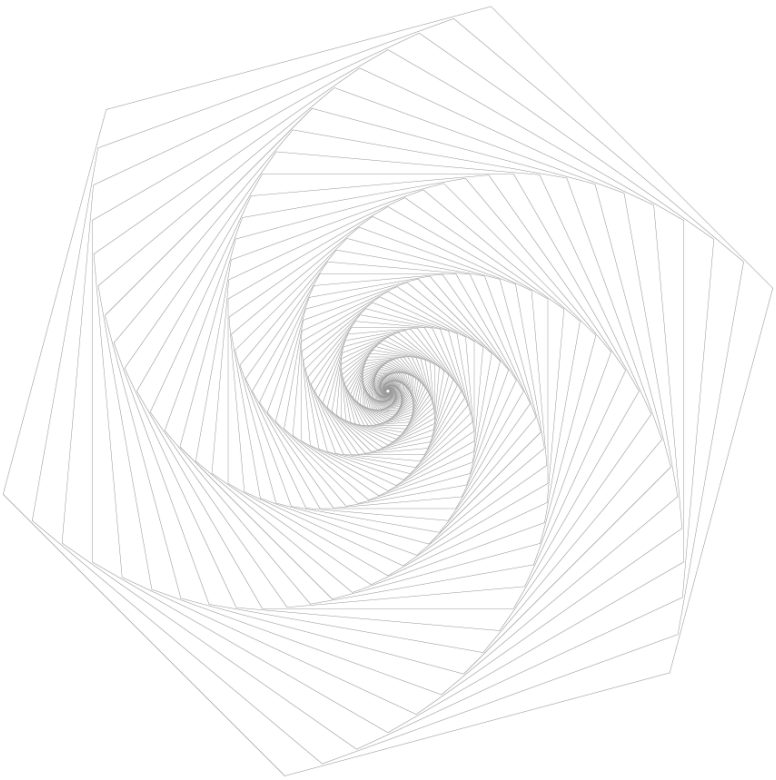
知道创宇

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	20220125	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
Blade 智能合约审计报告	V1.0	188a1d55d39f41da91e1478582b744f3	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

本次报告有效测试时间是从 2022 年 1 月 20 日开始到 2022 年 1 月 25 日结束，在此期间针对 **Blade 智能合约的代币代码**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	188a1d55d39f41da91e1478582b744f3
报告查询链接	https://attest.im/attestation/searchResult?qurey=188a1d55d39f41da91e1478582b744f3

2. 项目信息

2.1. 项目描述

Blade Warrior 是一款身临其境、奇幻、靠游戏赚钱的 NFT 区块链游戏。

2.2. 项目官网

https://www.blade.game/

2.3. 白皮书

https://docs.blade.game/

2.4. 审核版本代码

farm.zip

2.5. 合约文件及哈希/合约部署地址

合约文件	MD5
Ownable.sol	A9A5F037FF6EB170C61159489A2E59A5
SafeMath.sol	9D9111B542083B32ED6A981CD25587B9
IERC20.sol	A2B7C205E38A51FD414D256C8C294FEE
SafeERC20.sol	45646FB3F276A48B747A11C86E45EEDC
Context.sol	B184DAA83F6127830D8E8D657E6152D5
EnumerableSet.sol	CD0E1995CA83B2F2B80884427338499E
ReentrancyGuard.sol	10FA3498FD03560DDDB3D70A7CA7DDA4
IPriceOracle.sol	5549F4B8301CD9E579C9B8DB130A4665

MasterGardener.sol	B584C594A81BC54B4D30868F0F127024
PriceOracle.sol	047A112DAFEEF41C3FEB294D59D7369E
WithAdminRole.sol	1CAD70C6D88698874E9ED0547F0A0C5F
WithPriceOracle.sol	662FA6C993DBB18A6550AF08F2625E87

3. 外部可见性分析

3.1. MasterGardener 合约

MasterGardener					
函数名	**可见性**	**状态修改**	**修饰器**	**可支付接收**	**说明**
initialize	public	True	initializer	---	---
currentDayth	public	False	---	---	---
dayth	public	False	---	---	---
currentWeekth	public	False	---	---	---
currentLockPercentage	public	False	---	---	---
weekth	public	False	---	---	---
poolLength	external	False	---	---	---
add	public	True	restrictednonDuplicated(_lpToken)	---	---
fixTotalAllocPoint	public	True	restricted	---	---
set	public	True	restricted	---	---
addRewardToken	public	False	---	---	---
getRewardTokenInfo	public	True	---	---	---
updatePool	public	True	---	---	---
getMultiplier	public	False	---	---	---
pendingReward	public	True	---	---	---

claimReward	public	True	onlyNonContract nonReentrant	---	---
_harvest	internal	True	---	---	---
_sendReward	internal	True	---	---	---
unLockReward	public	True	---	---	---
getLockReward	public	False	---	---	---
getPoolInfo	public	False	---	---	---
lpLimit	public	False	---	---	---
getAprValues	public	True	restricted	---	---
getLockday	public	True	restricted	---	---
getLocdWithdraw	public	False	---	---	---
getBlockNum	public	False	---	---	---
pendingCherry	public	False	---	---	---
claimCherryReward	public	True	restricted	---	---
init	public	True	notInitialized	---	---
deposit	public	False	---	---	---
withdrawFor	public	True	restricted	---	---
setMutiply	public	False	---	---	---

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	**MasterGardener 回撤功能**	通过	经检测，不存在安全问题。
	MasterGardener 质押生息功能	通过	经检测，不存在安全问题。
	MasterGardener 合约权限控制功能	通过	经检测，不存在安全问题。
	MasterGardener 合约初始化功能	提示	经检测，不存在安全问题，仅进行提示。
	PriceOracle 合约价格预言功能	提示	经检测，不存在安全问题，仅进行提示。
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。

5. 业务安全性检测

5.1. MasterGardener 回撤功能【通过】

审计分析：对 MasterGardener 合约中的回撤功能进行安全审计；withdraw 实现在指定 pid 池子中通过 LP 回撤资金与奖励功能同时存在着防重入和 safemath 等安全措施。经审计，逻辑设计合理，未发现安全问题。

```
    function withdraw(uint256 _pid, uint256 _amount)
    public
    nonReentrant
    onlyNonContract
{// knownsec  回撤指定的池子指定的金额  非合约账户可执行
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];

    require(user.amount >= _amount, "over deposited");
    uint256 day = _getDays();
    if (isLock) {// knownsec  锁定检查
        require(day > lockDays, "lp_is_locked");
    }

    updatePool(_pid);// knownsec  更新池子
    _harvest(_pid);// knownsec  收取奖励

    if (_amount > 0) {
        LpInterface(address(liquidityaddr)).withdraw(pool.pid, _amount);// knownsec
liquidityaddr 回撤
        uint256 _userWithdrawAmount = _amount;
        if (isWithdraw) {
            if (day <= lockWithdrawDays) {
                _userWithdrawAmount = _amount.mul(withdrawPercent).div(100);
```

```
            }
        }
        IERC20(pool.lpToken).safeTransfer(
            address(msg.sender),// knownsec  转账
            _userWithdrawAmount
        );
        lpSupply = lpSupply.sub(_amount); //
        user.amount = user.amount.sub(_amount);
        user.rewardDebt = user.amount.mul(pool.accGovTokenPerShare);
        emit Withdraw(msg.sender, _pid, _amount);
    }
}
```

安全建议：无。

5.2. MasterGardener 质押生息功能【通过】

审计分析：对 MasterGardener 合约中的 deposit 质押生息功能进行安全审计。该函数功能是让用户质押自己 lpToken 并产生奖励。经审计，该功能逻辑设计合理，未发现安全问题。

```
    function deposit(uint256 _pid, uint256 _amount)
    public
    nonReentrant
    onlyNonContract
{// knownsec  质押指定池子 lp 并生息 EGG
    require(
        _amount > 0,
        "CND 0"
    );

    PoolInfo storage pool = poolInfo[_pid];
```

```
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount + _amount <= lpLimit(), "out of 200 limit");// knownsec lp 限制 200
ether

        // When a user deposits, we need to update the pool and harvest beforehand,
        // since the rates will change.
        updatePool(_pid);// knownsec 更新池子
        _harvest(_pid);// knownsec 收取奖励

        IERC20(pool.lpToken).safeTransferFrom(// knownsec 转账
          address(msg.sender),
          address(this),
          _amount
        );

        bool isSuccess = pool.lpToken.approve(liquidityaddr, _amount);// knownsec 批准流动
性池子
        require(isSuccess, "approve failed");// knownsec require 检查
        LpInterface(address(liquidityaddr)).deposit(pool.pid, _amount);// knownsec 质押
liquidityaddr

        if (isLock) {
          IERC20(certificateaddr).safeTransfer(msg.sender, certificateNums);
        }
        lpSupply = lpSupply.add(_amount); //
        _depositeBlockNumber[msg.sender] = block.number;
        if (user.amount == 0) {
          user.rewardDebtAtBlock = block.number;
        }
        user.amount = user.amount.add(_amount);
        user.rewardDebt = user.amount.mul(pool.accGovTokenPerShare);// 奖励更新
        emit Deposit(msg.sender, _pid, _amount);
      }
```

安全建议：无。

5.3. MasterGardener 合约权限控制功能【通过】

审计分析：对 MasterGardener 合约中的函数调用权限进行审计。各个重要函数如状态变量的修改均使用了 restricted 修饰器进行控制。经审计，该修饰器实现为检查 msg.sender 是否是 GAME_ADMIN，不存在安全问题。

```
    function addRewardToken(string memory name, address rewardToken)
    external
    // uint256 price
    restricted// knownsec  增加奖励 token  修饰器 restricted
{
    require(rewardToken != address(0), "token is null");

    uint256 len = rewardTokenInfos.length;// knownsec  列表长度
    for (uint256 i = 0; i < len; i++) {// knownsec  高 gas 消耗，可采用其他检查方式
        require(
            rewardToken != rewardTokenInfos[i].rewardToken,// knownsec
            "already add"
        );
    }

    RewardTokenInfo storage rt = rewardTokenInfos.push();// knownsec  更新
    rt.name = name;
    rt.rewardToken = rewardToken;
    // rt.price = price;
    emit NewRewardToken(len, rewardToken);
}
...
    modifier restricted() {
        require(hasRole(GAME_ADMIN, msg.sender), "NGA");
```

```
        _;
    }
```

安全建议：无。

5.4. MasterGardener 合约初始化功能【提示】

审计分析：对 MasterGardener 合约中的初始化功能进行安全审计。经审计，函数不存在权限控制，因此在部署时可能被人抢占初始化。但该问题不属于安全问题，因此仅进行提示。

```
    function init(
        address _liquidityaddr,
        uint256 _rewardPerBlock,
        uint256 _startBlock,
        uint256[] memory _rewardMultiplier
    ) public notInitialized {// knownsec  初始化  只能初始化一次  未控制权限
        // super.initOwner();
        _INITIALIZED_ = true;
        liquidityaddr = _liquidityaddr;
        REWARD_PER_BLOCK = _rewardPerBlock;
        START_BLOCK = _startBlock;
        REWARD_MULTIPLIER = _rewardMultiplier;
        for (uint256 i = 0; i < REWARD_MULTIPLIER.length - 1; i++) {
            uint256 halvingAtBlock = _halvingAfterBlock
                .mul(i + 1)
                .add(_startBlock)
                .add(1);
            HALVING_AT_BLOCK.push(halvingAtBlock);
        }
        FINISH_BONUS_AT_BLOCK = _halvingAfterBlock
            .mul(REWARD_MULTIPLIER.length - 1)
```

```
            .add(_startBlock);
        HALVING_AT_BLOCK.push(uint256(-1));
        //    init PERCENT_LOCK_BONUS_REWARD
        for (uint256 i = 0; i < REWARD_MULTIPLIER.length - 1; i++) {
            uint256 unlockPercentage = initPercent.add(incrPercent.mul(i));
            uint256 lockPercentage = unlockPercentage >= 100
                ? 0
                : 100 - unlockPercentage;
            PERCENT_LOCK_BONUS_REWARD.push(lockPercentage);
        }
        PERCENT_LOCK_BONUS_REWARD.push(0);
    }
```

安全建议：无。

5.5. PriceOracle 合约价格预言功能【提示】

审计分析：对 PriceOracle 合约中的价格预言功能进行安全审计。该预言机通过了一种不足够安全的获取池子内的交易对余额进行价格判断，属于不安全的价格预言，与项目方沟通后仅用于 apr 的价格表述获取。经审计，代码内仅 getAprValues 进行了调用，目前不存在安全问题，因此此仅进行提示，后续开发应注意该预言机的使用场景。

```
        function usdtReserved() public view override returns (uint256 usdtAmount) {
            usdtAmount = _usdtContract.balanceOf(_cheLpContract);// knownsec cheLp 的 USDT
        }

        function bladeReserved() public view override returns (uint256 blade) {
            blade = _bladeContract.balanceOf(_cheLpContract);// knownsec chelp 的 blade
        }
```

```
//   xxxx usdt gwei/blade
function usdtBladePrice() public view override returns (uint256 price) {
        price = (usdtReserved() * 1000000000) / bladeReserved();
}

// xxxx    blade gwei/usdt
function bladeUsdtPrice() public view override returns (uint256 price) {
        price = (bladeReserved() * 1000000000) / usdtReserved();
}
```

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 0.6.0，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 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。

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**
用户抵押代币资产	deposit	安全
用户抵押平台币资产	---	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。

知道创宇

区块链安全实验室

官方网站

www.knownseclab.com

邮箱

blockchain@knownsec.com

微信公众号