HAECHI AUDIT

Luxon - Crystalstaking

Smart Contract Security Analysis

Published on : Oct 25, 2022

Version v1.1

HAECHI AUDIT

Smart Contract Audit Certificate

Luxon - Crystalstaking

Security Report Published by HAECHI AUDIT
v1.1 Oct 25, 2022

Auditor : Jade Han, Jeremy Lim

hojung han [illegible]

Found issues

Severity of Issues	Findings	Resolved	Acknowledged	Comment
Critical	1	1	-	-
High	-	-	-	-
Medium	1	1	-	-
Low	1	-	1	-
Tips	7	7	-	-

TABLE OF CONTENTS

ABOUT US

The most reliable web3 security partner.

HAECHI AUDIT is a flagship service of HAECHI LABS, the leader of the global blockchain industry. We bring together the best Web2 and Web3 experts. Security Researchers with expertise in cryptography, leaders of the global best hacker team, and blockchain/smart contract experts are responsible for securing your Web3 service.

We have secured the most well-known web3 services including 1inch, SushiSwap, Klaytn, Badger DAO, SuperRare, Netmarble, Klaytn and Chainsafe. We have secured $60B crypto assets on over 400 main-nets, Defi protocols, NFT services, P2E, and Bridges.

HAECHI AUDIT is the only blockchain technology company selected for the Samsung Electronics Startup Incubation Program in recognition of our expertise. We have also received technology grants from the Ethereum Foundation and Ethereum Community Fund.

Inquiries: audit@haechi.io

Website: audit.haechi.io

Executive Summary

Purpose of this report

This report was prepared to audit the security of the Crypstalstaking contracts developed by the Luxon team. HAECHI AUDIT conducted the audit focusing on whether the system created by the Luxon team is soundly implemented and designed as specified in the published materials, in addition to the safety and security of the Crystalstaking.
In detail, we have focused on the following

- Project availability issues like Denial of Service.
- Storage variable access control.
- Function access control
- Staking Token and Reward Token theft
- Reward amount manipulate
- Gas Optimization
- Unhandled Exception

Codebase Submitted for the Audit

The codes used in this Audit can be found on GitHub (https://github.com/nerdy-star/nerdy_smart_contract/blob/haechi-labs/crystal-staking/).

The last commit of the code used for this Audit is "6dfc1702b62bece149ce846745c6224840177fbb".

Audit Timeline

Date	Event
2022/10/04	Audit Initiation
2022/10/18	Delivery of v1.0 report.

Findings

HAECHI AUDIT found 1 critical and 1 medium. There are 7 Tips issues explained that would improve the code's usability or efficiency upon modification.

Severity	Issue	Status
Critical	Reward Token fraudulent receiving	(Found - v1.0)
Medium	updateUserReward() method will crashed by DoS caused by stake() method	(Found - v1.0)
Low	fee collection bypass before calling updateUserReward()	(Found - v1.0)
TIPS	stake() method gas optimization issue	(Found - v1.0)
TIPS	disabledLockUp() implementation problem	(Found - v1.0)
TIPS	removeByIndex() gas optimization issue	(Found - v1.0)
TIPS	removeByIndexUnStaker() gas optimization issue	(Found - v1.0)
TIPS	unStake() function implements stakingUsers delete logics invalid	(Found - v1.0)
TIPS	claim() function use reversed equal sign	(Found - v1.0)
TIPS	unStake() method does not check _amount is zero	(Found - v1.0)

OVERVIEW

Protocol overview

▪ **MemorialCrystal Token**

The contract implements ERC1155 Token, and the owner (Owner) can change the information of ERC1155 Token and airdrop and mint to any user.

▪ **CrystalStaking**

Users can staking the issued MemorialCrystal token and receive reward tokens for the duration of the staking. Reward tokens are periodically supplied to the contract by the admin.

Scope

```
├── Admin
│   ├── LuxOnAdmin.sol
│   └── LuxOnService.sol
└── LUXON
    ├── myPage
    │   ├── token
    │   │   └── DSP.sol
    │   └── inventory
    │       ├── crystal
    │       │   └── MemorialCrystal.sol
    │       └── crystalStaking.sol
    └── utils
        ├── ERC20LUXON.sol
        ├── ERC20LXN.sol
        ├── ERC721LUXON.sol
        ├── ERC1155LUXON.sol
        ├── FindTokenList.sol
        ├── IERC20LUXON.sol
        ├── IERC721LUXON.sol
        ├── LuxOnLive.sol
        ├── LuxOnSuperOperators.sol
        ├── Payback.sol
        ├── RealOwner.sol
        ├── Staking.sol
        └── SuperOperators.sol
```

18 files, 1575 lines

Access Controls

Crystalstaking contracts have the following access control mechanisms.

- onlyOwner()
- onlySuperOperator()

onlyOwner() : modifier that controls access to variables including contract addresses that communicate with each other, operator address, and settings related to Payback and Service Live Status and so on.

- LuxOnAdmin#setSuperOperator
- LuxOnSuperOperators#setLuxOnAdmin
- LuxOnSuperOperators#setOperator
- LuxOnService#setInspection
- LuxOnLive#setLuxOnService
- crystalStaking#setTokenId
- crystalStaking#setTokenIds
- crystalStaking#setWeeklySupply
- crystalStaking#setRewardsPerPeriod
- crystalStaking#setLockUpDay
- crystalStaking#setLockUpFee
- crystalStaking#getFee
- crystalStaking#updateUserReward
- MemorialCrystal#getLastSeason
- MemorialCrystal#addNewSeason
- MemorialCrystal#setName
- MemorialCrystal#setSymbol
- MemorialCrystal#setURI
- MemorialCrystal#pause
- MemorialCrystal#unpause
- MemorialCrystal#mintBatch
- MemorialCrystal#airdropSingle
- MemorialCrystal#airdrop
- ERC20LUXON#setPaybackFrom
- ERC721LUXON#setBaseURI
- ERC1155LUXON#setName
- ERC1155LUXON#setSymbol
- ERC1155LUXON#setURI
- Payback#transferTokenAddress
- Payback#transferPaybackPercentageRate

- Payback#transferPaybackPercentage
- Staking#setStakingContract
- Staking#deleteStakingContract
- SuperOperator#setSuperOperator

onlySuperOperator() : modifier that controls access to mint and burn functions and staking reward token funding and so on.

- crystalStaking#fundRewardToken
- crystalStaking#refundRewardToken
- MemorialCrystal#mint
- ERC20LUXON#paybackByMint
- ERC20LUXON#paybackByTransfer
- ERC20LXN#burnFor
- ERC1155LUXON#mint
- ERC1155LUXON#mintBatch
- ERC1155LUXON#burn
- ERC1155LUXON#burnBatch
- RealOwner#setRealOwner

The owner has permissions that can change the crucial part of the system. It is highly recommended to maintain the private key as securely as possible and strictly monitor the system state changes.

FINDINGS

1. Reward Token fraudulent receiving

ID: Nerdystar-crystalstaking-01 Severity: Critical

Type: Logic Error Difficulty: Low

File: LUXON/myPage/inventory/crystalStaking.sol

Issue

Most of the allocated reward tokens can be received by one person illegally.

```
function calculateRewards(address userAddr, uint256 tokenId) internal view returns
(uint256) {
        address[] memory users = stakingUsers[tokenId];
        for (uint i = 0; i < users.length; i++) {
            if (userAddr == users[i]) {,
                return weeklySupply / rewardsPerPeriod / users.length;
            }
        }
        return 0;
    }

    // macro
    function updateUserReward() public onlyOwner {
        for(uint i = 0; i < userList.length; i++) {
            address userAddress = userList[i];
            if (address(0) == userAddress) {
                continue;
            }

            for (uint j = 0; j < stakers[userAddress].stakedTokens.length; j++) {
                uint256 updateReward = calculateRewards(userAddress,
stakers[userAddress].stakedTokens[j].tokenId);
                stakers[userAddress].unclaimedRewards =
stakers[userAddress].unclaimedRewards.add(updateReward);
                stakers[userAddress].cumulativeAmount =
stakers[userAddress].cumulativeAmount.add(updateReward);
            }
        }
    }
```

[https://github.com/nerdy-star/nerdy_smart_contract/blob/haechi-labs/crystal-staking/contracts/LUXON/myPage/inventory/crystalStaking.sol#L338]

The amount of tokens the user has staked is ignored in the formula for calculating the amount of reward tokens the user will receive.

Therefore, if a malicious user owns several ERC1155 Tokens, divides them into dozens or hundreds of wallets, staking one ERC1155 Token for each wallet, and receives reward tokens, malicious users will be able to take most of the allocated reward tokens.

Recommendation

The total amount of user rewards accumulated by the updateUserReward() function is determined by the number of user's staking counting, not by the total amount of tokens that have been staked.
For this reason, if there is a user who has staked 2000 tokens, if other users have staked 1 token a total of 2 times, the total amount of staking will be higher for the first user, but the second user will receive more rewards to be taken.
For this reason, it is necessary to set reward calculations that reflect the total amount of tokens that the user has staked, so that it varies according to the staked token amount ratio when providing rewards.

Update

```
function calculateRewards(address userAddr) internal view returns (uint256) {
        // (주간 보상량 / 28 / 현재 스테이킹 총 개수) * 유저 스테이킹 개수
        return (weeklySupply / rewardsPerPeriod / tokenTotalAmount) *
stakers[userAddr].totalAmount;
    }

    // macro
function updateUserReward() public onlyOwner {
        for (uint i = 0; i < userList.length; i++) {
            address userAddress = userList[i];
            if (address(0) == userAddress) {
                continue;
            }

            uint256 updateReward = calculateRewards(userAddress);
            stakers[userAddress].unclaimedRewards =
stakers[userAddress].unclaimedRewards.add(updateReward);
            stakers[userAddress].cumulativeAmount =
stakers[userAddress].cumulativeAmount.add(updateReward);
        }
}
```

Internally, revised implementation removes double for loop statement, and then changed the existing searching stakedTokens logic to search the member from userList to prevent duplicated reward reception. and the calculateRewards() function changed implementation to calculate corresponding rewards from user staked tokens.

2. updateUserReward() method will crashed by DoS caused by stake() method

ID: Nerdystar-crystalstaking-02　　Severity: Medium
Type: Logic Error　　Difficulty: Low
File: LUXON/myPage/inventory/crystalStaking.sol

Issue

When updateUserReward() is executed, the contract may not be able to provide rewards to users due to gas griefing from the owner.

```
    function stake(uint256 _tokenId, uint256 _amount) public nonReentrant {
        require(memorialCrystal.balanceOf(msg.sender, _tokenId) > _amount, "you require more
memorial-crystal");

        bool exist = findUserByTokenId(msg.sender, _tokenId);
        if (false == exist) {
            stakingUsers[_tokenId].push(msg.sender);
        }

        bool isNew = true;
        if (0 == stakers[msg.sender].totalAmount) {
            userList.push(msg.sender);
            addStakeInfo(msg.sender, _tokenId, _amount);
        } else {
            // If token ID you have
            for (uint i = 0; i < stakers[msg.sender].stakedTokens.length; i++) {
                if (_tokenId == stakers[msg.sender].stakedTokens[i].tokenId) {
                    addStakeInfo(msg.sender, _tokenId, _amount);
                    isNew = false;
                }
            }

            // If new token ID
            if (isNew) {
                addStakeInfo(msg.sender, _tokenId, _amount);
            }
        }

        memorialCrystal.safeTransferFrom(
            msg.sender,
            address(this),
            _tokenId,
            _amount,
            ""
        );

        emit Stake(msg.sender, _tokenId, _amount, block.timestamp);
    }
```

[https://github.com/nerdy-star/nerdy_smart_contract/blob/haechi-labs/crystal-staking/contracts/LUXON/myPage/inventory/crystalStaking.sol#L76]

The stake function takes _tokenId and _amount as parameters.

Since there is no logic to check whether the parameter _amount is 0, a malicious user who has only one token can infinitely call the stake function with the parameter _amount equal to 0.

As a result, by the addStakeInfo function, meaningless dummy data can be infinitely put into stakedTokens and userList.

```
    function updateUserReward() public onlyOwner {
        for(uint i = 0; i < userList.length; i++) {
            address userAddress = userList[i];
            if (address(0) == userAddress) {
                continue;
            }

            for (uint j = 0; j < stakers[userAddress].stakedTokens.length; j++) {
                uint256 updateReward = calculateRewards(userAddress,
stakers[userAddress].stakedTokens[j].tokenId);
                stakers[userAddress].unclaimedRewards =
stakers[userAddress].unclaimedRewards.add(updateReward);
                stakers[userAddress].cumulativeAmount =
stakers[userAddress].cumulativeAmount.add(updateReward);
            }
        }
    }
```

[https://github.com/nerdy-star/nerdy_smart_contract/blob/haechi-labs/crystal-staking/contracts/LUXON/myPage/inventory/crystalStaking.sol#L349]

Because of this, when the owner executes the updateUserReward() function to update the reward, the code is executed while traversing all users and all tokens, which may cause gas gripping problems.

Due to meaningless dummy data, the number of loops in the updateUserReward function increases and gas may be consumed excessively.

Recommendation

Add logic to check if _amount is 0 or not.

Update

```
function stake(uint256 _tokenId, uint256 _amount) public nonReentrant {
        require(memorialCrystal.balanceOf(msg.sender, _tokenId) > _amount, "you require
more memorial-crystal");
```

```
        require(0 != _amount, "invalid amount");

        …
}
```

In the previous implementation, it was not possible to check whether _amount is 0, so the DoS at the time of reward payment, which could be caused by adding too many duplicate structure instances, was patched by adding logic to check whether _amount is 0.

3. fee collection bypass before calling updateUserReward()

ID: Nerdystar-crystalstaking-03
Severity: Low
Type: Logic Error
Difficulty: Low
File: LUXON/myPage/inventory/crystalStaking.sol

Issue

When calling the unStake function with _amount set to 0, the user can add dummy data to unStakers. A malicious user can bypass fee collection before calling updateUserReward function.

```
function disabledLockUp(uint256[] memory _indexes) public {
        // 해제 비용 계산식
        // DSP 누적 정산량 / 언스테키잉 당시 토큰 전체 개수 / 락업 전체 일 수 * 락업 진행 일 수 / 10 *
언스테이킹 토큰 개수;

        for (uint i = 0; i < _indexes.length; i++) {
            UnStaker memory unstaker = unStakers[msg.sender][_indexes[i]];
            Staker memory staker = stakers[msg.sender];

            uint256 remainLockupTime = unstaker.endAt - unstaker.createAt;
            uint256 fee = staker.cumulativeAmount / unstaker.totalTokenAmount / _lockupDay *
(remainLockupTime % DAY) / lockUpFeePercentage * unstaker.amount;

            cumulativeFee = cumulativeFee.add(fee);
            uint256 subCumulativeAmount = staker.cumulativeAmount / unstaker.amount; // 100%
            stakers[msg.sender].cumulativeAmount = staker.cumulativeAmount.sub(subCumulativeAmount);
            rewardsToken.safeTransferFrom(
                msg.sender,
                address(this),
                fee
            );
            emit DisabledLockUp(msg.sender, unStakers[msg.sender][_indexes[i]].tokenId,
unStakers[msg.sender][_indexes[i]].amount, fee, block.timestamp);
        }
    }
...
function updateUserReward() public onlyOwner {
       for(uint i = 0; i < userList.length; i++) {
           address userAddress = userList[i];
           if (address(0) == userAddress) {
               continue;
           }
           for (uint j = 0; j < stakers[userAddress].stakedTokens.length; j++) {
               uint256 updateReward = calculateRewards(userAddress,
stakers[userAddress].stakedTokens[j].tokenId);
               stakers[userAddress].unclaimedRewards =
stakers[userAddress].unclaimedRewards.add(updateReward);
               stakers[userAddress].cumulativeAmount =
stakers[userAddress].cumulativeAmount.add(updateReward);
           }
       }
   }
```

[https://github.com/nerdy-star/nerdy_smart_contract/blob/haechi-labs/crystal-staking/contracts/LUXON/myPage/inventory/crystalStaking.sol#L195]

The staker.cumulativeAmount is zero before updateUserReward function first calling.

Therefore, fee calculations always return zero and users do not pay any fee.

Recommendation

if staker.cumulativeAmount is zero, code should not be executed.

Update

Project team decided to keep the fee calculation as zero before receiving the first reward token.

4. stake() method gas optimization issue

ID: Nerdystar-crystalstaking-04	Severity: Tips
Type: Logic Error	Difficulty: N/A
File: LUXON/myPage/inventory/crystalStaking.sol	

Issue

The stake() method has logic to check Token ID existence, but actually this implementation always runs the same whether isNew is true or false. Therefore, the for loop implementation is not need on this logic

```
 bool isNew = true;
if (0 == stakers[msg.sender].totalAmount) {
    userList.push(msg.sender);
    addStakeInfo(msg.sender, _tokenId, _amount);
} else {
    // If token ID you have
    for (uint i = 0; i < stakers[msg.sender].stakedTokens.length; i++) {
        if (_tokenId == stakers[msg.sender].stakedTokens[i].tokenId) {
            addStakeInfo(msg.sender, _tokenId, _amount);
            isNew = false;
        }
    }

    // If new token ID
    if (isNew) {
        addStakeInfo(msg.sender, _tokenId, _amount);
        }
    }
```

[https://github.com/nerdy-star/nerdy_smart_contract/blob/haechi-labs/crystal-staking/contracts/LUXON/myPage/inventory/crystalStaking.sol#L84]

Recommendation

Remove ‘for loop’ and then apply only using addStakeInfo() whether isNew is true or false.

Update

```
function stake(uint256 _tokenId, uint256 _amount) public nonReentrant {
        require(memorialCrystal.balanceOf(msg.sender, _tokenId) > _amount, "you require
more memorial-crystal");
        require(0 != _amount, "invalid amount");

        bool exist = findUserByTokenId(msg.sender, _tokenId);
        if (false == exist) {
            stakingUsers[_tokenId].push(msg.sender);
        }

        if (0 == stakers[msg.sender].totalAmount) {
            userList.push(msg.sender);
        }
        addStakeInfo(msg.sender, _tokenId, _amount);
        tokenTotalAmount = tokenTotalAmount.add(_amount);

        memorialCrystal.safeTransferFrom(
            msg.sender,
            address(this),
            _tokenId,
            _amount,
            ""
        );

        emit Stake(msg.sender, _tokenId, _amount, block.timestamp);
    }
```

기존에 모든 스테이킹 토큰을 모든 내부 요소를 순회하면서 확인하는 굳이 필요하지 않은 로직을 제거하고 Gas optimization 이슈를 해결하였음

The project team removed the unnecessary if statement and for loop statement from the code and completed the gas optimization.

5. disabledLockUp() implementation problem

ID: Nerdystar-crystalstaking-05　　Severity: Tips

Type: Logic Error　　Difficulty: N/A

File: LUXON/myPage/inventory/crystalStaking.sol

Issue

The disabledLockUp() function seems to be a function to release the lock that has been blocked for a certain period of time to prevent unstake. However, the function to release the lock is not implemented due to the internal function.

```
    function disabledLockUp(uint256[] memory _indexes) public {
        // 해제 비용 계산식
        // DSP 누적 정산량 / 언스테이킹 당시 토큰 전체 개수 / 락업 전체 일 수 * 락업 진행 일 수 / 10 *
언스테이킹 토큰 개수;

        for (uint i = 0; i < _indexes.length; i++) {
            UnStaker memory unstaker = unStakers[msg.sender][_indexes[i]];
            Staker memory staker = stakers[msg.sender];

            uint256 remainLockupTime = unstaker.endAt - unstaker.createAt;
            uint256 fee = staker.cumulativeAmount / unstaker.totalTokenAmount / _lockupDay *
(remainLockupTime % DAY) / lockUpFeePercentage * unstaker.amount;

            cumulativeFee = cumulativeFee.add(fee);
            uint256 subCumulativeAmount = staker.cumulativeAmount / unstaker.amount; // 100%
            stakers[msg.sender].cumulativeAmount = staker.cumulativeAmount.sub(subCumulativeAmount);

            rewardsToken.safeTransferFrom(
                msg.sender,
                address(this),
                fee                         // 정산된 fee를 컨트랙트에 전달해 저장함
            );

            emit DisabledLockUp(msg.sender, unStakers[msg.sender][_indexes[i]].tokenId,
unStakers[msg.sender][_indexes[i]].amount, fee, block.timestamp);
        }
    }
```

[https://github.com/nerdy-star/nerdy_smart_contract/blob/haechi-labs/crystal-staking/contracts/LUXON/myPage/inventory/crystalStaking.sol#L195]

Recommendation

disabledLockUp() method only contains implements for updating cumulativeAmount and rewardsToken variables and then effects on it, that’s all. Thus, we need to apply new implements to change user’s actual lockup status variables.

Update

```
function disabledLockUp(uint256[] memory _indexes) public {
            …
            // 현재 시간으로 초기화
            unStakers[msg.sender][i].endAt = block.timestamp;
            …
}
```

In the original version of the code, endAt was not initialized, so it was not possible to unstake the staking token.

EndAt is updated so user can unstake.

6. removeByIndex() gas optimization issue

ID: Nerdystar-crystalstaking-06 Severity: Tips
Type: Logic Error Difficulty: N/A
File: LUXON/myPage/inventory/crystalStaking.sol

Issue

As the amount of values in the list array increases, the cost of gas consumption may become excessive.

```
    function removeByIndex(uint i, address[] storage list) private {
         uint256 size = list.length;
         while (i < size - 1) {
             list[i] = list[i + 1];
             i++;
         }

         list.pop();
     }
```

[https://github.com/nerdy-star/nerdy_smart_contract/blob/haechi-labs/crystal-staking/contracts/LUXON/myPage/inventory/crystalStaking.sol#L406]

Recommendation

You can solve this by replacing the element you want to delete with the last element and popping the last duplicated element

Update

```
     function removeByIndex(uint i, address[] storage list) private {
         uint256 size = list.length;
         list[i] = list[size - 1];
         list.pop();
     }
```

In the past, the gas optimization issue, which occurred because the elements of the array were sorted while traversing one by one, was improved by replacing the element at a specific location with the last element and removing the rearmost element from the array.

7. removeByIndexUnStaker() gas optimization issue

ID: Nerdystar-crystalstaking-07 Severity: Tips
Type: Logic Error Difficulty: N/A
File: LUXON/myPage/inventory/crystalStaking.sol

Issue

As the amount of values in the list array increases, the cost of gas consumption may become excessive.

Recommendation

You can solve this by replacing the element you want to delete with the last element and popping the last duplicated element.

```
   function removeByIndex(uint i, address[] storage list) private {
        uint256 size = list.length;
        list[i] = list[list.length - 1];
        list.pop();
    }
```

[https://github.com/nerdy-star/nerdy_smart_contract/blob/haechi-labs/crystal-staking/contracts/LUXON/myPage/inventory/crystalStaking.sol#L406]

Update

```
function removeByIndexUnStaker(address msgSender, uint256 index) private {
        uint256 size = unStakers[msgSender].length;
        unStakers[msgSender][index] = unStakers[msgSender][size - 1];
        unStakers[msgSender].pop();
    }
```

The project team solved the gas optimization issue in the same way as issue 6 update.

8. unStake() function implements stakingUsers delete logics invalid

ID: Nerdystar-crystalstaking-08	Severity: Tips
Type: Logic Error	Difficulty: N/A
File: LUXON/myPage/inventory/crystalStaking.sol

Issue

In the unStake() function, the logic to delete user information when the user no longer holds the staking status was implemented incorrectly.

In this case, gas wasting may occur because user information is not deleted even though the user is no longer staking.

```
        if (0 == stakers[msg.sender].stakedTokens[tokenIndex].amount) {
            bool exist = findUserByTokenId(msg.sender, _tokenId);
            if (false == exist) {
                removeByValue(msg.sender, stakingUsers[_tokenId]);
            }
        }
```

[https://github.com/nerdy-star/nerdy_smart_contract/blob/haechi-labs/crystal-staking/contracts/LUXON/myPage/inventory/crystalStaking.sol#L124]

In the above logic inside the unStake() function, findUserByTokenId returns true if msg.sender holds the corresponding token in the staking item.

However, in the conditional statement, it compares with false and removeByValue() is performed. if 'exist' variable is false, stakingUsers data is already removed from storage.

Recommendation

If the logic that compares 'exist' variables is changed to a code that checks whether it matches true, it is judged that the code will be able to perform its intended purpose.

Update

```
function unStake(uint256 _tokenId, uint256 _amount) public nonReentrant {
        …
        if (0 == stakers[msg.sender].stakedTokens[tokenIndex].amount) {
            bool exist = findUserByTokenId(msg.sender, _tokenId);
            if (true == exist) {
                removeByValue(msg.sender, stakingUsers[_tokenId]);
            }
        }

        …
}
```

Contrary to the intention of the project team, the specified value could not be removed from the stakingUsers list because the if statement was incorrect.

Currently, the if statement is updated correctly to remove the specified value from the stakingUsers list.

9. claim() function use reversed equal sign

ID: Nerdystar-crystalstaking-09 Severity: Tips

Type: Logic Error Difficulty: N/A

File: LUXON/myPage/inventory/crystalStaking.sol

Issue

The claim() function has a require statement that compares endAt with the current time.

Even though the timestamp is not claimable, the condition to check whether the current timestamp is a claimable timestamp can be bypassed because the equal sign is set in reverse.

```
   function claim(uint256 _index, bool disableLockup) public nonReentrant {
        if (0 < unStakers[msg.sender].length && disableLockup) {
            require(unStakers[msg.sender][_index].endAt > block.timestamp, "It's not time to get a
reward yet.");
            uint256 subCumulativeAmount =
stakers[msg.sender].cumulativeAmount.div(unStakers[msg.sender][_index].amount);
            stakers[msg.sender].cumulativeAmount =
stakers[msg.sender].cumulativeAmount.sub(subCumulativeAmount);
            removeByIndexUnStaker(msg.sender, _index);

            memorialCrystal.safeTransferFrom(
                address(this),
                msg.sender,
                unStakers[msg.sender][_index].tokenId,
                unStakers[msg.sender][_index].amount,
                ""
            );
        }
```

[https://github.com/nerdy-star/nerdy_smart_contract/blob/haechi-labs/crystal-staking/contracts/LUXON/myPage/inventory/crystalStaking.sol#L143]

Recommendation

Make a reverse the direction of the equal sign can solve the problem.

Update

```
function claim(uint256 _index, bool disableLockup) public nonReentrant {
        if (0 < unStakers[msg.sender].length && disableLockup) {
            require(unStakers[msg.sender][_index].endAt < block.timestamp, "It's not
time to get a reward yet.");

            …
    }
```

The sign of the require statement, which had the wrong sign in the past, was corrected in the opposite direction.

10. unStake() method does not check _amount is zero

ID: Nerdystar-crystalstaking-10 Severity: Tips
Type: Logic Error Difficulty: N/A
File: LUXON/myPage/inventory/crystalStaking.sol

Issue

When calling the unStake function with parameter _amount set to 0, the user can add dummy data to unStakers.

The above behavior can cause a situation that removeByIndexUnStaker function inside the claim function can waste gas.

```
    function unStake(uint256 _tokenId, uint256 _amount) public nonReentrant {
        uint256 tokenIndex = getStakedTokenIndex(msg.sender, _tokenId);
        require(noneToken != tokenIndex, "not found token");

        uint256 refundAmount = stakers[msg.sender].stakedTokens[tokenIndex].amount;
        require(refundAmount.sub(_amount) >= 0, "More than the number of holdings");

        stakers[msg.sender].stakedTokens[tokenIndex].amount =
stakers[msg.sender].stakedTokens[tokenIndex].amount.sub(_amount);
        stakers[msg.sender].totalAmount = stakers[msg.sender].totalAmount.sub(_amount);

        if (0 == stakers[msg.sender].stakedTokens[tokenIndex].amount) {
            bool exist = findUserByTokenId(msg.sender, _tokenId);
            if (false == exist) {
                removeByValue(msg.sender, stakingUsers[_tokenId]);
            }
        }

        if (0 == stakers[msg.sender].totalAmount) {
            removeByValue(msg.sender, userList);
        }

        // lockup
        uint256 endAt = block.timestamp + lockupDayMinutes;
        unStakers[msg.sender].push(UnStaker(_tokenId, block.timestamp, endAt, _amount,
stakers[msg.sender].totalAmount));

        emit UnStake(msg.sender, _tokenId, _amount, block.timestamp);
        emit LockUp(msg.sender, _tokenId, _amount, block.timestamp, endAt);
    }
```

[https://github.com/nerdy-star/nerdy_smart_contract/blob/haechi-labs/crystal-staking/contracts/LUXON/myPage/inventory/crystalStaking.sol#L114]

Recommendation

If _amount is 0 in the first line of code in the unStake() function, the code in the unStake function should not be executed.

Update

```
function unStake(uint256 _tokenId, uint256 _amount) public nonReentrant {
        uint256 tokenIndex = getStakedTokenIndex(msg.sender, _tokenId);
        require(noneToken != tokenIndex, "not found token");
        require(0 != _amount, "invalid amount");

        …
 }
```

The problem was solved by adding a require statement to check if the _amount parameter is 0.

DISCLAIMER

This report does not guarantee investment advice, the suitability of the business models, and codes that are secure without bugs. This report shall only be used to discuss known technical issues. Other than the issues described in this report, undiscovered issues may exist such as defects on the main network. In order to write secure smart contracts, correction of discovered problems and sufficient testing thereof are required.

Appendix. A

Severity Level

CRITICAL	Must be addressed as a vulnerability that has the potential to seize or freeze substantial sums of money.
HIGH	Has to be fixed since it has the potential to deny users compensation or momentarily freeze assets.
MEDIUM	Vulnerabilities that could halt services, such as DoS and Out-of-Gas, need to be addressed.
LOW	Issues that do not comply with standards or return incorrect values
TIPS	Tips that makes the code more usable or efficient when modified

Difficulty Level

	Low	**Medium**	**High**
Privilege	anyone	Miner/Block Proposer	Admin/Owner
Capital needed	Small or none	Gas fee or volatile as price change	More than exploited amount
Probability	100%	Depend on environment	Hard as mining difficulty

Vulnerability Category

Arithmetic	▪ Integer under/overflow vulnerability ▪ floating point and rounding accuracy
Access & Privilege Control	▪ Manager functions for emergency handle ▪ Crucial function and data access ▪ Count of calling important task, contract state change, intentional task delay
Denial of Service	▪ Unexpected revert handling ▪ Gas limit excess due to unpredictable implementation
Miner Manipulation	▪ Dependency on the block number or timestamp. ▪ Frontrunning
Reentrancy	▪Proper use of Check-Effect-Interact pattern. ▪Prevention of state change after external call ▪ Error handling and logging.
Low-level Call	▪ Code injection using delegatecall ▪ Inappropriate use of assembly code
Off-standard	▪ Deviate from standards that can be an obstacle of interoperability.
Input Validation	▪ Lack of validation on inputs.
Logic Error/Bug	▪ Unintended execution leads to error.
Documentation	▪Coherency between the documented spec and implementation
Visibility	▪ Variable and function visibility setting
Incorrect Interface	▪ Contract interface is properly implemented on code.

End of Document