智能合约安全审计报告

1. 概要

慢雾安全团队于 2021 年 2 月 3 日，收到 Starlink 团队对 StarlinkMinerV2 系统安全审计的申请，根据项目特点慢雾安全团队制定如下审计方案。

慢雾安全团队将采用“白盒为主，黑灰为辅”的策略，以最贴近真实攻击的方式，对项目进行安全审计。

慢雾科技 DeFi 项目测试方法：

黑盒测试	站在外部从攻击者角度进行安全测试。
灰盒测试	通过脚本工具对代码模块进行安全测试，观察内部运行状态，挖掘弱点。
白盒测试	基于项目的源代码，进行脆弱性分析和漏洞挖掘。

慢雾科技 DeFi 漏洞风险等级：

严重漏洞	严重漏洞会对项目的安全造成重大影响，强烈建议修复严重漏洞。
高危漏洞	高危漏洞会影响项目的正常运行，强烈建议修复高危漏洞。
中危漏洞	中危漏洞会影响项目的运行，建议修复中危漏洞。
低危漏洞	低危漏洞可能在特定场景中会影响项目的业务操作，建议项目方自行评估和考虑这些问题是否需要修复。
弱点	理论上存在安全隐患，但工程上极难复现。
增强建议	编码或架构存在更好的实践方法。

2. 审计方法

慢雾安全团队智能合约安全审计流程包含两个步骤:

- 使用开源或内部自动化分析的工具对合约代码中常见的安全漏洞进行扫描和测试。
- 人工审计代码的安全问题，通过人工分析合约代码，发现代码中潜在的安全问题。

如下是合约代码审计过程中我们会重点审查的漏洞列表:

（其他未知安全漏洞不包含在本次审计责任范围）

- 重入攻击
- 重放攻击
- 重排攻击
- 短地址攻击
- 拒绝服务攻击
- 交易顺序依赖
- 条件竞争攻击
- 权限控制攻击
- 整数上溢/下溢攻击
- 时间戳依赖攻击
- Gas 使用，Gas 限制和循环
- 冗余的回调函数
- 不安全的接口使用
- 函数状态变量的显式可见性
- 逻辑缺陷
- 未声明的存储指针
- 算术精度误差
- tx.origin 身份验证
- 假充值漏洞
- 变量覆盖

3. 项目背景

3.1 项目介绍

星链 StarLinkNetwork 的核心目标是搭建中心化及去中心化世界的全局搜索。目前世界主流搜索引擎仅局限于中心化网站内容，随着去中心化世界的爆发式发展，用户急需一个更加全面的搜索引擎去满足日益增长的多元化需求。StarLinkNetwork 将会让中心化世界及去中心化世界的内容出现在搜索结果中，让更多用户体验无边界的搜索引擎，让更多用户更加便捷的获取去中心化世界中的交易数据、链上数据、链上资源等信息，让更多用户感受到 StarLinkNetwork 带来的更加客观的搜索结果。

审计合约文件：

项目源代码

审计初始版本：

https://github.com/starlink-so/starlinkminerv2/blob/main/contracts/SLNToken.sol

https://github.com/starlink-so/starlinkminerv2/blob/main/contracts/StarPools.sol

commit: 86277148ab0e4c9c5c4faa7f4ec36a0a1203c3fe

审计最终版本：

https://github.com/starlink-so/starlinkminerv2/blob/main/contracts/SLNToken.sol

https://github.com/starlink-so/starlinkminerv2/blob/main/contracts/StarPools.sol

commit: 74dec8ec37fad2deb08cc0199b7e67b3fd264870

本次审计范围为：SLNToken.sol 与 StarPools.sol 部分

3.2 审计合约结构

```
├── SLNToken.sol
└── StarPools.sol
```

4. 代码概述

4.1 主要合约地址

合约暂未在主网进行部署。

4.2 主要合约函数可见性分析

在审计过程中，慢雾安全团队对核心合约的可见性进行分析，结果如下:

SLNToken			
Function Name	Decorated	Mutability	Modifiers
setpool	External	Can modify state	onlyOwner
setv1swap	External	Can modify state	onlyOwner
mint	Public	Can modify state	-
receive	External	Payable	-

StarPoolsV2			
Function Name	Decorated	Mutability	Modifiers
poolLength	External	-	-
add	Public	Can modify state	onlyOwner
setRewardPerBlock	External	Can modify state	onlyOwner
setRewardDistributionFactor	External	Can modify state	onlyOwner
setAllocPoint	External	Can modify state	onlyOwner
setPoolType	External	Can modify state	onlyOwner
setReductionArgs	External	Can modify state	onlyOwner
getBlocksReward	Public	-	-

getBlockReward	Public	-	-
pendingRewards	Public	-	-
totalRewards	Internal	-	-
massUpdatePools	Public	Can modify state	-
updatePool	Public	Can modify state	-
deposit	External	Can modify state	-
withdraw	External	Can modify state	-
claimAll	External	Can modify state	-
emergencyWithdraw	External	Can modify state	-
safeSlnTransfer	Internal	Can modify state	-
dev	External	Can modify state	-
ope	External	Can modify state	-
receive	External	Payable	-

4.3 代码审计详情

4.3.1 中危漏洞

4.3.1.1 权限过大风险

1）在 SLNToken 中存在 owner 角色，owner 可以通过 setpool 函数更改 starpools 的地址，这个地址控制着铸币函数，这将导致 owner 权限过大的风险。

修复建议：建议将 owner 权限移交社区治理。

代码位置：SLNToken.sol

```
function setpool(address _pool) external onlyOwner {
    starpools = _pool;
}

function mint(address _to, uint256 _amount) public {
    require(msg.sender == starpools || msg.sender == slnv1swap, 'from pools or swap call');
    require(totalSupply().add(_amount) <= capmax, 'cap exceeded');
    _mint(_to, _amount);
```

```
    }
```

修复状态：已在 commit: 68000d901b469c8f339efe76254ac72eb9df38c1 中修复。

2) 在 StarPoolsV2 合约中存在 owner 角色，owner 可以对挖矿合约的池子权重、挖矿周期等敏感参数进行修改，这将导致 owner 权限过大的风险，此风险将会影响用户未来收益，但对已获得收益及本金无影响。

修复建议：建议将 owner 权限移交社区治理。

修复状态：经与项目方沟通反馈后，项目方表示开矿前期需根据抵押量进行灵活调整，因此项目方将在稳定挖矿后将 owner 权限移交给 timelock 合约以避免权限过大风险。

4.3.2 低危漏洞

4.3.2.1 紧急退出时收益归零问题

挖矿合约存在 emergencyWithdraw 函数，用以紧急情况下将抵押品取出的操作，但在调用的同时把用户的 rewardRemain 也置为了 0，但未将收益发放给用户，导致用户收益归零的问题。

修复建议：建议不将收益置 0。

代码位置：StarPools.sol

```
    function emergencyWithdraw(uint256 _pid) external {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        user.rewardRemain = 0;
        pool.totalAmount = pool.totalAmount.sub(amount);
        pool.lpToken.safeTransfer(address(msg.sender), amount);
        emit EmergencyWithdraw(msg.sender, _pid, amount);
    }
```

修复状态：已在 commit: 74dec8ec37fad2deb08cc0199b7e67b3fd264870 中修复。

4.3.2.2 跨周期奖励计算问题

合约通过 getBlocksReward 函数来计算在一定区块期间内的收益。如果用户抵押之后就无人再进行 updatePool 操作，那么当用户在经过多个奖励减少的周期后再进行 claimAll，将导致收益已旧的奖励周期的标准进行发放的问题。

修复建议：每个周期都更新所有池子。

代码位置：StarPools.sol

```
    function getBlocksReward(uint256 _from, uint256 _to) public view returns (uint256 value) {
        uint256 prevReductionBlock = nextReductionBlock.sub(reductionBlockPeriod);
        if ((_from >= prevReductionBlock && _to <= nextReductionBlock) ||
            (_from > bonusStableBlock))
        {
            value = getBlockReward(_to.sub(_from), rewardPerBlock, reductionCounter);
        }
        else if (_from < prevReductionBlock && _to < nextReductionBlock)
        {
            uint256 part1 = getBlockReward(_to.sub(prevReductionBlock), rewardPerBlock, reductionCounter);
            uint256 part2 = getBlockReward(prevReductionBlock.sub(_from), rewardPerBlock,
reductionCounter.sub(1));
            value = part1.add(part2);
        }
        else // if (_from > prevReductionBlock && _to > nextReductionBlock)
        {
            uint256 part1 = getBlockReward(_to.sub(nextReductionBlock), rewardPerBlock,
reductionCounter.add(1));
            uint256 part2 = getBlockReward(nextReductionBlock.sub(_from), rewardPerBlock, reductionCounter);
            value = part1.add(part2);
        }

        value = value.mul(rewardDistributionFactor).div(1e9);
    }
```

修复状态：经与项目方沟通反馈后，项目方表示奖励周期为 7 天，每个周期项目方会协助调用所有池子结算

收益。

4.3.3 增强建议

4.3.3.1 LP 池子权重改变导致收益变化问题

Owner 在调用 add 函数与 setAllocPoint 函数进行添加新池子或重设池子权重操作时，所有的 LP 池子权重都会因此发生改变。Owner 可以通过传入值为 true 的_withUpdate 参数，在调整权重前更新所 有池子以保证用户在池子权重改变之前的收益不会受到池子权重调整的影响，但如果传入值为 false 的 _withUpdate 参数，则在池子权重调整前不会先更新所有池子，这将导致用户在池子权重改变之前收益受到影响。

修复建议：建议在 LP 池子权重调整前更新所有池子，以避免用户收益受到影响。

代码位置： StarPools.sol

```
function add(uint256 _allocPoint, IERC20 _lpToken, uint256 _pooltype, bool _withUpdate) public onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    uint256 lastRewardBlock = block.number > startBlock ? block.number : startBlock;
    totalAllocPoint = totalAllocPoint.add(_allocPoint);
    poolInfo.push(PoolInfo({
        lpToken: _lpToken,
        allocPoint: _allocPoint,
        lastRewardBlock: lastRewardBlock,
        accRewardPerShare: 0,
        totalAmount: 0,
        pooltype: _pooltype
    }));
}

function setAllocPoint(uint256 _pid, uint256 _allocPoint, bool _withUpdate) external onlyOwner {
    if (_withUpdate) {
        massUpdatePools();
    }
    totalAllocPoint = totalAllocPoint.sub(poolInfo[_pid].allocPoint).add(_allocPoint);
```

```
        poolInfo[_pid].allocPoint = _allocPoint;
    }
```

修复状态：已在 commit: 6064964f87927cdfa6932fac044bea588e6772e8 中修复。

4.3.3.2 提现失败问题

在用户调用 withdraw 提现时，将调用合约的 safeTransfer 函数将抵押品转至用户账户，若目标合约(pool.lpToken) 的 transfer 函数定义了返回值但写法为按照 EIP20 标准进行 return(如 波场 USDT 合约)，这将导致 safeTransfer 函数对返回值校验失败，造成无法成功为用户进行提现。

修复建议：建议在接入抵押品时，严格检查其是否符合 EIP20 标准。

代码位置：StarPools.sol

```
    function withdraw(uint256 _pid, uint256 _amount) external {
        updatePool(_pid);
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(user.amount >= _amount, "withdraw: not good");
        user.rewardRemain = pendingRewards(_pid, msg.sender);
        user.rewardDebt = 0;
        if(_amount > 0) {
            user.amount = user.amount.sub(_amount);
            pool.totalAmount = pool.totalAmount.sub(_amount);
            pool.lpToken.safeTransfer(address(msg.sender), _amount);
        }
        user.rewardDebt = totalRewards(pool, user);
        emit Withdraw(msg.sender, _pid, _amount);
    }
```

修复状态：经与项目方沟通反馈后，项目方表示目前所有抵押品都会经过测试保证后才上线。

4.3.3.3 合约接收 HT 问题

合约中存在 pay 函数用于拒接接收 HT，但未遵循 Solidity 的接收函数写法。

修复建议：建议使用标准编码规范拒绝接收 HT。

代码位置：SLNToken.sol, StarPoolsV2

```
    function pay() public payable {
        revert();
    }
```

修复状态：已在 commit: 74dec8ec37fad2deb08cc0199b7e67b3fd264870 中修复。

5. 审计结果

5.1 总结

审计结论 ：低风险

审计编号 ：0X002102060002

审计时间 ：2021 年 02 月 06 日

审计团队 ：慢雾安全团队

审计总结 ：慢雾安全团队采用人工结合内部工具对代码进行分析。审计期间发现了 6 个问题。其中包含 1 个中危漏洞、2 个低危漏洞，并提出了 3 点增强建议。目前项目方暂未将 StarPoolsV2 合约的 owner 权限移交至 timelock 合约，StarPoolsV2 合约的 owner 仍存在权限过大的风险。此风险将会影响用户未来收益，但对已获得收益及本金无影响。项目方表示由于矿池前期质押量波动较大，需要使用 owner 权限对参数进行调节以确保前期矿池的稳定运行，并将在项目稳定运行后再将权限移交至 timelock 合约以避免此风险。

6. 声明

慢雾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，慢雾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慢雾提供的文件和资料(简称“已提供资料”)。慢雾假设：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，慢雾对由此而导致的损失和不利影响不承担任何责任。

慢雾科技
slow mist

官方网址

www.slowmist.com

电子邮箱

team@slowmist.com

微信公众号