freshcoins

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

pETH

$PETH

Stake Smart Contract

31/03/2023

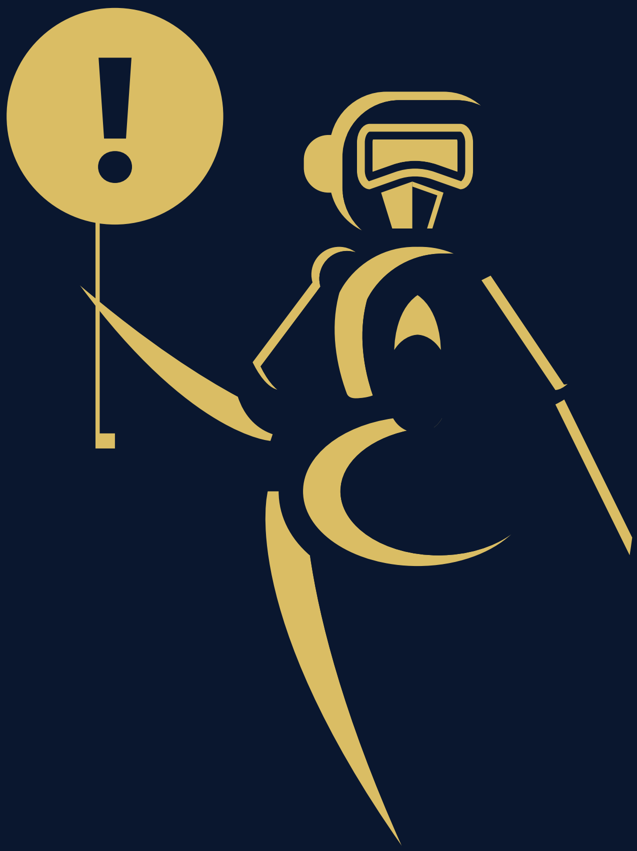
TABLE OF CONTENTS

DISCLAIMER

The information provided on this analysis document is only for general information and should not be used as a reason to invest.

FreshCoins Team will take no payment for manipulating the results of this audit.

The score and the result will stay on this project page information on our website https://freshcoins.io
FreshCoins Team does not guarantees that a project will not sell off team supply, or any other scam strategy (RUG or Honeypot etc)



INTRODUCTION

FreshCoins (Consultant) was contracted by GenesisRewardPool - Stake Smart Contract (Customer) to conduct a Smart Contract Code Review and Security Analysis.

0x79500dc1F7Bef37896cc3e70986d824e7A13102b

Network: Arbitrum

This report presents the findings of the security assessment of Customer's smart contract and its code review conducted on 31/03/2023

WEBSITE DIAGNOSTIC

https://app.palacefinance.io/farm

🟥 0-49

🟨 50-89

🟩 90-100

92	90	82	90	NA
Performance	Accessibility	Best Practices	SEO	Progressive Web App

Metrics

🟩 First Contentful Paint
3.2 s

🟥 Time to interactive
10.4 s

🟥 Speed Index
12.2 s

🟩 Total Blocking Time
344 ms

🟥 Large Contentful Paint
6.5 s

🟩 Cumulative Layout Shift
0.019

WEBSITE IMPROVEMENTS

Reduce unused CSS

Reduce unused JavaScript

Ensure text remains visible during webfont load

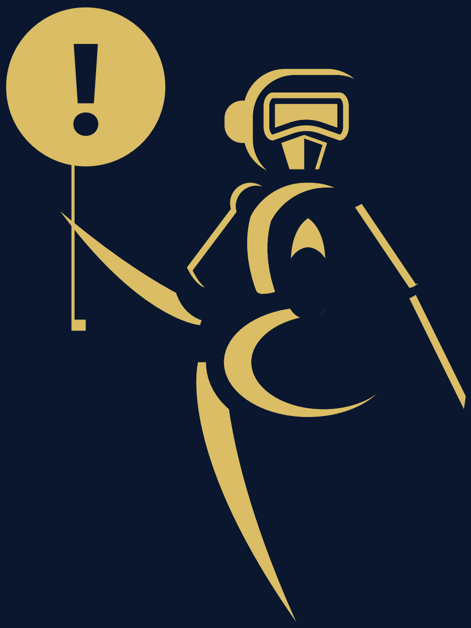
Image elements do not have explicit width and height

Reduce JavaScript execution time 2.0 s

Image elements do not have [alt] attributes

Links do not have a discernible name

Heading elements are not in a sequentially-descending order



AUDIT OVERVIEW

84

Security Score

94

Static Scan
Automatic scanning for common vulnerabilities

80

ERC Scan
Automatic checks for ERC's conformance

0	High
2	Medium
0	Low
0	Optimizations
0	Informational

No.	Issue description	Checking Status
1	Compiler Errors / Warnings	Passed
2	Reentrancy and Cross-function	Passed
3	Front running	Passed
4	Timestamp dependence	Passed
5	Integer Overflow and Underflow	Passed
6	Reverted DoS	Passed
7	DoS with block gas limit	Passed
8	Methods execution permissions	Passed
9	Exchange rate impact	Passed
10	Malicious Event	Passed
11	Scoping and Declarations	Passed
12	Uninitialized storage pointers	Passed
13	Design Logic	Passed
14	Safe Zeppelin module	Passed

PRIVILEGES

Stake/unstake NFT functions

```
uint256 public constant DISCOUNT_RATE = 2000;
uint256 public constant DENOMINATOR = 10000;
uint256 public constant NFT_LOCKS = 15 days;
```

```
function stakeNFT(uint256 _pid, uint256 _nftCount) public {
        require(_pid < poolInfo.length, "Invalid pool id");
        require(_nftCount > 0, "NFT count should be greater than 0");
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        NftUserInfo storage nftUser = nftUserInfo[_pid][msg.sender];
        updatePool(_pid);

        uint256 poolTokenType = pool.tokenType;
        require(poolTokenType < 2, "NFTs can't be staked into the LP token pool");

        uint256 nftBalance = nftToken.balanceOf(msg.sender);
        require(nftBalance >= _nftCount, "NFT balance shold be greater than count");

        for(uint256 i = 0; i < _nftCount; i++) {
          uint256 tokenId = nftToken.tokenOfOwnerByIndex(msg.sender, i);
          nftToken.transferFrom(msg.sender, address(this), tokenId);
          nftUser.nftTokenIds.push(tokenId);
        }
        nftUser.nftAmount = nftUser.nftAmount.add(_nftCount);
        nftUser.stakedTS = block.timestamp;
        uint256 kBoost = getKBoost(_pid, msg.sender);
        uint256 nftPrice = ILKEY(address(nftToken)).getNFTPriceInToken(address(pool.token));
        uint256 _nftDiscountValue = nftPrice.mul(_nftCount).mul(DISCOUNT_RATE).div(DENOMINATOR);
        user.amount = user.amount.add(_nftDiscountValue);
        uint256 oldBoostedAmount = user.boostedAmount;
        user.boostedAmount = user.amount.mul(DENOMINATOR.add(kBoost)).div(DENOMINATOR);
        pool.boostedSupply = pool.boostedSupply.add(user.boostedAmount).sub(oldBoostedAmount);
}
```

```
function unstakeNFT(uint256 _pid) public {
        NftUserInfo storage nftUser = nftUserInfo[_pid][msg.sender];
        uint256 nftStakedTS = nftUser.stakedTS;
        require(nftStakedTS + NFT_LOCKS >= block.timestamp, "Can stake after locks up from last staked time" );
        for (uint256 i = 0; i < nftUser.nftAmount; i++) {
          uint256 tokenId = nftUser.nftTokenIds[nftUser.nftAmount - i - 1];
          nftToken.transferFrom(address(this), msg.sender, tokenId);
          nftUser.nftTokenIds.pop();
        }
        nftUser.nftAmount = 0;
}
```

Participants are prohibited from withdrawing until 15 days have elapsed since their initial deposit.

Contract owner can supply pool and set fee up to 100%

```
function add(uint256 _allocPoint, IERC20 _token, uint256 _depositFee, bool _withUpdate, uint256 _lastRewardTime, uint256 _tokenType) public onlyOwner {
    checkPoolDuplicate(_token);
    if (_withUpdate) {
        massUpdatePools();
    }
    if (block.timestamp < poolStartTime) {
        // chef is sleeping
        if (_lastRewardTime == 0) {
            _lastRewardTime = poolStartTime;
        } else {
            if (_lastRewardTime < poolStartTime) {
                _lastRewardTime = poolStartTime;
            }
        }
    } else {
        // chef is cooking
        if (_lastRewardTime == 0 || _lastRewardTime < block.timestamp) {
            _lastRewardTime = block.timestamp;
        }
    }
    bool _isStarted = (_lastRewardTime <= poolStartTime) || (_lastRewardTime <= block.timestamp);
    poolInfo.push(
        PoolInfo({
            token: _token,
            tokenType: _tokenType,
            allocPoint: _allocPoint,
            depositFee: _depositFee,
            lastRewardTime: _lastRewardTime,
            accPETHPerShare: 0,
            boostedSupply: 0,
            isStarted: _isStarted}));
    if (_isStarted) {
        totalAllocPoint = totalAllocPoint.add(_allocPoint);
    }
}
```

Contract owner can change pool settings and set fee up to 100%

```
function set(uint256 _pid, uint256 _allocPoint, uint256 _depositFee) public onlyOwner {
    require(_depositFee < 10000, "deposit fee should be less than 10000");

    massUpdatePools();
    PoolInfo storage pool = poolInfo[_pid];
    if (pool.isStarted) {
        totalAllocPoint = totalAllocPoint.sub(pool.allocPoint).add(_allocPoint);
    }
    pool.allocPoint = _allocPoint;
    pool.depositFee = _depositFee;
}
```

Deposit/withdraw tokens

```
uint256 public constant REFERRAL_COMMISSION_RATE = 3000;
uint256 public constant DENOMINATOR = 10000;
```

```
function deposit(uint256 _pid, uint256 _amount, address _referrer) public {
    address _sender = msg.sender;
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_sender];
    updatePool(_pid);
    if (_amount > 0 && address(referral) != address(0) && _referrer != address(0) && _referrer != msg.sender) {
        referral.recordReferral(msg.sender, _referrer);
    }
    uint256 kBoost = getKBoost(_pid, msg.sender);
    if (user.amount > 0) {
        uint256 boostedUserAmount = user.boostedAmount;
        uint256 _pending = boostedUserAmount.mul(pool.accPETHPerShare).div(1e18).sub(user.rewardDebt);
        if (_pending > 0) {
            safePETHTransfer(_sender, _pending);
            emit RewardPaid(_sender, _pending);
        }
    }
    if (_amount > 0) {
        pool.token.safeTransferFrom(_sender, address(this), _amount);
        uint256 _boostingAmount = (_amount).mul(DENOMINATOR.add(kBoost)).div(DENOMINATOR);
        user.amount = user.amount.add(_amount);
        user.boostedAmount = user.boostedAmount.add(_boostingAmount);
        if (pool.depositFee > 0) {
            uint256 feeAmount = _amount.mul(pool.depositFee).div(DENOMINATOR);
            uint256 referralCommissionAmount = 0;
            if(_referrer != address(0) && _referrer != msg.sender) {
                referralCommissionAmount = feeAmount.mul(REFERRAL_COMMISSION_RATE).div(DENOMINATOR);
                payReferralCommission(_sender, pool.token, referralCommissionAmount);
            }
            pool.token.safeTransfer(feeCollector, feeAmount.sub(referralCommissionAmount));
            user.amount = user.amount.sub(feeAmount);
        }
        pool.boostedSupply = pool.boostedSupply.add(_boostingAmount);
    }
    user.rewardDebt = user.boostedAmount.mul(pool.accPETHPerShare).div(1e18);
    emit Deposit(_sender, _pid, _amount);
}
```

```
function withdraw(uint256 _pid, uint256 _amount) public {
    address _sender = msg.sender;
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][_sender];

    require(user.amount >= _amount, "withdraw: not good");
    updatePool(_pid);
    uint256 kBoost = getKBoost(_pid, msg.sender);
    uint256 _pending = user.boostedAmount.mul(pool.accPETHPerShare).div(1e18).sub(user.rewardDebt);
    if (_pending > 0) {
        safePETHTransfer(_sender, _pending);
```

```
            emit RewardPaid(_sender, _pending);
        }
        if (_amount > 0) {
            uint256 _boostingAmount = (_amount).mul(DENOMINATOR.add(kBoost)).div(DENOMINATOR);
            user.amount = user.amount.sub(_amount);
            user.boostedAmount = user.boostedAmount.sub(_boostingAmount);
            pool.boostedSupply = pool.boostedSupply.sub(_boostingAmount);
            pool.token.safeTransfer(_sender, _amount);
        }
        user.rewardDebt = user.boostedAmount.mul(pool.accPETHPerShare).div(1e18);
        emit Withdraw(_sender, _pid, _amount);
    }
```

```
function emergencyWithdraw(uint256 _pid) public {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        uint256 _amount = user.amount;
        user.amount = 0;
        user.rewardDebt = 0;
        pool.boostedSupply = pool.boostedSupply.sub(user.boostedAmount);
        pool.token.safeTransfer(msg.sender, _amount);
        emit EmergencyWithdraw(msg.sender, _pid, _amount);
}
```

● Contract owner can change feeCollector address

Current value: 0xfb7af288da5cee71dccc018e5b00af6a977639a1

```
function setFeeCollector(address _feeCollector) public onlyOwner {
        require(_feeCollector != address(0), "Fee collector should be non-zero address");
        feeCollector = _feeCollector;
}
```

Update pool public functions (mass update)

```
function massUpdatePools() public {
    uint256 length = poolInfo.length;
    for (uint256 pid = 0; pid < length; ++pid) {
        updatePool(pid);
    }
}

function updatePool(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    if (block.timestamp <= pool.lastRewardTime) {
        return;
    }
    // uint256 tokenSupply = pool.token.balanceOf(address(this));
    uint256 boostedTokenSupply = pool.boostedSupply;
    if (boostedTokenSupply == 0) {
        pool.lastRewardTime = block.timestamp;
        return;
    }
    if (!pool.isStarted) {
        pool.isStarted = true;
        totalAllocPoint = totalAllocPoint.add(pool.allocPoint);
    }
    if (totalAllocPoint > 0) {
        uint256 _generatedReward = getGeneratedReward(pool.lastRewardTime, block.timestamp);
        uint256 _PETHReward = _generatedReward.mul(pool.allocPoint).div(totalAllocPoint);
        pool.accPETHPerShare = pool.accPETHPerShare.add(_PETHReward.mul(1e18).div(boostedTokenSupply));
    }
    pool.lastRewardTime = block.timestamp;
}
```

Contract owner can renounce ownership

```
function renounceOwnership() public virtual onlyOwner {
    _transferOwnership(address(0));
}
```

Contract owner can transfer ownership

```
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(newOwner != address(0), "Ownable: new owner is the zero address");
    _transferOwnership(newOwner);
}

function _transferOwnership(address newOwner) internal virtual {
    address oldOwner = _owner;
    _owner = newOwner;
    emit OwnershipTransferred(oldOwner, newOwner);
}
```

CONCLUSION AND ANALYSIS

Smart Contracts within the scope were manually reviewed and analyzed with static tools.

Audit report overview contains all found security vulnerabilities and other issues in the reviewed code.

Found no HIGH issues during the first review.

TECHNICAL DISCLAIMER

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. The audit can’t guarantee the explicit security of the audited project / smart contract.