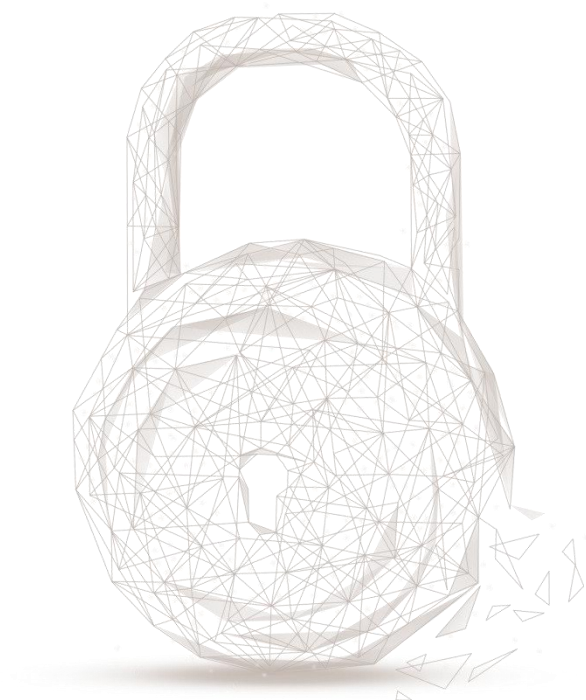




# 智能合约安全审计报告



审计编号：202009211852

报告查询名称：MRK

审计合约信息：

合约名称	审计合约地址	审计合约链接地址
MoonRabbit.finance (MRK)	TRqJw3csFiyswCY7tYjVMpfk9jxYaWmPME	<a href="https://tronscan.io/#/contract/TRqJw3csFiyswCY7tYjVMpfk9jxYaWmPME/code">https://tronscan.io/#/contract/TRqJw3csFiyswCY7tYjVMpfk9jxYaWmPME/code</a>
MoonRabbitPoolMRKTRXReward	TYUqwFyacZCcXUaSN3XpTugbYmZ6GR7La	<a href="https://tronscan.io/#/contract/TYUqwFyacZCcXUaSN3XpTugbYmZ6GR7La/code">https://tronscan.io/#/contract/TYUqwFyacZCcXUaSN3XpTugbYmZ6GR7La/code</a>
MoonRabbitPoolUSDTReward	TG9NDV2V2YXaXwXNxCcFbNg5kKG9FLHJC9	<a href="https://tronscan.io/#/contract/TG9NDV2V2YXaXwXNxCcFbNg5kKG9FLHJC9/code">https://tronscan.io/#/contract/TG9NDV2V2YXaXwXNxCcFbNg5kKG9FLHJC9/code</a>
MoonRabbitPoolTAIReward	TGdkeiGUfuwMK5yKSAu3nUjfo4A6rMDQJH	<a href="https://tronscan.io/#/contract/TGdkeiGUfuwMK5yKSAu3nUjfo4A6rMDQJH/code">https://tronscan.io/#/contract/TGdkeiGUfuwMK5yKSAu3nUjfo4A6rMDQJH/code</a>
MoonRabbitPoolMFIReward	TBNE4n6cVw4hfoWQdX4Y5CRfeb3RVPQF5	<a href="https://tronscan.io/#/contract/TBNE4n6cVw4hfoWQdX4Y5CRfeb3RVPQF5/code">https://tronscan.io/#/contract/TBNE4n6cVw4hfoWQdX4Y5CRfeb3RVPQF5/code</a>

合约审计开始日期：2020. 09. 21

合约审计完成日期：2020. 09. 21

审计结果：通过

审计团队：成都链安科技有限公司

## 审计类型及结果：

序号	审计类型	审计子项	审计结果
1	代码规范审计	编译器版本安全审计	通过
		弃用项审计	通过
		冗余代码审计	通过
		require/assert 使用审计	通过
		gas 消耗审计	通过
2	通用漏洞审计	整型溢出审计	通过
		重入攻击审计	通过
		伪随机数生成审计	通过
		交易顺序依赖审计	通过

		拒绝服务攻击审计	通过
		函数调用权限审计	通过
		call/delegatecall 安全审计	通过
		返回值安全审计	通过
		tx.origin 使用安全审计	通过
		重放攻击审计	通过
		变量覆盖审计	通过
3	业务审计	业务逻辑审计	通过
		业务实现审计	通过

备注：审计意见及建议请见代码注释。

免责声明：本次审计仅针对本报告载明的审计类型及结果表中给定的审计类型范围进行审计，其他未知安全漏洞不在本次审计责任范围之内。成都链安科技仅根据本报告出具前已经存在或发生的攻击或漏洞出具本报告，对于出具以后存在或发生的新的攻击或漏洞，成都链安科技无法判断其对智能合约安全状况可能的影响，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，仅基于合约提供者在本报告出具前已向成都链安科技提供的文件和资料，且该部分文件和资料不存在任何缺失、被篡改、删减或隐瞒的前提下作出的；如提供的文件和资料存在信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符等情况或提供文件和资料在本报告出具后发生任何变动的，成都链安科技对由此而导致的损失和不利影响不承担任何责任。成都链安科技出具的本审计报告系根据合约提供者提供的文件和资料依靠成都链安科技现掌握的技术而作出的，由于任何机构均存在技术的局限性，成都链安科技作出的本审计报告仍存在无法完整检测出全部风险的可能性，成都链安科技对由此产生的损失不承担任何责任。

本声明最终解释权归成都链安科技所有。

## 审计结果说明：

本公司采用形式化验证、静态分析、动态分析、典型案例测试和人工审核的方式对MRK项目智能合约代码规范性、安全性以及业务逻辑三个方面进行多维度全面的安全审计。**经审计，MRK项目智能合约通过所有检测项，合约审计结果为通过。**以下为本合约详细审计信息。

## 代码规范审计

### 1. 编译器版本安全审计

老版本的编译器可能会导致各种已知安全问题，建议开发者在代码中指定合约代码采用最新的编译器版本，并消除编译器告警。

- 安全建议：无
- 审计结果：通过

## 2. 弃用项审计

Solidity智能合约开发语言处于快速迭代中，部分关键字已被新版本的编译器弃用，如throw、years等，为了消除其可能导致的隐患，合约开发者不应该使用当前编译器版本已弃用的关键字。

- 安全建议：无
- 审计结果：通过

## 3. 冗余代码审计

智能合约中的冗余代码会降低代码可读性，并可能需要消耗更多的gas用于合约部署，建议消除冗余代码。

- 安全建议：无
- 审计结果：通过

## 4. require/assert 使用审计

Solidity使用状态恢复异常来处理错误。这种机制将会撤消对当前调用(及其所有子调用)中的状态所做的所有更改，并向调用者标记错误。函数assert和require可用于检查条件并在条件不满足时抛出异常。assert函数只能用于测试内部错误，并检查非变量。require函数用于确认条件有效性，例如输入变量，或合约状态变量是否满足条件，或验证外部合约调用的返回值。

- 安全建议：无
- 审计结果：通过

## 5. gas 消耗审计

波场虚拟机执行合约代码需要消耗gas，当gas不足时，代码执行会抛出out of gas异常，并撤销所有状态变更。合约开发者需要控制代码的gas消耗，避免因为gas不足导致函数执行一直失败。

- 安全建议：无
- 审计结果：通过

# 通用漏洞审计

## 1. 整型溢出审计

整型溢出是很多语言都存在的安全问题，它们在智能合约中尤其危险。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uint类型时，0减去1会下溢得到最大数字值。溢出情况会导致不正确的结果，特别是如果其可能的结果未被预期，可能会影响程序的可靠性和安全性。

- 安全建议：无
- 审计结果：通过

## 2. 重入攻击审计

重入漏洞是最典型的智能合约漏洞，该漏洞原因是Solidity中的`call.value()`函数在被用来发送TRX的时候会消耗它接收到的所有gas，当调用`call.value()`函数发送TRX的逻辑顺序存在错误时，就会存在重入攻击的风险。

- 安全建议：无
- 审计结果：通过

## 3. 伪随机数生成审计

智能合约中可能会使用到随机数，在solidity下常见的是用block区块信息作为随机因子生成，但是这样使用是不安全的，区块信息是可以被矿工控制或被攻击者在交易时获取到，这类随机数在一定程度上是可预测或可碰撞的，比较典型的例子就是fomo3d的airdrop随机数可以被碰撞。

- 安全建议：无
- 审计结果：通过

## 4. 交易顺序依赖审计

在波场的交易打包执行过程中，面对相同难度的交易时，矿工往往会选择gas费用高的优先打包，因此用户可以指定更高的gas费用，使自己的交易优先被打包执行。

- 安全建议：无
- 审计结果：通过

## 5. 拒绝服务攻击审计

拒绝服务攻击，即Denial of Service，可以使目标无法提供正常的服务。在波场智能合约中也会存在此类问题，由于智能合约的不可更改性，该类攻击可能使得合约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有很多种，包括在作为交易接收方时的恶意revert、代码设计缺陷导致gas耗尽等等。

- 安全建议：无
- 审计结果：通过

## 6. 函数调用权限审计

智能合约如果存在高权限功能，如：铸币、自毁、change owner等，需要对函数调用做权限限制，避免权限泄露导致的安全问题。

- 安全建议：无
- 审计结果：通过

## 7. call/delegatecall安全审计



Solidity中提供了call/delegatecall函数来进行函数调用，如果使用不当，会造成call注入漏洞，例如call的参数如果可控，则可以控制本合约进行越权操作或调用其他合约的危险函数。

- 安全建议：无
- 审计结果：通过

## 8. 返回值安全审计

在Solidity中存在transfer()、send()、call.value()等方法中，transfer转账失败交易会回滚，而send和call.value转账失败会return false，如果未对返回做正确判断，则可能会执行到未预期的逻辑；另外在TRC20 Token的transfer/transferFrom功能实现中，也要避免转账失败return false的情况，以免造成假充值漏洞。

- 安全建议：无
- 审计结果：通过

## 9. tx.origin使用安全审计

在波场智能合约的复杂调用中，tx.origin表示交易的初始创建者地址，如果使用tx.origin进行权限判断，可能会出现错误；另外，如果合约需要判断调用方是否为合约地址时则需要使用tx.origin，不能使用extcodesize。

- 安全建议：无
- 审计结果：通过

## 10. 重放攻击审计

重放攻击是指如果两份合约使用了相同的代码实现，并且身份鉴权在传参中，当用户在向一份合约中执行一笔交易，交易信息可以被复制并且向另一份合约重放执行该笔交易。

- 安全建议：无
- 审计结果：通过

## 11. 变量覆盖审计

波场存在着复杂的变量类型，例如结构体、动态数组等，如果使用不当，对其赋值后，可能导致覆盖已有状态变量的值，造成合约执行逻辑异常。

- 安全建议：无
- 审计结果：通过

## 代币信息审计

### 1. MRK代币基本信息

代币名称	MoonRabbit.finance
代币简称	MRK
代币精度	18
代币总量	当前数量约10542枚（可铸币，铸币上限为12000枚）
代币类型	TRC20

表格 1 代币基本信息

## 业务审计

在本项目中依据同一代码架构实现了4个流动性挖矿奖池智能合约，分别为MoonRabbitPoolMRKTRXReward、MoonRabbitPoolUSDTReward、MoonRabbitPoolTAIReward和MoonRabbitPoolMFIReward，这四个智能合约实现代码在抵押代币地址、挖矿代币地址、阶段完成时间初始化等部分存在细微差异，不影响各个奖池合约运行，以下截图以MoonRabbitPoolUSDTReward为准。

### 1. 抵押初始化

- **业务描述：**合约的“抵押-奖励”模式需要初始化相关参数（奖励比例rewardRate、首次更新时间lastUpdateTime、阶段完成时间periodFinish），通过指定的奖励分配管理员地址rewardDistribution调用notifyRewardAmount函数，输入初始用于计算奖励比例的奖励数值reward，初始化抵押与奖励相关参数。该函数可由指定地址rewardDistribution在任意时刻调用，来控制奖励比例与关键时间判断节点，即使奖励比例rewardRate在checkhalve修饰器执行逻辑时被减半更新后仍可通过在该函数输入指定数值reward来修改，如数值过小会使得用户收益与预期不符。

```

729 function notifyRewardAmount(uint256 reward)
730 external
731 onlyRewardDistribution
732 updateReward(address(0))
733 {
734     if (block.timestamp >= periodFinish) {
735         rewardRate = reward.div(DURATION);
736     } else {
737         uint256 remaining = periodFinish.sub(block.timestamp);
738         uint256 leftover = remaining.mul(rewardRate);
739         rewardRate = reward.add(leftover).div(DURATION);
740     }
741
742     mfi.mint(address(this), reward);
743
744     lastUpdateTime = block.timestamp;
745     periodFinish = block.timestamp.add(DURATION);
746     emit RewardAdded(reward);
747 }
748 }
749

```

图 1 notifyRewardAmount 函数源码截图

- **相关函数:** notifyRewardAmount、rewardPerToken、lastTimeRewardApplicable
- **安全建议:** 无
- **审计结果:** 通过

## 2. 抵押代币

- **业务描述:** 合约实现了stake函数用于抵押USDT代币，用户预先授权该合约地址，通过调用USDT合约中的transferFrom函数，合约地址代理用户将指定数量的USDT代币转至本合约地址；该函数限制用户仅可在“抵押-奖励”模式开启（到达指定时间）后进行调用；每次调用该函数抵押代币时通过修饰器updateReward更新奖励相关数据；以及每次调用通过修饰器checkhalve检查是否到达阶段完成时间，并进行奖励减半操作和奖励比例与阶段完成时间的更新。

```

612 function stake(uint256 amount) public {
613     _totalSupply = _totalSupply.add(amount);
614     _balances[msg.sender] = _balances[msg.sender].add(amount);
615     y.safeTransferFrom(msg.sender, address(this), amount);
616 }
617

```

图 2 stake 函数源码截图 (LPTokenUSDWrapper 合约)

```

685 function stake(uint256 amount) public updateReward(msg.sender) checkhalve checkStart{
686     require(amount > 0, "Cannot stake 0");
687     super.stake(amount);
688     emit Staked(msg.sender, amount);
689 }

```

图 3 stake 函数源码截图 (MoonRabbitPoolUSDTReward 合约)

- **相关函数:** stake、rewardPerToken、lastTimeRewardApplicable、earned、balanceOf



- 安全建议：无
- 审计结果：通过

### 3. 提取抵押代币

- **业务描述：**合约实现了withdraw函数用于提取已抵押的y代币，通过调用USDT合约中的transfer函数，合约地址将指定数量的USDT代币转至函数调用者（用户）地址；该函数限制用户仅可在“抵押-奖励”模式开启（到达指定时间）后进行调用；每次调用该函数抵押代币时通过修饰器updateReward更新奖励相关数据；以及每次调用通过修饰器checkhalve检查是否到达阶段完成时间，并进行奖励减半操作和奖励比例与阶段完成时间的更新。

```
618     function withdraw(uint256 amount) public {
619         _totalSupply = _totalSupply.sub(amount);
620         _balances[msg.sender] = _balances[msg.sender].sub(amount);
621         y.transfer(msg.sender, amount);
622     }
```

图 4 withdraw 函数源码截图（LPTokenUSDTWrapper 合约）

```
691     function withdraw(uint256 amount) public updateReward(msg.sender) checkhalve checkStart{
692         require(amount > 0, "Cannot withdraw 0");
693         super.withdraw(amount);
694         emit Withdrawn(msg.sender, amount);
695     }
```

图 5 withdraw 函数源码截图（MoonRabbitPoolUSDTReward 合约）

- **相关函数：**withdraw、rewardPerToken、lastTimeRewardApplicable、earned、balanceOf
- 安全建议：无
- 审计结果：通过

### 4. 领取抵押奖励

- **业务描述：**合约实现了getReward函数用于领取抵押奖励（MRK代币），通过调用MRK合约中的transfer函数，合约地址将指定数量（用户的全部抵押奖励）的MRK代币转至函数调用者（用户）地址，同时该函数将铸造用户全部抵押奖励的10%（MRK代币）到governance地址；该函数限制用户仅可在“抵押-奖励”模式开启（到达指定时间）后进行调用；每次调用该函数抵押代币时通过修饰器updateReward更新奖励相关数据；以及每次调用通过修饰器checkhalve检查是否到达阶段完成时间，并进行奖励减半操作和奖励比例与阶段完成时间的更新。

```
702     function getReward() public updateReward(msg.sender) checkhalve checkStart{
703         uint256 reward = earned(msg.sender);
704         if (reward > 0) {
705             rewards[msg.sender] = 0;
706             mfi.safeTransfer(msg.sender, reward);
707             mfi.mint(governance, reward.div(10));
708             emit RewardPaid(msg.sender, reward);
709         }
710     }
```

图 6 getReward 函数源码截图

- **相关函数：**getReward、rewardPerToken、lastTimeRewardApplicable、earned、balanceOf
- **安全建议：**无
- **审计结果：**通过

## 5. 退出抵押奖励参与

- **业务描述：**合约实现了exit函数用于调用者退出抵押奖励参与，调用withdraw函数提取全部已抵押的USDT代币，调用getReward函数领取完调用者的抵押奖励，结束“抵押-奖励”模式参与。此时用户地址由于其已抵押USDT代币数量为空，无法获得新的抵押奖励。

```
697     function exit() external {
698         withdraw(balanceOf(msg.sender));
699         getReward();
700     }
```

图 7 exit 函数源码截图

- **相关函数：**exit、withdraw、getReward、rewardPerToken、lastTimeRewardApplicable、earned、balanceOf
- **安全建议：**无
- **审计结果：**通过

## 6. 奖励相关数据查询功能

- **业务描述：**合约用户可通过调用lastTimeRewardApplicable函数查询当前时间戳与阶段完成时间中最早的时间戳；调用rewardPerToken函数可查询每个抵押代币可获得的抵押奖励；调用earned函数可查询指定地址所获取的总抵押奖励。
- **相关函数：**lastTimeRewardApplicable、rewardPerToken、earned
- **安全建议：**无
- **审计结果：**通过



成都链安  
B E O S I N

官方网址

<https://lianantech.com>

电子邮箱

[vaas@lianantech.com](mailto:vaas@lianantech.com)

微信公众号

