
    }

    // uses a heuristic to produce a token symbol from the address
    // the heuristic returns the first 6 hex of the address string in upper case
    function addressToSymbol(address token) private pure returns (string memory) {
        return AddressStringUtil.toAsciiString(token, 6);
    }

    // calls an external view token contract method that returns a symbol or name, and parses the output into a
string
    function callAndParseStringReturn(address token, bytes4 selector) private view returns (string memory) {
        (bool success, bytes memory data) = token.staticcall(abi.encodeWithSelector(selector));
        // if not implemented, or returns empty data, return empty string
        if (!success || data.length == 0) {
            return '';
        }
        // bytes32 data always has length 32
        if (data.length == 32) {
            bytes32 decoded = abi.decode(data, (bytes32));
            return bytes32ToString(decoded);
        } else if (data.length > 64) {
            return abi.decode(data, (string));
        }
        return '';
    }

    // attempts to extract the token symbol. if it does not implement symbol, returns a symbol derived from the
address
    function tokenSymbol(address token) internal view returns (string memory) {
        // 0x95d89b41 = bytes4(keccak256("symbol()"))
        string memory symbol = callAndParseStringReturn(token, 0x95d89b41);
        if (bytes(symbol).length == 0) {
            // fallback to 6 uppercase hex of address
            return addressToSymbol(token);
        }
        return symbol;
    }

    // attempts to extract the token name. if it does not implement name, returns a name derived from the address
    function tokenName(address token) internal view returns (string memory) {
        // 0x06fdde03 = bytes4(keccak256("name()"))
        string memory name = callAndParseStringReturn(token, 0x06fdde03);
        if (bytes(name).length == 0) {
            // fallback to full hex of address
            return addressToName(token);
        }
        return name;
    }
}
```

TransferHelper.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.5.0;

// helper methods for interacting with BEP20 tokens and sending ETH that do not consistently return true/false
library TransferHelper {
    function safeApprove(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper:
APPROVE_FAILED');
    }

    function safeTransfer(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper:
TRANSFER_FAILED');
    }

    function safeTransferFrom(
        address token,
        address from,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'TransferHelper:
TRANSFER_FROM_FAILED');
    }
    function safeTransferBNB(address to, uint256 value) internal {
        (bool success, ) = to.call{value: value}(new bytes(0));
        require(success, 'TransferHelper: BNB_TRANSFER_FAILED');
    }
}
```

LizMinePool.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity >=0.5.0;
import "./lib/utils/TransferHelper.sol";
import "./lib/token/BEP20/IBEP20.sol";

contract LizMinePool
{
    address _owner;
    address _token;
    using TransferHelper for address;

    constructor(address tokenaddress)
    {
        _owner=msg.sender;
        _token=tokenaddress;
    }

    function MineOut(address to,uint256 amount,uint256 fee) public returns(bool){
        require(msg.sender==_owner);
        _token.safeTransfer(to, amount); //knownsec//提现 amount
        IBEP20(_token).burn(fee); //knownsec//销毁手续费
        return true;
    }
}
```

LizMiner.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity >=0.5.0;

import "./lib/math/SafeMath.sol";
import "./lib/utils/ReentrancyGuard.sol";
import "./lib/utils/TransferHelper.sol";
import "./lib/token/BEP20/IBEP20.sol";
import "./LizMinerDefine.sol";
import "./LpWallet.sol";
import "./LizMinePool.sol";


contract LizMiner is ReentrancyGuard,LizMinerDefine {
    using TransferHelper for address;
    using SafeMath for uint256;
    address private _Lizaddr;
    address private _Liztrade;
    address private _bnbtradeaddress;
    address private _wrappedbnbaddress;
    address private _usdtaddress;
    address private _owner;
    address private _feeowner;
    LizMinePool private _minepool;

    struct PoolInfo {
        LpWallet poolwallet;
        uint256 hashrate;   //    The LP hashrate
        uint256 tokenDecimals;
        address tradeContract;
    }

    mapping(uint=>uint256[20]) internal _levelconfig; //credit level config
    uint256[11] _vipbuyprice =[0,200,400,600,800,1000,1500,2000,3000,4000,5000];
    CheckPoint[] _checkpoints;

    uint256 _nowtotalhash;
    uint256 _nowjthash;
    mapping(address=>mapping(address=>uint256)) _userLphash;
    mapping(address=>mapping(uint=>uint256)) _userlevelhashtotal; // level hash in my team
    mapping(address=>mapping(address=>uint256)) _userTeamDetail;
    mapping(address=>address) internal _parents;//Inviter
    mapping(address=>UserInfo) _userInfos;
    mapping(address=>PoolInfo) _lpPools;
    event BindingParents(address indexed user,address    inviter);
    event VipChanged(address indexed user,uint256    userlevel);
    event TradingPooladded(address indexed tradetoken);
    event UserBuied(address indexed tokenaddress,uint256 amount);
    event TakedBack(address indexed tokenaddress,uint256 pct);

    constructor()
    {
        _owner=msg.sender;
    }

    function getMinerPoolAddress() public view returns(address)
    {
        return address(_minepool);
    }

    function InitalContract(address lizToken,address liztrade,address wrappedbnbaddress,address
bnbtradeaddress,address usdtaddress,address feeowner) public
    {
        require(msg.sender==_owner);
        require(_checkpoints.length==0);
        _Lizaddr=lizToken;
        _Liztrade=liztrade;
        _bnbtradeaddress=bnbtradeaddress;
        _usdtaddress=usdtaddress;
        _wrappedbnbaddress= wrappedbnbaddress;
        _nowjthash=1;
        _nowtotalhash=1;
        _feeowner= feeowner;
        _minepool = new LizMinePool(lizToken,feeowner);
         _parents[msg.sender] = address(_minepool);
        _checkpoints.push(CheckPoint({startblock:block.number,totalhash:1,selfhashtotal:1}));
        _levelconfig[0] = [100,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        _levelconfig[1] = [150,100,50,30,10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        _levelconfig[2] = [180,120,60,36,12,12,0,0,0,0,0,0,0,0,0,0,0,0,0,0];
        _levelconfig[3] = [195,130,65,39,13,13,13,13,0,0,0,0,0,0,0,0,0,0,0,0];
        _levelconfig[4] = [210,140,70,42,14,14,14,14,14,14,0,0,0,0,0,0,0,0,0,0];
        _levelconfig[5] = [225,150,75,45,15,15,15,15,15,15,15,15,0,0,0,0,0,0,0,0];
        _levelconfig[6] = [240,160,80,48,16,16,16,16,16,16,16,16,16,16,0,0,0,0,0,0];
        _levelconfig[7] = [255,170,85,51,17,17,17,17,17,17,17,17,17,17,17,17,0,0,0,0];
        _levelconfig[8] = [270,180,90,54,18,18,18,18,18,18,18,18,18,18,18,18,18,18,0,0];
        _levelconfig[9] = [285,190,95,57,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19,19];
        _levelconfig[10] = [300,200,100,60,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20,20];
    }


    function addTradingPool(address tokenAddress,address tradecontract,uint256 rate,uint256 tokendecimals)
public returns (bool)
    {
        require(msg.sender==_owner);
        require(rate > 0,"ERROR RATE");
        require(_lpPools[tokenAddress].hashrate==0,"LP EXISTS");

        LpWallet wallet = new LpWallet(tokenAddress,_Lizaddr,_feeowner);
        _lpPools[tokenAddress] = PoolInfo({
         poolwallet:wallet,
         hashrate:rate,
         tokenDecimals:tokendecimals,
         tradeContract:tradecontract
        });
        emit TradingPooladded(tokenAddress);
        return true;
    }

     //******************Getters ******************/
    function getOwner() public view returns(address) {
        return _owner;
    }

    function getParent(address user) public view returns(address)
    {
        return _parents[user];
    }

    function CurrentBlockReward() public view returns (uint256)
    {
        return OneBlockReward(_nowjthash);
    }

    function OneBlockReward(uint256 totalhash) public pure returns (uint256)
    {
        if(totalhash < 10000000 * 1000000 )
            return   totalhash.div(100000);
        else
            return 100000000;
    }

     function getTotalHash() public view returns (uint256)
     {
         return _nowtotalhash;
     }

     function getTotalLphash() public view returns(uint256)
     {
        return _nowjthash;
    }

    function getMyLpInfo(address user,address tokenaddress) public view returns (uint256[3] memory )
    {
        uint256[3] memory bb;
        bb[0]= _lpPools[tokenaddress].poolwallet.getBalance(user,true);
        bb[1]= _lpPools[tokenaddress].poolwallet.getBalance(user,false);
        bb[2]= _userLphash[user][tokenaddress];
        return bb;
    }

    function getUserLevel(address user) public view returns (uint)
    {
        return _userInfos[user].userlevel;
    }

    function getUserTeamHash(address user) public view returns (uint256)
    {
        return _userInfos[user].teamhash;
    }

    function getUserSelfHash(address user) public view returns (uint256)
    {
        return _userInfos[user].selfhash;
    }

    function getFeeOnwer() public view returns (address)
    {
        return _feeowner;
    }

    function getTeamHashFromUser(address leader,address user) public view returns (uint256)
    {
        return _userTeamDetail[leader][user];
    }

    function getExchangeCountOfOneUsdt(address lptoken) public view returns (uint256)
    {
        require(_lpPools[lptoken].tradeContract !=address(0));

        if(lptoken == address(2))//BNB
        {
            uint256 balancea=IBEP20(_wrappedbnbaddress).balanceOf(_bnbtradeaddress);
            uint256 balanceb=IBEP20(_usdtaddress).balanceOf(_bnbtradeaddress);

            if(balancea==0 || balanceb==0)
                return 0;

            return balancea.mul(1000000).div(balanceb);
        }
        else
        {
            uint256 balancea=IBEP20(_wrappedbnbaddress).balanceOf(_bnbtradeaddress);
            uint256 balanceb=IBEP20(_usdtaddress).balanceOf(_bnbtradeaddress);

            uint256 balancec= IBEP20(lptoken).balanceOf(_lpPools[lptoken].tradeContract);
            uint256 balanced=IBEP20(_wrappedbnbaddress).balanceOf(_lpPools[lptoken].tradeContract);
            if(balancea==0 || balanceb==0 || balanced==0)
                return 0;
            return balancec.mul(1000000).div(balanceb.mul(balanced).div(balancea));
        }
    }

    function buyVipPrice(address user,uint newlevel) public view returns (uint256)
    {
        require(newlevel <11,"ERROR LEVEL");
        uint256 userlevel=_userInfos[user].userlevel;
        if(userlevel >= newlevel)
            return 0;
        uint256 costprice=_vipbuyprice[newlevel] - _vipbuyprice[userlevel];
        uint256 costcount=costprice.mul(getExchangeCountOfOneUsdt(_Lizaddr));
        return costcount;
    }

    //*****************Getters ***********************************/
    function getWalletAddress(address lptoken) public view returns (address)
    {
        return address(_lpPools[lptoken].poolwallet);
    }

    function logCheckPoint(uint256 totalhashdiff,uint256 jthash,bool add) private
    {
        if(add)
        {
            _nowjthash = _nowjthash.add(jthash);
            _nowtotalhash= _nowtotalhash.add(totalhashdiff);
        }
        else
        {
            _nowjthash = _nowjthash.sub(jthash);
            _nowtotalhash= _nowtotalhash.sub(totalhashdiff);
        }

        if(_checkpoints.length > 0)
        {
            if(block.number > _checkpoints[_checkpoints.length -1].startblock)
_checkpoints.push(CheckPoint({startblock:block.number,totalhash:_nowtotalhash,selfhashtotal:_nowjthash}));
            else
            {
                _checkpoints[_checkpoints.length -1].totalhash = _nowtotalhash;
                _checkpoints[_checkpoints.length -1].selfhashtotal = _nowjthash;
            }
        }
        else
        {
_checkpoints.push(CheckPoint({startblock:block.number,totalhash:_nowtotalhash,selfhashtotal:_nowjthash}));
        }
    }

    function getHashDiffOnLevelChange(address user,uint newlevel) private view returns (uint256)
    {
         uint256 hashdiff=0;
        uint userlevel = _userInfos[user].userlevel;
        for(uint i=0;i<20;i++)
        {
            if(_userlevelhashtotal[user][i] > 0)
            {
                if(_levelconfig[userlevel][i] >0)
                {
                    uint256
dff=_userlevelhashtotal[user][i].mul(_levelconfig[newlevel][i]).div(_levelconfig[userlevel][i]);
                    hashdiff = hashdiff.add(dff);
                }
                else
                {
                    uint256 dff=_userlevelhashtotal[user][i].mul(_levelconfig[newlevel][i]);
                    hashdiff = hashdiff.add(dff);
                }
            }
        }
        return hashdiff;
    }


    function buyVip(uint newlevel) public nonReentrant returns (bool)
    {
        uint256 costcount=buyVipPrice(msg.sender,newlevel);
        require(costcount>0);
        IBEP20(_Lizaddr).burnFrom(msg.sender, costcount);
        uint256 hashdiff=getHashDiffOnLevelChange(msg.sender,newlevel);
        if(hashdiff > 0)
        {
            UserHashChanged(msg.sender,0,hashdiff,true);
            logCheckPoint(hashdiff,0,true);
        }
        _userInfos[msg.sender].userlevel=newlevel;
        emit VipChanged(msg.sender,newlevel);
        return true;
    }

    function bindParent(address parent) public
    {
        require(_parents[msg.sender]==address(0),"Already bind");
        require(parent !=address(0),"ERROR parent");
        require(parent !=msg.sender,"error parent");
        require(_parents[parent]!=address(0));
        _parents[msg.sender]=parent;
        emit BindingParents(msg.sender,parent);
    }


    function    getPendingCoin(address user) public view returns(uint256)
    {
        if(_userInfos[user].lastblock==0)
        {
            return 0;
        }

        uint256 total= _userInfos[user].pendingreward;
        uint256 lastblock = _userInfos[user].lastblock;
        uint256 mytotalhash=_userInfos[user].selfhash.add(_userInfos[user].teamhash);
        for(uint i= _userInfos[user].lastcheckpoint + 1;i<_checkpoints.length;i++ )
        {
            uint256 blockcount = _checkpoints[i].startblock-lastblock;
            uint256 oneblock = OneBlockReward(_checkpoints[i -1].selfhashtotal);
            if(_checkpoints[i -1].totalhash >0)
            {
                uint256 get= blockcount.mul(oneblock).mul(mytotalhash).div(_checkpoints[i -1].totalhash);
                total = total.add(get);
            }

            lastblock= _checkpoints[i].startblock;
        }

        if(lastblock < block.number)
        {
            uint256 blockcount = block.number-lastblock;
            uint256 oneblock = OneBlockReward(_nowjthash);
            if(_nowtotalhash > 0)
            {
                uint256 get= blockcount.mul(oneblock).mul(mytotalhash).div(_nowtotalhash);
                total = total.add(get);
            }
        }
        return total;
    }

    function UserHashChanged(address user,uint256 selfhash,uint256 teamhash,bool add) private
    {
        UserInfo memory info = _userInfos[user];
        info.pendingreward= getPendingCoin(user);
        info.lastblock= block.number;
        info.lastcheckpoint=_checkpoints.length -1;
        if(selfhash >0)
        {
            if(add)
            {
                info.selfhash= info.selfhash.add(selfhash);
            }
            else
                info.selfhash= info.selfhash.sub(selfhash);
        }
        if(teamhash > 0)
        {
            if(add)
            {
                info.teamhash= info.teamhash.add(teamhash);
            }
            else
                info.teamhash= info.teamhash.sub(teamhash);

        }
        _userInfos[user]=info;
    }


    function WithDrawCredit() public nonReentrant returns (bool)
    {
        uint256 amount = getPendingCoin(msg.sender);
        if(amount < 100)
            return true;
        _userInfos[msg.sender].pendingreward=0;
        _userInfos[msg.sender].lastblock=block.number;
        _userInfos[msg.sender].lastcheckpoint= _checkpoints.length -1;
        uint256 fee= amount.div(100);
        _minepool.MineOut(msg.sender, amount.sub(fee),fee);
        return true;
    }

    function TakeBack(address tokenAddress,uint256 pct) public nonReentrant returns (bool)
    {
        require(pct >=10000 &&pct <=1000000);
        require(_lpPools[tokenAddress].poolwallet.getBalance(msg.sender,true)        >=        10000,"ERROR
AMOUNT");
        uint256 balancea=_lpPools[tokenAddress].poolwallet.getBalance(msg.sender,true);
        uint256 balanceb=_lpPools[tokenAddress].poolwallet.getBalance(msg.sender,false);
        uint256 totalhash=_userLphash[msg.sender][tokenAddress];

        uint256 amounta= balancea.mul(pct).div(1000000);
        uint256 amountb= balanceb.mul(pct).div(1000000);
        uint256 decreasehash= _userLphash[msg.sender][tokenAddress].mul(pct).div(1000000);

         if(balanceb.sub(amountb) <= 10000)
        {
            decreasehash=totalhash;
            amounta=balancea;
            amountb=balanceb;
```

```
                _userLphash[msg.sender][tokenAddress]=0;

            }else
            {
                _userLphash[msg.sender][tokenAddress]= totalhash.sub(decreasehash);
            }
            _nowjthash= _nowjthash.sub(decreasehash);
            uint256 hashdiff=decreasehash;
            address parent=msg.sender;

            for(uint i=0;i<20;i++)
            {
                parent = _parents[parent];
                if(parent==address(0))
                    break;
                uint256 parentlevel= _userInfos[parent].userlevel;
                uint256 oldhash= userTeamDetail[parent][msg.sender];
                if(totalhash > oldhash)
                {
                    continue;
                }
                uint256 diffhash=decreasehash;

                if(totalhash.add(decreasehash)> oldhash)
                    diffhash= oldhash.sub(totalhash);

                if(diffhash > 0)
                {
                    _userTeamDetail[parent][msg.sender] = _userTeamDetail[parent][msg.sender].sub(diffhash);
                    uint256 pdechash= diffhash.mul(_levelconfig[parentlevel][i]).div(1000);
                    hashdiff=hashdiff.add(pdechash);
                    UserHashChanged(parent,0,pdechash,false);
                }
            }
            UserHashChanged(msg.sender,decreasehash,0,false);
            logCheckPoint(hashdiff,decreasehash,false);
            _lpPools[tokenAddress].poolwallet.TakeBack(msg.sender,amounta,amountb);
            if(tokenAddress==address(2))
            {
               (bool success, ) = msg.sender.call{value: amounta}(new bytes(0));
                require(success, 'TransferHelper: BNB_TRANSFER_FAILED');
                if(amountb>=100)
               {
                   uint256 fee = amountb.div(100);//Destory 1%
                   _Lizaddr.safeTransfer(msg.sender, amountb.sub(fee));
                   _Lizaddr.safeTransfer(_feeowner,fee);
               }
               else
               {
                   _Lizaddr.safeTransfer(msg.sender, amountb);
               }
            }
            emit TakedBack(tokenAddress,pct);
            return true;
        }

        function getPower(address tokenAddress,uint256 amount,uint lpscale) public view returns (uint256)
        {
            uint256                                                                          hashb=
amount.mul(1000000).mul(100).div(lpscale).div(getExchangeCountOfOneUsdt(tokenAddress));
            return hashb;
        }

        function getLpPayLiz(address tokenAddress,uint256 amount,uint lpscale) public view returns (uint256)
        {
            require(lpscale<=100);
            uint256                                                                          hashb=
amount.mul(1000000).mul(100).div(lpscale).div(getExchangeCountOfOneUsdt(tokenAddress));
            uint256      costabc      =          hashb.mul(getExchangeCountOfOneUsdt(_Lizaddr)).mul(100      -
lpscale).div(1000000 * 100);
            return costabc;
        }

        function depositNFT(uint256 amount) public nonReentrant payable returns (bool)
        {


        }

        function deposit(address tokenAddress,uint256 amount,uint dppct) public nonReentrant payable returns (bool)
        {
            if(tokenAddress==address(2))
            {
                amount = msg.value;
            }
            require(amount > 10000);
        require(dppct<=100);
        uint256 hashb= getPower(tokenAddress,amount,dppct);
        uint256 costliz =    hashb.mul(getExchangeCountOfOneUsdt(_Lizaddr)).mul(100 - dppct).div(1000000 *
100);
        uint256 abcbalance=IBEP20(_Lizaddr).balanceOf(msg.sender);


        if(abcbalance < costliz)
        {
              require(tokenAddress!=address(2),"Not enough liz balance");
            amount = amount.mul(abcbalance).div(costliz);
            hashb= hashb.mul(abcbalance).div(costliz);
            costliz= abcbalance;
        }

        if(tokenAddress==address(2))
        {
             if(costliz>0)
                _Lizaddr.safeTransferFrom(msg.sender, address(this), costliz);
        }
        else
        {
            tokenAddress.safeTransferFrom(msg.sender, address(_lpPools[tokenAddress].poolwallet), amount);
            if(costliz>0)
                _Lizaddr.safeTransferFrom(msg.sender, address(_lpPools[tokenAddress].poolwallet), costliz);
        }

        _lpPools[tokenAddress].poolwallet.addBalance(msg.sender,amount,costliz);
        _userLphash[msg.sender][tokenAddress] = _userLphash[msg.sender][tokenAddress].add(hashb);
        _nowjthash= _nowjthash.add(hashb);

        uint256 hashdiff=hashb;
        address parent=msg.sender;
        uint256 userhash=_userInfos[msg.sender].selfhash;

        for(uint i=0;i<20;i++)
        {
            parent = _parents[parent];
            if(parent==address(0))
                break;

            uint256 parentlevel= _userInfos[parent].userlevel;
            uint256 parenthash = _userInfos[parent].selfhash;
            uint256 hashbase=hashb;

            if(userhash.add(hashb) >=parenthash)
            {
                if(userhash<parenthash)
                {
                    hashbase= parenthash.sub(userhash);
                }
                else
                    hashbase=0;
            }

            if(hashbase==0)
            {
                continue;
            }
             _userTeamDetail[parent][msg.sender] = _userTeamDetail[parent][msg.sender].add(hashbase);
            if(_levelconfig[parentlevel][i] > 0)
            {
                uint256 addhash= hashbase.mul(_levelconfig[parentlevel][i]).div(1000);
                hashdiff=hashdiff.add(addhash);
                UserHashChanged(parent,0,addhash,true);
            }
        }
        UserHashChanged(msg.sender,hashb,0,true);
        logCheckPoint(hashdiff,hashb,true);
        emit UserBuied(tokenAddress,amount);
        return true;
    }

}
```

LizMinerDefine.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity >=0.5.0;

contract LizMinerDefine
{
    struct UserInfo
    {
        uint256 selfhash;//user hash total count
        uint256 teamhash;
        uint256 userlevel; // my userlevel
        uint256 pendingreward;
        uint256 lastcheckpoint;
        uint256 lastblock;
    }

    struct CheckPoint
    {
        uint256 startblock;
        uint256 totalhash;
        uint256 selfhashtotal;
    }
}
```

Liztoken.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity >=0.5.0;

import "./lib/token/BEP20/BEP20.sol";

// ABCToken with Governance.
contract DSCPToken is BEP20('DSCP', 'DSCP',8) {

    // /// @notice Creates `_amount` token to `_to`. Must only be called by the owner
    // function mint(address _to, uint256 _amount) public onlyOwner {
    //      _mint(_to, _amount);
    // }

    constructor()
    {
        _mint(msg.sender, 50000000* 100000000);
    }

}
```

LpWallet.sol

```
// SPDX-License-Identifier: GPL-3.0-or-later
pragma solidity >=0.5.0;
import "./lib/utils/TransferHelper.sol";
import "./lib/math/SafeMath.sol";
import "./lib/token/BEP20/IBEP20.sol";

contract LpWallet //EMPTY CONTRACT TO HOLD THE USERS assetS
{
    address lptoken;
    address liztoken;
    address _MainContract;

    mapping(address=>uint256) _balancesa;
    mapping(address=>uint256) _balancesb;

    using TransferHelper for address;
    using SafeMath for uint256;

    event eventWithDraw(address indexed to,uint256 indexed    amounta,uint256 indexed amountb);

    constructor(address tokena,address tokenb) //Create by lizmain
    {
        _MainContract=msg.sender;// The lizmain CONTRACT
        lptoken =tokena;
        liztoken=tokenb;
    }

    function getBalance(address user,bool isa) public view returns(uint256)
    {
        if(isa)
            return _balancesa[user];
       else
           return _balancesb[user];
    }

    function addBalance(address user,uint256 amounta,uint256 amountb) public
    {
        require(_MainContract==msg.sender);//Only lizmain can do this
        _balancesa[user] = _balancesa[user].add(amounta);
        _balancesb[user] = _balancesb[user].add(amountb);
    }

    function TakeBack(address to,uint256 amounta,uint256 amountb) public
    {
        require(_MainContract==msg.sender);//Only ABCmain can do this
        _balancesa[to]= _balancesa[to].sub(amounta);
        _balancesb[to]= _balancesb[to].sub(amountb);
        if(lptoken!= address(2))//BNB
        {
            lptoken.safeTransfer(to, amounta);
```

```
                if(amountb>=100) //knownsec//amountb 大于等于 100 时需要销毁 1%
                {
                    uint256 fee = amountb.div(100);//销毁 1%
                    liztoken.safeTransfer(to, amountb.sub(fee));
                    IBEP20(liztoken).burn(fee);
                }
                else
                {
                    liztoken.safeTransfer(to, amountb);
                }
            }
        }
    }
```

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 B N B 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 HT 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 B N B 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