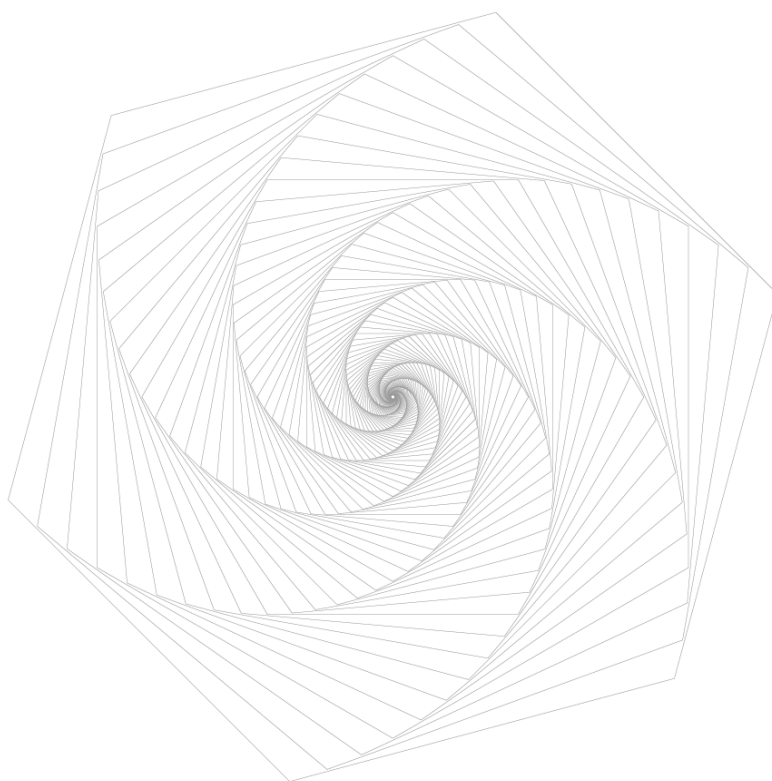




知道创宇

区块链安全实验室

智能合约审计报告



版本说明

修订内容	时间	修订者	版本号
编写文档	20220309	知道创宇区块链安全实验室	V1.0

文档信息

文档名称	文档版本	报告编号	保密级别
ROD 智能合约审计报告	V1.0	194661d2e69947a993571f862d6db ebc	项目组公开

声明

创宇区块链安全实验室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，创宇区块链安全实验室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区块链安全实验室提供的文件和资料。创宇区块链安全实验室假设：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，创宇区块链安全实验室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	- 6 -
2. 项目信息	- 7 -
2.1. 项目描述	- 7 -
2.2. 项目官网	- 7 -
2.3. 白皮书	- 7 -
2.4. 审核版本代码	- 7 -
2.5. 合约文件及哈希/合约部署地址	- 7 -
3. 外部可见性分析	- 8 -
3.1. RODToken 合约	- 8 -
4. 代码漏洞分析	- 11 -
4.1. 审计结果汇总说明	- 11 -
5. 业务安全性检测	- 13 -
5.1. 代币功能【通过】	- 13 -
5.2. 白名单账户设置功能【通过】	- 15 -
5.3. 转账功能【提示】	- 16 -
6. 代码基本漏洞检测	- 20 -
6.1. 编译器版本安全【通过】	- 20 -
6.2. 冗余代码【通过】	- 20 -
6.3. 安全算数库的使用【通过】	- 20 -
6.4. 不推荐的编码方式【通过】	- 20 -

6.5. require/assert 的合理使用【通过】	- 21 -
6.6. fallback 函数安全【通过】	- 21 -
6.7. tx.origin 身份验证【通过】	- 21 -
6.8. owner 权限控制【通过】	- 21 -
6.9. gas 消耗检测【通过】	- 22 -
6.10. call 注入攻击【通过】	- 22 -
6.11. 低级函数安全【通过】	- 22 -
6.12. 增发代币漏洞【通过】	- 22 -
6.13. 访问控制缺陷检测【通过】	- 23 -
6.14. 数值溢出检测【通过】	- 23 -
6.15. 算术精度误差【通过】	- 23 -
6.16. 错误使用随机数【通过】	- 24 -
6.17. 不安全的外部调用【通过】	- 24 -
6.18. 变量覆盖【通过】	- 24 -
6.19. 未初始化的储存指针【通过】	- 25 -
6.20. 返回值调用验证【通过】	- 25 -
6.21. 交易顺序依赖【通过】	- 26 -
6.22. 时间戳依赖攻击【通过】	- 26 -
6.23. 拒绝服务攻击【通过】	- 27 -
6.24. 假充值漏洞【通过】	- 27 -
6.25. 重入攻击检测【通过】	- 27 -
6.26. 重放攻击检测【通过】	- 28 -

6.27. 重排攻击检测【通过】 - 28 -

7. 附录 A：合约资金管理安全评估 - 29 -

Knownsec

1. 综述

本次报告有效测试时间是从 2022 年 3 月 5 日开始到 2022 年 3 月 9 日结束，在此期间针对 **ROD 智能合约的 RODToken 代币代码安全性和规范性**进行审计并以此作为报告统计依据。

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

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第六章节）以及合约具体业务安全项进行了全面的分析，综合评定为**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。

创宇存证信息:

类别	信息
报告编号	194661d2e69947a993571f862d6dbebc
报告查询链接	https://attest.im/attestation/searchResult?query=194661d2e69947a993571f862d6dbebc

2. 项目信息

2.1. 项目描述

ROD 是币安智能链上的数字代币，满足双币种储存与管理的便捷，ROD 与 BSC-USDT 可以自由兑换，全新金融的兑换机制构建完美的生态，达到生态最终的需求。完全匿名交易。财产自由，零知识点的交易方式，真正地自由掌握自己的财产，便捷安全。

2.2. 项目官网

<https://bscrod.github.io>

2.3. 白皮书

<https://bscrod.github.io/whitePaper.pdf>

2.4. 审核版本代码

<https://bscscan.com/address/0x9a829d93b956193bb3c28182e72d1052f3ec4893#code>

2.5. 合约文件及哈希/合约部署地址

合约文件	MD5
ROD0303. sol	b5e7b0f176a4f8785f9b8eaa67d910ca

3. 外部可见性分析

3.1. RODToken 合约

RODToken					
函数名	可见性	状态修改	修饰器	可支付接收	说明
name	public	False	---	---	---
symbol	public	False	---	---	---
decimals	public	False	---	---	---
totalSupply	public	False	override	---	---
balanceOf	public	False	override	---	---
transfer	public	True	override	---	---
allowance	public	False	override	---	---
approve	public	True	override	---	---
transferFrom	public	True	override	---	---
increaseAllowance	public	True	virtual	---	---
decreaseAllowance	public	True	virtual	---	---
isExcludedFromReward	public	False	---	---	---
totalFees	public	False	---	---	---
deliver	public	True	---	---	---
reflectionFromToken	public	False	---	---	---

tokenFromReflection	public	False	---	---	---
excludeFromReward	public	True	onlyOwner	---	---
includeInReward	external	True	onlyOwner	---	---
_transferBothExcluded	private	True	---	---	---
excludeFromFee	public	True	onlyOwner	---	---
includeInFee	public	True	onlyOwner	---	---
_reflectFee	private	True	---	---	---
_getValues	private	False	---	---	---
_getTVValues	private	False	---	---	---
_getRVValues	private	False	---	---	---
_getRate	private	False	---	---	---
_getCurrentSupply	private	False	---	---	---
_takeLiquidity	private	True	---	---	---
calculateTaxFee	private	False	---	---	---
calculateLiquidityFee	private	False	---	---	---
removeAllFee	private	True	---	---	---
restoreAllFee	private	True	---	---	---
isExcludedFromFee	public	False	---	---	---
_approve	private	True	---	---	---
_transfer	private	True	---	---	---

setMarketingWall et	external	True	onlyOwner	---	---
swapAndMarketi ng	private	True	lockTheSwap	---	---
swapTokensForU sdt	private	True	---	---	---
_tokenTransfer	private	True	---	---	---
_transferStandar d	private	True	---	---	---
_transferToExclu ded	private	True	---	---	---
_transferFromEx cluded	private	True	---	---	---
saveStuckedToke n	public	True	onlyOwner	---	---
sweep	external	True	onlyOwner	---	---

4. 代码漏洞分析

4.1. 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。
	白名单账户设置功能	通过	经检测，不存在安全问题。
	转账功能	提示	经检测，不存在安全问题，但事件监听错误。
代码基本漏洞检测	编译器版本安全	通过	经检测，不存在该安全问题。
	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。
	require/assert 的合理使用	通过	经检测，不存在该安全问题。
	fallback 函数安全	通过	经检测，不存在该安全问题。
	tx.origin 身份验证	通过	经检测，不存在该安全问题。
	owner 权限控制	通过	经检测，不存在该安全问题。
	gas 消耗检测	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	增发代币漏洞	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。
	错误使用随机数检测	通过	经检测，不存在该安全问题。
	不安全的外部调用	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。

KNOWNS

5. 业务安全性检测

5.1. 代币功能【通过】

审计分析：合约 RODToken.sol 中，代币继承自 IERC20 合约类，符合 ERC20 代币合约标准。经审计，代币功能完整，逻辑设计合理。

```
contract RODToken is Context, IERC20, Ownable {
    using SafeMath for uint256;
    using Address for address;

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

    mapping (address => bool) private _isExcludedFromFee;
    mapping (address => bool) private _isExcluded;
    address[] private _excluded;

    uint256 private constant MAX = ~uint256(0);
    uint256 private _tTotal = 2 * 10**8 * 10**18;
    uint256 private _rTotal = (MAX - (MAX % _tTotal));
    uint256 private _tFeeTotal;

    string private _name = "ROD"; // knownsec 代币名称
    string private _symbol = "ROD"; // knownsec 代币符号
    uint8 private _decimals = 18; // knownsec 代币精度

    uint256 public _taxFee = 3;
    uint256 private _previousTaxFee = _taxFee;

    uint256 public _liquidityFee = 7;
```

```

uint256 private _previousLiquidityFee = _liquidityFee;

IUniswapV2Router02 public uniswapV2Router;
address public uniswapV2Pair;

bool inSwapAndLiquify;
bool public swapAndLiquifyEnabled = true;

uint256 private numTokensSellToAddToLiquidity = 1 * 10**18;
IERC20 public usdt = IERC20(0x55d398326f99059fF775485246999027B3197955);

address payable marketingWalletAddress =
0x3fbe0540b69aAF7883cCAf5E9b3B85E570Bb0003;
address payable deadAddress = 0x00000000000000000000000000000000dEaD;

event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
event SwapAndLiquifyEnabledUpdated(bool enabled);
event SwapAndLiquify(
    uint256 tokensSwapped,
    uint256 ethReceived,
    uint256 tokensIntoLiquidity
);

modifier lockTheSwap {
    inSwapAndLiquify = true;
    _;
    inSwapAndLiquify = false;
}

constructor () public { // knownsec 构造函数
    _rOwned[_msgSender()] = _rTotal;

    IUniswapV2Router02 _uniswapV2Router =

```

```
IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);

    // Create a uniswap pair for this new token

    uniswapV2Pair =
    IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this), address(usdt));

    // set the rest of the contract variables

    uniswapV2Router = _uniswapV2Router;

    //exclude owner and this contract from fee

    _isExcludedFromFee[owner()] = true;
    _isExcludedFromFee[address(this)] = true;

    emit Transfer(address(0), _msgSender(), _tTotal);
}
```

安全建议：无。

5.2. 白名单账户设置功能【通过】

审计分析：合约 RODToken.sol 中，通过 _isExcluded[account] 判断是否为白名单账户，Owner 通过调用 excludeFromReward 和 includeInReward 实现设置该用户是否为白名单功能。经审计，权限控制正确，逻辑设计合理。

```
function excludeFromReward(address account) public onlyOwner() {// knownsec 设置用户白名单

    require(!_isExcluded[account], "Account is already excluded");
    if(_rOwned[account] > 0) {
        _tOwned[account] = tokenFromReflection(_rOwned[account]);
    }
    _isExcluded[account] = true;
    _excluded.push(account);
}
```

```
function includeInReward(address account) external onlyOwner() {// knownsec 取消用户白名单

    require(!_isExcluded[account], "Account is already excluded");

    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

安全建议：无。

5.3. 转账功能【提示】

审计分析：合约 RODToken.sol 中，转账前先校验转账账号以及转账金额是否为零，根据条件判断是否符合 swapAndMarketing，再判断是否收取手续费，最后进行代币转账。转账时通过判断发送者和接收者是否为白名单账号，根据实际情况使用不同的转账函数并扣除转账手续费和流动性手续费后完成转账。经审计，权限控制正确，功能设计合理。在 swapAndMarketing()函数中，应该监听该函数事件，但本合约中监听的是一个不存在的函数事件 SwapAndLiquify。

```
function transfer(address recipient, uint256 amount) public override returns (bool) {// knownsec 转账

    _transfer(_msgSender(), recipient, amount);

    return true;

}

.....
```



```
function _transfer(
    address from,
    address to,
    uint256 amount
) private {
    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");
    require(amount > 0, "Transfer amount must be greater than zero");

    // is the token balance of this contract address over the min number of
    // tokens that we need to initiate a swap + liquidity lock?
    // also, don't get caught in a circular liquidity event.
    // also, don't swap & liquify if sender is uniswap pair.
    uint256 contractTokenBalance = balanceOf(address(this));
    bool overMinTokenBalance = contractTokenBalance >=
numTokensSellToAddToLiquidity; // knownsec 代币合约余额是否大于
numTokensSellToAddToLiquidity
    if (
        overMinTokenBalance &&
        !inSwapAndLiquify &&
        from != uniswapV2Pair &&
        swapAndLiquifyEnabled
    ) { // knownsec 符合条件后, swapAndMarketing
        swapAndMarketing(contractTokenBalance); // knownsec 将七成
contractTokenBalance 数量的代币销毁, 另外三成代币置换成 usdt 到 marketingWalletAddress
    }

    //indicates if fee should be deducted from transfer
    bool takeFee = true;

    //if any account belongs to _isExcludedFromFee account then remove the fee
    if(!_isExcludedFromFee[from] || !_isExcludedFromFee[to] || from ==
uniswapV2Pair){ // knownsec 判断是否为白名单账户, 若是白名单则不收取手续费
```

```

        takeFee = false;
    }

    //transfer amount, it will take tax, burn, liquidity fee
    _tokenTransfer(from,to,amount,takeFee); // knownsec 转账
}

function swapAndMarketing(uint256 contractTokenBalance) private lockTheSwap {
    //5% destroy 2% marketing
    uint256 deadFee = contractTokenBalance.mul(71).div(100); //knownsec 5%进行销毁, 销毁到黑洞地址
    uint256 marketingFee = contractTokenBalance.mul(29).div(100); //knownsec 2%进行市场营销, 用于支持营销活动及空投

    // split the contract balance into halves
    uint256 half = marketingFee;

    // swap tokens for ETH
    swapTokensForUsdt(half); // <- this breaks the ETH -> HATE swap when swap+liquify is triggered //knownsec 传入市场钱包

    // how much ETH did we just swap into?
    IERC20(address(this)).transfer(deadAddress, deadFee); //knownsec 销毁

    emit SwapAndLiquify(contractTokenBalance, deadFee, marketingFee); //knownsec 此处应该监听 swapAndMarketing 而不是 SwapAndLiquify
}

.....

function _tokenTransfer(address sender, address recipient, uint256 amount,bool takeFee) private
{ // knownsec 实际转账
    if(!takeFee)
        removeAllFee();

    // knownsec 根据是否为白名单账户, 选取转账通道
    if(!_isExcluded[sender] && !_isExcluded[recipient]) {
        _transferFromExcluded(sender, recipient, amount);
    }
}

```

```

    } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
        _transferToExcluded(sender, recipient, amount);
    } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
        _transferStandard(sender, recipient, amount);
    } else if (_isExcluded[sender] && _isExcluded[recipient]) {
        _transferBothExcluded(sender, recipient, amount);
    } else {
        _transferStandard(sender, recipient, amount);
    }

    if(!takeFee)
        restoreAllFee();
}

```

安全建议：无。

6. 代码基本漏洞检测

6.1. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本。

检测结果：经检测，智能合约代码中制定了编译器版本 0.6.12，不存在该安全问题。

安全建议：无。

6.2. 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。

6.3. 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。

6.4. 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。

6.5. 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。

6.6. 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。

6.7. 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。

6.8. 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。

6.9. 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。

6.10. call 注入攻击【通过】

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.11. 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。

6.12. 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。

6.13. 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。

6.14. 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。

6.15. 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、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。

6.16. 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。

6.17. 不安全的外部调用【通过】

检查合约代码实现中是否使用了不安全的外部接口，接口可控可能导致执行环境被切换，控制合约执行任意代码。

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。

6.18. 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。

6.19. 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 storage 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。

6.20. 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代币，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，可能由于代币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。

6.21. 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。

6.22. 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。

6.23. 拒绝服务攻击【通过】

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。

6.24. 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。

6.25. 重入攻击检测【通过】

Solidity 中的 call.value()函数在被用来发送代币的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送代币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。

6.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

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

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

6.27. 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。

7. 附录 A：合约资金管理安全评估

合约资金管理		
合约内资产类型	涉及函数	安全风险
用户转账代币资产	transfer、swapAndMarketing	安全

检查合约业务逻辑中用户转入数字货币资产的管理安全性。观察是否存在转入合约的数字货币资产被错误记录、错误转出、后门提现等易引起客户资金损失的安全风险。



官方网站

www.knownseclab.com

邮箱

blockchain@knownsec.com

微信公众号

