智能合约安全审计报告

目录

1. 概要

思维世纪团队于 2021 年 4 月 3 日，收到CloudLink 团队对 CLOUDLINK 系统安全审计的申请，根据项目特点思维世纪团队制定如下审计方案。
思维世纪团队将采用“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。

思维世纪 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。

思维世纪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

思维世纪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

CLOUDLINK 是一个建立在HECO链上的DeFi数据提供，预言，交换协议。它是由Solidity语言的去中心化自治组织（DAO）和网页端去中心化应用（DAPP）组成，Cloud Link代币（CLK）是其NetWork的原生代币，用于DAO治理，抵押，借贷以及接受链上数据协议。

审计范围：

CLOUDLINK（HRC20）合约地址：

0x3366806629A55Fc2278Dcd8a11da8CCB13494976

CLK矿池合约地址：0x455208De5B89eE3F4431B0AA060771d156eafcB8

CLK - USDT LP矿池合约地址：0x070aEC929C8fC4B064595ECCD2Ee9Ea881C96EF8

CLK – WHT LP矿池合约地址：0x2f298C43B667d61acBe484571dF40Cc6C3768DE1

3.2审计合约结构

```
├──CLOUDLINKToken
└──StakingRewards
```

4.代码概述

4.1 主要合约地址

Contract Name	Contract Address
CLOUDLINK（HRC20）	0x9ede2df15deeee26a916b9b93ea7d9363d7531d7
1 号池	0x608713FF1c2DCab6a6cF78F7a3922A5ddA7bcE1F
2 号池	0x5849Deb866b4f0b2eff5077673e879F0Ce663eE5
3 号池	0xba90125BbFd0e561E61d8F41e7ae54cB9Ce29b17

4.2 主要合约函数可见性分析

在审计过程中，思维世纪团队对核心合约的可见性进行分析，结果如下:

CloudLinkToken			
Function Name	Decorated	Mutability	Modifiers
mint	Public	Can modify state	-
addMinter	Public	Can modify state	onlyGov
removeMinter	Public	Can modify state	onlyGov
setPendingGov	External	Can modify state	onlyGov
acceptGov	External	Can modify state	-
_beforeTokenTransfer	Internal	Can modify state	

RewardsDistributionRecipient			
Function Name	Decorated	Mutability	Modifiers
notifyRewardAmount	External	Can modify state	-
setRewardDistribution	External	Can modify state	onlyOwner

StakingRewards			
Function Name	Decorated	Mutability	Modifiers
totalSupply	External	-	-
balanceOf	External	-	-
lastTimeRewardApplicable	Public	-	-
rewardPerToken	Public	-	-
earned	Public	-	-
getRewardForDuration	External	-	-
rewardPerWeight	Public	-	-
earnedByWeight	Public	-	-
getWeightForDuration	External	-	-
getDeltaWeight	Public	-	-
setReferrer	Internal	Can modify state	-
stake	External	Can modify state	nonReentrant updateReward
onWithdraw	Internal	Can modify state	-

withdraw	Public	Can modify state	nonReentrant updateReward
getReward	Public	Can modify state	nonReentrant updateReward
getRewardWeight	Public	Can modify state	nonReentrant updateReward
getRewardAll	Public	Can modify state	updateReward
exit	External	Can modify state	-
notifyRewardAmount	External	Can modify state	onlyRewardsDistribution updateReward
updateByDetal	Public	-	-
updateRewardInt	Internal	Can modify state	-
updateReferrerWeight	Internal	Can modify state	-

4.3 代码审计详情

4.3.1 增强建议

4.3.1.1 事件缺失

在 CloudLinkToken 合约中存在addMinterremoveMinter 函数，合约Gov 角色可以通过此函数更改minter 角色地址。但更改时并未进行事件记录。(StakingRewards 合约的setRewardDistribution 函数同样如此)。修复建议：建议更改合约角色时进行事件记录。

代码位置：CloudLinkToken, StakingRewards

```
function addMinter(address _minter) public onlyGov { minters[_minter] = true;
}

function removeMinter(address _minter) public onlyGov { minters[_minter] = false;
}

function setRewardDistribution(address _rewardDistribution) external
onlyOwner
rewardsDistribution = _rewardDistribution;
```

}

修复状态：未修复。

4.3.1.2 双份收益问题

在 StakingRewards 合约中用户在stake 时会通过 setReferrer 函数来设置推荐人，设置推荐人有进行检查：(balance==0&&weightAcc==0&&existingReferrer==address(0))

if(initialStakeTime[referrer]>0&&referrerMap[referrer]!=account)

正常情况下用户第一次 stake 时无法将推荐人设置成自己，但当用户首次 stake 时把推荐人设置为 0 地址， 然后再withdraw，此时满足判断条件因为 withdraw 操作之后balance, weightAc, existingReferrer 都为 0。所以用户再进行一次 stake 就可以把推荐人设置为自己。

修复建议：若非预期设计，建议对推荐人地址进行检查。

代码位置：StakingRewards

```
function setReferrer (address account, address referrer) internal returns (address) { uint256 balance = _balances[account];

uint256 weightAcc = weight[account];
address existingReferrer = referrerMap[account];

if  (balance == 0 && weightAcc == 0 && existingReferrer == address(0)) {      if (initialStakeTime[referrer] > 0 && referrerMap[referrer] != account) {
referrerMap[account] = referrer; existingReferrer = referrer;
}
}

return existingReferrer;
}
```

修复状态：经与项目方沟通，此为预期设计，且推荐人模块将不会启用，故忽略。

4.3.1.3 潜在的转账失败问题

StakingRewards 合约使用 safeTransferFrom 函数和 safeTransfer 函数分别将用户的抵押品转入转出合约，在调用 safeTransferFrom 函数和 safeTransfer 函数进行转账操作是其将对返回值进行检查。若抵押品合约的转账函数定义了返回值但没有按照标准规范进行 return(如波场的 USDT 合约)，将导致转账操作失败。

修复建议：建议后续再添加新池时应该注意确认转账函数是否

使用标准写法。 代码位置：StakingRewards

```
function callOptionalReturn(IERC20 token, bytes memory data)
private {
/Weneedtoperformalowlevelcallhere,tobypassSolidity'sretur
ndatasizecheckingmechanism,since
/ we'reimplementingitourselves.
/ ASolidityhighlevelcallhasthreeparts:
/1.Thetargetaddressischeckedtoverifyitcontainscontractcod
e
/  2.Thecallitselfismade,andsuccessasserted
/3.Thereturnvalueisdecoded,whichinturnchecksthesizeofther
eturneddata.
/ solhint-disable-next-linemax-line-length
require(address(token).isContract(), "SafeERC20: call to
non-contract");
/ solhint-disable-next-lineavoid-low-level-calls
(bool success, bytes memory returndata) =
address(token).call(data); require(success, "SafeERC20:
low-level call failed");
```

```
if (returndata.length > 0) {/ Returndataisoptional
/ solhint-disable-next-linemax-line-length
require(abi.decode(returndata, (bool)), "SafeERC20: ERC20
operation did not succeed");
}
}
```

修复状态：未修复。

5. 审计结果

5.1 总 结

审计结论 ： 通过

审计编号 ：0X202103310001

审计时间 ： 2021年4月5日

审计团队 ： 思维世纪团队

审计总结 ： 思维世纪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 3 个问题。其中包含 3 点增强建议。经过与项目方沟通反馈确认审计过程中发现的风险均已修复或在可承受范围内。

6.声明