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	20210924	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
GeFi 智能合约审计报告	V1.0	d92c064aa911435ba5d1dd02e14c2b79	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 9 月 16 日开始到 2021 年 9 月 24 日结束，在此期间针对 **GeFi 智能合约代码**的机枪池的安全性和规范性进行审计并以此作为报告统计依据。

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

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第三章节）进行了全面的分析，综合评定为**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：d92c064aa911435ba5d1dd02e14c2b79

报告查询地址链接:

https://attest.im/attestation/searchResult?qurey=d92c064aa911435ba5d1dd02e14c2b79

本次审计的目标信息：

条目	描述	
项目名称	Ge.Finance	
合约地址	Token(GEG)	0x99dAB6065951BecaC1dECBaC0C1A16b9BbF12913

	Token(GES)	0xfE34bd1C5FF5D1d1e7BF01d66f5B225E53F3D67f
	sGEGToken	0x8C024e31AD57435C7842334D405033199323f3DD
	UniswapReward(SGEG-GEG)	0x72fd8B7d965EB1E5d1cF5FD7DC8b30Cc12591eA7
	MoboxToken	0x3203c9E46cA618C8C1cE5dC67e7e9D75f5da2377
	UniswapReward(GEG-Mbox)	0xb9258333E4536Ccda964FC997c15F7659552fB62
	UniswapReward(GEG-WBNB)	0x80B5C9c4326D33F5c36dB1E8CF623463eF4Dc68E
	UniswapReward(GEG-BUSD)	0xC66A9E0459458D2A8cD4660c72e2005FEEE1241F
	UniswapReward(GEG-BNB)	0xA56339f711A82f6bB22A91c7855ab268559149D1
	UniswapReward(cake-bnb)	0x89d4e8511ED7566887C0b4F6E612d62293f99232
	UniswapReward(BUSD-bnb)	0x93d8800A82C06efaF8c3662D988e369D9270313E
	UniswapReward(axs-bnb)	0xbC2013E2437FbAAfb86468C38eBE4D95EFA5C9Bd
	UniswapReward(skill-bnb)	0xe5854375ebE4C07c894deaB7C6975268a1A788de

	UniswapReward(mbox-bnb)	0xC2e6d61fbBf77f5ad3Fc328bA296cA99Adb0ecf2
代码类型	BSC 智能合约代码	
代码语言	Solidity	

合约文件及哈希：

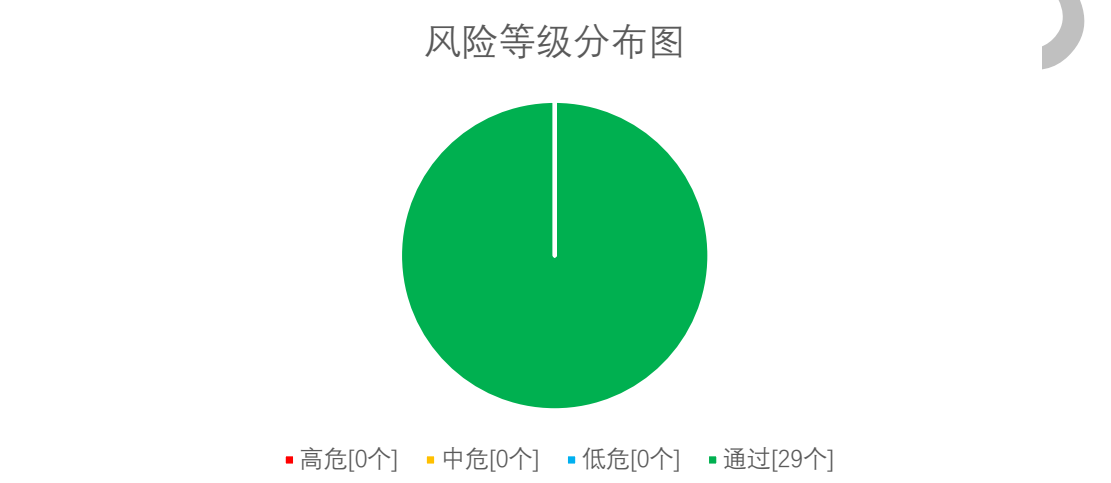
合约文件	MD5
gegtoken.sol	79b711de890886b1ebbd11b7122d3fdd
stake.sol	2db6c980b31ababbe752b598c329ed3e

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	29



2.2 审计结果汇总说明

审计结果			
审计项目	审计内容	状态	描述
业务安全性检测	代币功能	通过	经检测，不存在安全问题。
	质押提款功能	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. 代币功能【通过】

审计分析：代币功能在 gegtoken.sol 中实现，用于授权、铸币、转账和设置相关属性。该功能合理，函数不存在安全问题。

```
function approve(address spender, uint256 amount) public
    returns (bool)
    {
        require(msg.sender != address(0), "ERC20: approve from the zero address");
        require(spender != address(0), "ERC20: approve to the zero address");

        _allowances[msg.sender][spender] = amount;
        emit Approval(msg.sender, spender, amount);

        return true;
    }

    /**
    * @dev Function to check the amount of tokens than an owner _allowed to a spender.
    * @param owner address The address which owns the funds.
    * @param spender address The address which will spend the funds.
    * @return A uint256 specifying the amount of tokens still available for the spender.
    */
    function allowance(address owner, address spender) public view
    returns (uint256)
    {
        return _allowances[owner][spender];
    }

    /**
```

```
* @dev Gets the balance of the specified address.
* @param owner The address to query the the balance of.
* @return An uint256 representing the amount owned by the passed address.
*/
function balanceOf(address owner) public    view
returns (uint256)
{
    return _balances[owner];
}

/**
* @dev return the token total supply
*/
function totalSupply() public view
returns (uint256)
{
    return _totalSupply;
}

/**
* @dev for mint function
*/
function mint(address account, uint256 amount) public //knownsec//  挖矿
{
    require(account != address(0), "ERC20: mint to the zero address");
    require(_minters[msg.sender], "!minter"); //knownsec//  调用者是矿工
    require(_minters_number[msg.sender]>=amount);
    uint256 curMintSupply = _totalSupply.add(_totalBurnToken);
    uint256 newMintSupply = curMintSupply.add(amount);
    require( newMintSupply <= _maxSupply,"supply is max!");

    _totalSupply = _totalSupply.add(amount);
    _balances[account] = _balances[account].add(amount);
```

```
        _minters_number[msg.sender] = _minters_number[msg.sender].sub(amount);
        emit Mint(address(0), account, amount);
        emit Transfer(address(0), account, amount);
    }

    function addMinter(address _minter,uint256 number) public onlyGovernance
    {
        _minters[_minter] = true;
        _minters_number[_minter] = number;
    }


    function setMinter_number(address _minter,uint256 number) public onlyGovernance
    {
        require(_minters[_minter]);
        _minters_number[_minter] = number;
    }

    function removeMinter(address _minter) public onlyGovernance
    {
        _minters[_minter] = false;
        _minters_number[_minter] = 0;
    }


    function() external payable {
        revert();
    }

    /**
    * @dev for govern value
    */
    function setRate(uint256 burn_rate, uint256 reward_rate) public
```

```
        onlyGovernance
    {

        require(_maxGovernValueRate >= burn_rate && burn_rate >=
_minGovernValueRate,"invalid burn rate");
        require(_maxGovernValueRate >= reward_rate && reward_rate >=
_minGovernValueRate,"invalid reward rate");

        _burnRate = burn_rate;
        _rewardRate = reward_rate;

        emit eveSetRate(burn_rate, reward_rate);
    }




    /**
    * @dev for set reward
    */
    function setRewardPool(address rewardPool,address burnPool) public
        onlyGovernance
    {
        require(rewardPool != address(0x0));
        require(burnPool != address(0x0));
        _rewardPool = rewardPool;
        _burnPool = burnPool;

        emit eveRewardPool(_rewardPool,_burnPool);
    }
    /**
    * @dev transfer token for a specified address
    * @param to The address to transfer to.
```

```
 * @param value The amount to be transferred.
 */
function transfer(address to, uint256 value) public
returns (bool)
{
        return _transfer(msg.sender,to,value);
 }

 /**
 * @dev Transfer tokens from one address to another
 * @param from address The address which you want to send tokens from
 * @param to address The address which you want to transfer to
 * @param value uint256 the amount of tokens to be transferred
 */
 function transferFrom(address from, address to, uint256 value) public
 returns (bool)
 {
        uint256 allow = _allowances[from][msg.sender];
        _allowances[from][msg.sender] = allow.sub(value);

        return _transfer(from,to,value);
 }


 /**
 * @dev Transfer tokens with fee
 * @param from address The address which you want to send tokens from
 * @param to address The address which you want to transfer to
 * @param value uint256s the amount of tokens to be transferred
 */
 function _transfer(address from, address to, uint256 value) internal
 returns (bool)
```

```
{
    // :)
    require(_openTransfer || from == governance, "transfer closed");

    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");

    uint256 sendAmount = value;
    uint256 burnFee = (value.mul(_burnRate)).div(_rateBase); //knownsec// 计算销毁费用
    if (burnFee > 0) {
        //to burn
        _balances[_burnPool] = _balances[_burnPool].add(burnFee);
        _totalSupply = _totalSupply.sub(burnFee);
        sendAmount = sendAmount.sub(burnFee);

        _totalBurnToken = _totalBurnToken.add(burnFee);

        emit Transfer(from, _burnPool, burnFee);
    }

    uint256 rewardFee = (value.mul(_rewardRate)).div(_rateBase); //knownsec// 计算奖励
费用
    if (rewardFee > 0) {
        //to reward
        _balances[_rewardPool] = _balances[_rewardPool].add(rewardFee);
        sendAmount = sendAmount.sub(rewardFee);

        _totalRewardToken = _totalRewardToken.add(rewardFee);

        emit Transfer(from, _rewardPool, rewardFee);
    }

    _balances[from] = _balances[from].sub(value);
```

```
        _balances[to] = _balances[to].add(sendAmount);

        emit Transfer(from, to, sendAmount);

        return true;
    }
```

安全建议：无。

3.2. 质押提款功能【通过】

审计分析：质押提款功能在 stake.sol 中实现，用于用户质押和提款代币。该功能合理，函数不存在安全问题。

```
function stake(uint256 amount, string memory affCode) public { //knownsec// 质押
        _totalSupply = _totalSupply.add(amount);
        _balances[msg.sender] = _balances[msg.sender].add(amount);

        if( _powerStrategy != address(0x0)){ //knownsec// 地址不为 0
            _totalPower = _totalPower.sub(_powerBalances[msg.sender]);
            IPowerStrategy(_powerStrategy).lpIn(msg.sender, amount);

            _powerBalances[msg.sender] =
IPowerStrategy(_powerStrategy).getPower(msg.sender);
            _totalPower = _totalPower.add(_powerBalances[msg.sender]);
        }else{
            _totalPower = _totalSupply;
            _powerBalances[msg.sender] = _balances[msg.sender];
        }

        _lpToken.safeTransferFrom(msg.sender, address(this), amount);

    }
```

```
    function withdraw(uint256 amount) public { //knownsec// 提款
        require(amount > 0, "amout > 0");

        _totalSupply = _totalSupply.sub(amount);
        _balances[msg.sender] = _balances[msg.sender].sub(amount);

        if( _powerStrategy != address(0x0)){
            _totalPower = _totalPower.sub(_powerBalances[msg.sender]);
            IPowerStrategy(_powerStrategy).lpOut(msg.sender, amount);
            _powerBalances[msg.sender] = IPowerStrategy(_powerStrategy).getPower(msg.sender);
            _totalPower = _totalPower.add(_powerBalances[msg.sender]);

        }else{
            _totalPower = _totalSupply;
            _powerBalances[msg.sender] = _balances[msg.sender];
        }

        _lpToken.safeTransfer(_admina_address, amount.mul(3).div(1000)); //knownsec// 千分之3的手续费给管理员
        _lpToken.safeTransfer(_msg.sender, amount.mul(997).div(1000)); //knownsec// 千分之997转给用户
    }
```

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 BNB，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；

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

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.21. 交易顺序依赖【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.22. 时间戳依赖攻击【通过】

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

检查合约代码实现中是否存在有依赖于时间戳的关键功能

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.23. 拒绝服务攻击【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.24. 假充值漏洞【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.25. 重入攻击检测【通过】

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

额之前时，就会存在重入攻击的风险。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

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

检测结果：经检测，智能合约未使用 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：安全风险评级标准

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

6. 附录 B：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