智能合约安全审计报告

NewBFarm 智能合约安全审计报告

审计团队：零时科技安全团队

时间： 2021-05-06

NewBFarm智能合约安全审计报告

1.概述

零时科技安全团队于2021年04月30日，接到 **NewBFarm项目**的安全审计需求，团队与05月06日完成了 **NewBFarm智能合约**安全审计，审计过程中零时科技安全审计专家与NewBFarm项目相关接口人进行沟通，保持信息对称，在操作风险可控的情况下进行安全审计工作，尽量规避在测试过程中对项目产生和运营造成风险。

经过与NewBFarm项目方沟通反馈确认审计过程中发现的漏洞及风险均已修复或在可承受范围内，本次NewBFarm智能合约安全审计结果：通过安全审计。

合约报告MD5：C8B0FA3567CF8C7A966F916FBBFF0903

2.项目背景

2.1 项目简介

> 项目名称：NewBFarm
>
> 合约类型：代币合约，DeFi合约
>
> 代码语言： Solidity
>
> 项目官方Github：https://github.com/NewBFarm/vault
>
> 合约文件： Controller.sol，StakePool.sol，StrategyBake.sol，StrategyLp.sol，V2 strategy.sol，V2 strategy BNB.sol，Vault.sol

2.2 审计范围

NewBFarm官方提供的合约文件及对应MD5：

Vault.sol	6EE9C5385A9588A6899E4C174CE57C3D
StakePool.sol	3F836AD06057B2FEDDF755F12AC7D34C
Controller.sol	F2F25EF96D07DF94A13C3DCD76BAC287
StrategyLp.sol	4F82E4FBFAB50B55DDF0A598114F469F
V2 strategy.sol	F3D4844FD8102B5068E9E391368866D9
StrategyBake.sol	E9A61DE2514F5894A920339EF5FECE0B
V2 strategy BNB.sol	8094DCFEE63ED3081DF258124B1B7379

2.3 安全审计项

零时科技安全专家对约定内的安全审计项目进行安全审计，本次智能合约安全审计的范围，不包含未来可能出现的新型攻击方式，不包含合约升级活篡改后的代码，不包括在后续跨链部署时的操作过程，不包含项目前端代码安全与项目平台服务器安全。

本次智能合约安全审计项目包括如下：

- 整数溢出
- 重入攻击
- 浮点数和数值精度
- 默认可见性
- Tx.origin身份验证
- 错误的构造函数
- 未验证返回值
- 不安全的随机数
- 时间戳依赖
- 交易顺序依赖
- Delegatecall调用
- Call调用
- 拒绝服务
- 逻辑设计缺陷
- 假充值漏洞
- 短地址攻击
- 未初始化的存储指针
- 代币增发
- 冻结账户绕过
- 权限控制
- Gas使用

3.合约架构分析

3.1 目录结构

```
|
├─NewBFarm
|       Controller.sol
|       StakePool.sol
|       StrategyBake.sol
|       StrategyLp.sol
|       V2 strategy BNB.sol
|       V2 strategy.sol
|       Vault.sol
```

3.2 NewBFarm 合约

Contract

Controller

- setStrategist(address _strategist)
- setGovernance(address _governance)
- setVault(address _token, address _vault)

- approveStrategy(address _token, address _strategy)
- revokeStrategy(address _token, address _strategy)
- setConverter(address _input, address _output, address _converter)
- setStrategy(address _token, address _strategy)
- earn(address _token, uint _amount)
- balanceOf(address _token)
- withdrawAll(address _token)
- inCaseTokensGetStuck(address _token, uint _amount)
- inCaseStrategyTokenGetStuck(address _strategy, address _token)
- withdraw(address _token, uint _amount)
- rewards()

IRewardDistributionRecipient

- setRewardDistribution(address _rewardDistribution)

LPTokenWrapper

- totalSupply()
- balanceOf(address account)
- stake(uint256 amount)
- withdraw(uint256 amount)

USDTLavaRewards2

- lastTimeRewardApplicable()
- rewardPerToken()
- earned(address account)
- stake(uint256 amount)
- withdraw(uint256 amount)
- exit()
- getReward()
- notifyRewardAmount(uint256 reward)

StrategyBakeWar

- doApprove ()
- deposit()
- withdraw(IERC20 _asset)
- withdraw(uint _amount)
- withdrawAll()
- _withdrawAll()
- harvest()
- getNumOfRewards()
- doswap()
- dosplit()
- _withdrawSome(uint256 _amount)
- balanceOfWant()
- balanceOfPool()
- balanceOf()
- setGovernance(address _governance)
- setController(address _controller)
- setFee(uint256 _fee)
- setStrategyFee(uint256 _fee)
- setCallFee(uint256 _fee)
- setBurnFee(uint256 _fee)

- setBurnAddress(address _burnAddress)
- setWithdrawalFee(uint _withdrawalFee)

StrategyLp

- doApprove ()
- deposit()
- withdraw(IERC20 _asset)
- withdraw(uint _amount)
- withdrawAll()
- _withdrawAll()
- harvest()
- getNumOfRewards()
- doswap()
- dosplit()
- _withdrawSome(uint256 _amount)
- balanceOfWant()
- balanceOfPool()
- balanceOf()
- setGovernance(address _governance)
- setController(address _controller)
- setFee(uint256 _fee)
- setStrategyFee(uint256 _fee)
- setCallFee(uint256 _fee)
- setBurnFee(uint256 _fee)
- setBurnAddress(address _burnAddress)
- setWithdrawalFee(uint _withdrawalFee)

StrategyBNB

- setStrategist(address _strategist)
- setWithdrawalFee(uint256 _withdrawalFee)
- setStrategistReward(uint256 _strategistReward)
- setReinvest(uint256 _reinvest)
- deposit()
- withdraw(IERC20 _asset)
- withdraw(uint256 _amount)
- _withdrawSome(uint256 _amount)
- withdrawAll()
- _withdrawAll()
- harvest()
- balanceOfWant()
- balanceOfPool()
- balanceOf()
- setGovernance(address _governance)
- setController(address _controller)

StrategyBUSD

- addFarmer(address f)
- removeFarmer(address f)
- setGovernance(address _governance)
- setStrategist(address _strategist)
- setWithdrawalFee(uint256 _withdrawalFee)
- setStrategistReward(uint256 _strategistReward)

- setReinvest(uint256 _reinvest)
- getSuppliedView()
- getBorrowedView()
- balanceOfPool()
- balanceOfWant()
- balanceOf()
- getSupplied()
- getBorrowed()
- harvest()
- deposit()
- _deposit()
- _withdrawSome(uint256 _amount)
- withdrawAll()
- _withdrawAll()
- withdraw(uint256 _amount)
- withdraw(IERC20 _asset)
- e_exit()
- e_redeem(uint amount)
- e_redeemAll()
- e_redeemCToken(uint amount)
- e_redeemAllCToken()
- e_repayAll()
- e_repay(uint amount)
- e_collect()

zVault

- balance()
- setMin(uint _min)
- setGovernance(address _governance)
- setController(address _controller)
- setEarnLowerlimit(uint256 _earnLowerlimit)
- available()
- earn()
- depositAll()
- deposit(uint _amount)
- withdrawAll()
- withdraw(uint _shares)
- getPricePerFullShare()

4.审计详情

4.1 漏洞分布

本次安全审计漏洞风险按危险等级分布：

漏洞风险等级分布			
高危	中危	低危	通过
0	0	0	21

本次智能合约安全审计高危漏洞0个，中危0个，低危0个，通过21个，安全等级高。

4.2 漏洞详情

对约定内的智能合约进行安全审计，未发现可以直接利用并产生安全问题的安全漏洞，通过安全审计。

4.3 其他风险

其他风险是指合约安全审计人员认为有风险的代码，在特定情况下可能会影响项目稳定性，但不能构成直接危害的安全问题。

4.3.1 tx.origin变量使用安全

- **问题点**

NewBFarm合约，使用全局变量tx.origin将初始调用者赋予管理员权限，由于tx.origin遍历整个调用栈并返回最初发送调用的帐户地址，在调用过程中可能发生当前合约被中间合约调用。

```
    constructor() public {
        governance = tx.origin;
    }
```

- **安全建议**

建议将tx.origin变量改为调用者自己msg.sender。

- **修复状态**

通过与NewBFarm团队沟通，已采用安全建议。

4.3.2 地址有效性校验

- **问题点**

NewBFarm合约，多个方法输入地址未做有效性校验，可能出现空地址，如下图所示：

```
    function setGovernance(address _governance)

    function setVault(address _token, address _vault)

    function setConverter(address _input, address _output, address _converter)

    function setStrategy(address _token, address _strategy)

    function earn(address _token, uint _amount)

    function balanceOf(address _token)

    function withdrawAll(address _token)

    function inCaseTokensGetStuck(address _token, uint _amount)

    function withdraw(address _token, uint _amount)
```

- **安全建议**

建议添加检查地址有效性校验代码，如下所示：

```
    function setGovernance(address _governance)
        require(_governance != address(0) , 'ADDRESS ERROR!!!');
```

- **修复状态**

通过与NewBFarm团队沟通，已采用安全建议。

4.3.3 管理员权限过大

- **问题点**

NewBFarm合约，管理员可以设置多个数值并且存在转账功能，如果该管理员被恶意人员操控，就可导致异常资金流失及市场稳定性动摇，如下代码所示：

```
    function withdrawAll(address _token) public {
        require(msg.sender == governance, "!governance");
        Strategy(strategies[_token]).withdrawAll();
    }

    function inCaseTokensGetStuck(address _token, uint _amount) public {
        require(msg.sender == governance, "!governance");
        IERC20(_token).safeTransfer(governance, _amount);
    }
```

- **安全建议**

在保证安全的前提下，多份合理的保存私钥。

- **修复状态**

通过与NewBFarm团队沟通，已采用安全建议。

4.3.4 函数属性问题

- **问题点**

NewBFarm合约，多个方法存在授权功能，由于避免授权冲突，每次授权时会将授权额度改为0，之后修改为对应值。由于两个方法均为public属性，这里是否存在任意地址调用，造成项目其他功能不稳定。

```
    function doApprove () public{
        IERC20(output).safeApprove(unirouter, 0);
        IERC20(output).safeApprove(unirouter, uint(-1));
    }
```

- **安全建议**

建议以上方法添加管理员修饰器。

- **修复状态**

通过与NewBFarm团队沟通，已采用安全建议。

4.3.5 amount数值有效性安全

- **问题点**

NewBFarm合约，多个方法输入的_amount数值并未进行有效性判断，应判断该值是否大于0。

```
    function deposit(uint _amount) public {
        uint _pool = balance();

    function withdraw(uint _shares) public {
        uint r = (balance().mul(_shares)).div(totalSupply());
```

- **安全建议**

在函数内部判断_amount是否有效，如下图所示：

```
require(_amount > 0, erroramount);   //添加require判断
```

- **修复状态**

通过与NewBFarm团队沟通，已采用安全建议。

4.3.6 销毁数量问题

- **问题点**

NewBFarm合约，withdraw()方法中，用户输入任意数值就可进行销毁操作，应对销毁的数量进行正确性校验。

```
    function withdraw(uint _shares) public {
        uint r = (balance().mul(_shares)).div(totalSupply());
        _burn(msg.sender, _shares);
```

- **安全建议**

对销毁数量进行严格判定。

- **修复状态**

通过与NewBFarm团队沟通，已采用安全建议。

4.3.7 正常铸币问题

- **问题点**

NewBFarm合约，deposit()方法中，用户在进行存款操作后，deposit代码逻辑会给调用者地址进行铸币操作，铸币的数值为shares，该值通过铸币totalSupply总量和余额方法balance()中的两个余额组成，是否存在totalSupply和balance被恶意控制，导致shares值增大，从而获取大量铸币。

```
    function deposit(uint _amount) public {
        uint _pool = balance();
        uint _before = token.balanceOf(address(this));
        token.safeTransferFrom(msg.sender, address(this), _amount);
        uint _after = token.balanceOf(address(this));
        _amount = _after.sub(_before);
        uint shares = 0;
        if (totalSupply() == 0) {
            shares = _amount;
        } else {
            shares = (_amount.mul(totalSupply())).div(_pool);
        }
        _mint(msg.sender, shares);
        if (token.balanceOf(address(this))>earnLowerlimit){
          earn();
        }
    }
```

- **安全建议**

对铸币数量进行严格判定。

- **修复状态**

通过与NewBFarm团队沟通，无影响。

4.3.8 回退函数使用安全

- **问题点**

NewBFarm合约，使用回退函数fallback()，该函数使用payable修饰，并未有其他功能。

```
receive() payable external {}
```

- **安全建议**

如果回退函数没有特殊逻辑实现，建议移除该函数，避免安全问题发生。

- **修复状态**

通过与NewBFarm团队沟通，已采用安全建议。

5.安全审计工具

工具名称	功能
Oyente	可以用来检测智能合约中常见bug
securify	可以验证以太坊智能合约的常见类型
MAIAN	可以查找多个智能合约漏洞并进行分类
零时内部工具包	零时(鹰眼系统)自研发工具包+https://audit.noneage.com

6.漏洞风险评估标准

漏洞等级	漏洞风险描述
高危	能直接导致代币合约或者用户数字资产损失的漏洞，比如：整数溢出漏洞、假充值漏洞、重入漏洞、代币违规增发等。 能直接造成代币合约所属权变更或者验证绕过的漏洞，比如：权限验证绕过、call代码注入、变量覆盖、未验证返回值等。 能直接导致代币正常工作的漏洞，比如：拒绝服务漏洞、不安全的随机数等。
中危	需要一定条件才能触发的漏洞，比如代币所有者高权限触发的漏洞，交易顺序依赖漏洞等。 不能直接造成资产损失的漏洞，比如函数默认可见性错误漏洞，逻辑设计缺陷漏洞等。
低危	难以触发的漏洞，或者不能导致资产损失的漏洞，比如需要高于攻击收益的代价才能触发的漏洞 无法导致安全漏洞的错误编码问题。

零时科技
NONEAGE

咨询电话：86-17391945345 18511993344
邮　　箱：support@noneage.com
官　　网：www.noneage.com
微　　博：weibo.com/noneage