auditace.tech

AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

StakingNFTRarity

DATED : 23 November 23'

AUDIT SUMMARY

Project name – StakingNFTRarity

Date: 23 November 2023

Scope of Audit- Audit Ace was consulted to conduct the smart contract audit of the solidity source codes.

Audit Status: Passed

Issues Found

Status	Critical	High	Medium	Low	Suggestion
Open	0	0	0	0	0
Acknowledged	0	0	0	0	0
Resolved	0	0	0	0	0

USED TOOLS

Tools:

1- Manual Review:
A line by line code review has been performed by audit ace team.

2- BSC Test Network: All tests were conducted on the BSC Test network, and each test has a corresponding transaction attached to it. These tests can be found in the "Functional Tests" section of the report.

3- Slither :
The code has undergone static analysis using Slither.

Testnet version:
The tests were performed using the contract deployed on the BSC Testnet, which can be found at the following address:
https://testnet.bscscan.com/address/0xa8c371be06103b5826543a284fd3dc772af6ecc6

Token Information

Name - StakingNFTRarity

Token Adress-
0x3E85cC334b002a864B5532661d6a875644064cF7

Owner - 0xEE6fbEC777B8d04423B7964a984E42fCC22e100a

Deployer -
0xEE6fbEC777B8d04423B7964a984E42fCC22e100a

Network : Binance Smart Chain

Token Type : BEP20

Contract Type: Staking

Testnet version:
The tests were performed using the contract deployed on the BSC Testnet, which can be found at the following address:
https://testnet.bscscan.com/address/0xa8c371be06103b5826543a284fd3dc772af6ecc6

AUDIT METHODOLOGY

The auditing process will follow a routine as special considerations by Auditace:

- Review of the specifications, sources, and instructions provided to Auditace to make sure the contract logic meets the intentions of the client without exposing the user's funds to risk.
- Manual review of the entire codebase by our experts, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Specification comparison is the process of checking whether the code does what the specifications, sources, and instructions provided to Auditace describe.
- Test coverage analysis determines whether the test cases are covering the code and how much code isexercised when we run the test cases.
- Symbolic execution is analysing a program to determine what inputs cause each part of a program to execute.
- Reviewing the codebase to improve maintainability, security, and control based on the established industry and academic practices.

VULNERABILITY CHECKLIST

- [x] Return values of low-level calls
- [x] Private modifier
- [x] Multiple Sends
- [x] Using Suicide
- [x] Gas Limitand Loops
- [x] Address hardcoded
- [x] Exception Disorder
- [x] Using inline assembly
- [x] Divide before multiply
- [x] Missing Zero Address Validation
- [x] Compiler version not fixed
- [x] **Gasless Send**
- [x] Using block.timestamp
- [x] Re-entrancy
- [x] Tautology or contradiction
- [x] Timestamp Dependence
- [x] Revert/require functions
- [x] Use of tx.origin
- [x] Integer overflow/underflow
- [x] Dangerous strict equalities
- [x] Using SHA3
- [x] Using throw

CLASSIFICATION OF RISK

Severity	Description
Critical	These vulnerabilities could be exploited easily and can lead to asset loss, data loss, asset, or data manipulation. They should be fixed right away.
High-Risk	A vulnerability that affects the desired outcome when using a contract, or provides the opportunity to use a contract in an unintended way.
Medium-Risk	A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.
Low-Risk	A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.
Gas Optimization /Suggestion	A vulnerability that has an informational character but is not affecting any of the code.

Findings

Severity	Found
Critical	0
High-Risk	0
Medium-Risk	0
Low-Risk	0
Gas Optimization / Suggestions	0

OWNERSHIP PRIVILEGES FOR STAKING

- Owner can set Daily APR per NFT
- Owner can set collection size
- Owner can enable
- Owner can reset tiers
- Owner can add Tiers
- Owner can transfer

STATIC ANALYSIS

```
INFO:Detectors:
StakingNFTRarity.getCurrentInterestById(uint256) (StakingNFTRarity.sol#947-968) performs a multiplication on the result of a division:
        - tokensPerSeconds = tierToDailyAPR[uint256(tiers[stakeDetail.stakedNFTId - 1])] / ONE_DAY_IN_SECONDS (StakingNFTRarity.sol#960-962)
        - interest = totalSeconds.mul(tokensPerSeconds).sub(stakeDetail.claimedAmount) (StakingNFTRarity.sol#964-966)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#divide-before-multiply
INFO:Detectors:
StakingNFTRarity.countStakingIds(address) (StakingNFTRarity.sol#860-869) uses a dangerous strict equality:
        - idToStakeDetail[stakeIds[i]].status == StakeStatus.Staked (StakingNFTRarity.sol#864)
StakingNFTRarity.getCurrentInterestById(uint256) (StakingNFTRarity.sol#947-968) uses a dangerous strict equality:
        - stakeDetail.status == StakeStatus.Withdrawn (StakingNFTRarity.sol#951)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dangerous-strict-equalities
INFO:Detectors:
Reentrancy in StakingNFTRarity.claim(uint256) (StakingNFTRarity.sol#871-889):
        External calls:
        - tokenContract.transfer(msg.sender,currentInterest) (StakingNFTRarity.sol#884)
        State variables written after the call(s):
        - stakeDetail.claimedAmount = stakeDetail.claimedAmount.add(currentInterest) (StakingNFTRarity.sol#885-887)
        StakingNFTRarity.idToStakeDetail (StakingNFTRarity.sol#768) can be used in cross function reentrancies:
        - StakingNFTRarity.countStakingIds(address) (StakingNFTRarity.sol#860-869)
        - StakingNFTRarity.getCurrentInterestById(uint256) (StakingNFTRarity.sol#947-968)
        - StakingNFTRarity.getStakeDetail(uint256) (StakingNFTRarity.sol#823-827)
        - StakingNFTRarity.idToStakeDetail (StakingNFTRarity.sol#768)
Reentrancy in StakingNFTRarity.stake(uint256[]) (StakingNFTRarity.sol#829-858):
        External calls:
        - nftContract.transferFrom(msg.sender,address(this),_nftId) (StakingNFTRarity.sol#839)
        State variables written after the call(s):
        - addressToIds[msg.sender].push(currentId) (StakingNFTRarity.sol#854)
        StakingNFTRarity.addressToIds (StakingNFTRarity.sol#766) can be used in cross function reentrancies:
        - StakingNFTRarity.countStakingIds(address) (StakingNFTRarity.sol#860-869)
        - StakingNFTRarity.getCurrentTotalInterestOfAddress(address) (StakingNFTRarity.sol#970-980)
        - StakingNFTRarity.getStakingIds(address) (StakingNFTRarity.sol#982-986)
        - idToStakeDetail[currentId] = newStakeDetail (StakingNFTRarity.sol#848)
        StakingNFTRarity.idToStakeDetail (StakingNFTRarity.sol#768) can be used in cross function reentrancies:
        - StakingNFTRarity.countStakingIds(address) (StakingNFTRarity.sol#860-869)
        - StakingNFTRarity.getCurrentInterestById(uint256) (StakingNFTRarity.sol#947-968)
        - StakingNFTRarity.getStakeDetail(uint256) (StakingNFTRarity.sol#823-827)
        - StakingNFTRarity.idToStakeDetail (StakingNFTRarity.sol#768)
        - stakeHolderCount ++ (StakingNFTRarity.sol#851)
        StakingNFTRarity.stakeHolderCount (StakingNFTRarity.sol#725) can be used in cross function reentrancies:
```

```
INFO:Detectors:
StakingNFTRarity.countStakingIds(address)._owner (StakingNFTRarity.sol#860) shadows:
        - Ownable._owner (StakingNFTRarity.sol#56) (state variable)
StakingNFTRarity.getNFTsOfOwner(address).owner (StakingNFTRarity.sol#1004) shadows:
        - Ownable.owner() (StakingNFTRarity.sol#78-80) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
StakingNFTRarity.setCollectionSize(uint256) (StakingNFTRarity.sol#795-797) should emit an event for:
        - collectionSize = _collectionSize (StakingNFTRarity.sol#796)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic
INFO:Detectors:
StakingNFTRarity.stake(uint256[]) (StakingNFTRarity.sol#829-858) has external calls inside a loop: require(bool,string)(nftContract.ownerOf(_nftId) == msg.sender,You are not the owner of this nft) (StakingNFTRarity.sol#835-838)
StakingNFTRarity.stake(uint256[]) (StakingNFTRarity.sol#829-858) has external calls inside a loop: nftContract.transferFrom(msg.sender,address(this),_nftId) (StakingNFTRarity.sol#839)
StakingNFTRarity.unstake(uint256[]) (StakingNFTRarity.sol#913-945) has external calls inside a loop: nftContract.transferFrom(address(this),msg.sender,stakeDetail.stakedNFTId) (StakingNFTRarity.sol#925-929)
StakingNFTRarity.unstake(uint256[]) (StakingNFTRarity.sol#913-945) has external calls inside a loop: tokenContract.transfer(msg.sender,currentInterest) (StakingNFTRarity.sol#933)
StakingNFTRarity.getNFTsOfOwner(address) (StakingNFTRarity.sol#995-1024) has external calls inside a loop: owner = nftContract.ownerOf(i) (StakingNFTRarity.sol#1004-1011)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop
INFO:Detectors:
Reentrancy in StakingNFTRarity.stake(uint256[]) (StakingNFTRarity.sol#829-858):
        External calls:
        - nftContract.transferFrom(msg.sender,address(this),_nftId) (StakingNFTRarity.sol#839)
        State variables written after the call(s):
        - isStakeHolder[msg.sender] = true (StakingNFTRarity.sol#852)
Reentrancy in StakingNFTRarity.unstake(uint256[]) (StakingNFTRarity.sol#913-945):
        External calls:
        - nftContract.transferFrom(address(this),msg.sender,stakeDetail.stakedNFTId) (StakingNFTRarity.sol#925-929)
        - tokenContract.transfer(msg.sender,currentInterest) (StakingNFTRarity.sol#933)
        State variables written after the call(s):
        - isStakeHolder[msg.sender] = false (StakingNFTRarity.sol#942)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
StakingNFTRarity.countStakingIds(address) (StakingNFTRarity.sol#860-869) uses timestamp for comparisons
        Dangerous comparisons:
```

STATIC ANALYSIS

```
INFO:Detectors:
StakingNFTRarity.countStakingIds(address) (StakingNFTRarity.sol#860-869) uses timestamp for comparisons
        Dangerous comparisons:
        - idToStakeDetail[stakeIds[i]].status == StakeStatus.Staked (StakingNFTRarity.sol#864)
StakingNFTRarity.claim(uint256) (StakingNFTRarity.sol#871-889) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(stakeDetail.status == StakeStatus.Staked,Stake is already withdrawn) (StakingNFTRarity.sol#874-877)
        - require(bool,string)(stakeDetail.staker == msg.sender,You are not the staker of this stake) (StakingNFTRarity.sol#878-881)
        - currentInterest > 0 (StakingNFTRarity.sol#882)
StakingNFTRarity.claimAll() (StakingNFTRarity.sol#891-911) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(stakeDetail.staker == msg.sender,You are not the staker of this stake) (StakingNFTRarity.sol#898-901)
        - currentInterest > 0 (StakingNFTRarity.sol#902)
StakingNFTRarity.unstake(uint256[]) (StakingNFTRarity.sol#913-945) uses timestamp for comparisons
        Dangerous comparisons:
        - require(bool,string)(stakeDetail.status == StakeStatus.Staked,Stake is already claimed) (StakingNFTRarity.sol#917-920)
        - require(bool,string)(stakeDetail.staker == msg.sender,You are not the staker of this stake) (StakingNFTRarity.sol#921-924)
        - currentInterest > 0 (StakingNFTRarity.sol#931)
StakingNFTRarity.getCurrentInterestById(uint256) (StakingNFTRarity.sol#947-968) uses timestamp for comparisons
        Dangerous comparisons:
        - stakeDetail.status == StakeStatus.Withdrawn (StakingNFTRarity.sol#951)
        - currentTimestamp <= stakedTimestamp (StakingNFTRarity.sol#956)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.6']
        - ^0.8.0 (StakingNFTRarity.sol#12)
        - ^0.8.0 (StakingNFTRarity.sol#40)
        - ^0.8.0 (StakingNFTRarity.sol#126)
        - ^0.8.0 (StakingNFTRarity.sol#345)
        - ^0.8.0 (StakingNFTRarity.sol#392)
        - ^0.8.0 (StakingNFTRarity.sol#473)
        - ^0.8.0 (StakingNFTRarity.sol#502)
        - ^0.8.0 (StakingNFTRarity.sol#637)
        - ^0.8.6 (StakingNFTRarity.sol#716)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
INFO:Detectors:
StakingNFTRarity.stake(uint256[]) (StakingNFTRarity.sol#829-858) has costly operations inside a loop:
        - stakeHolderCount ++ (StakingNFTRarity.sol#851)
```

```
INFO:Detectors:
Parameter StakingNFTRarity.setDailyAPRPerNFT(uint256,uint256)._tier (StakingNFTRarity.sol#789) is not in mixedCase
Parameter StakingNFTRarity.setDailyAPRPerNFT(uint256,uint256)._dailyAPR (StakingNFTRarity.sol#790) is not in mixedCase
Parameter StakingNFTRarity.setCollectionSize(uint256)._collectionSize (StakingNFTRarity.sol#795) is not in mixedCase
Parameter StakingNFTRarity.setEnabled(bool)._enabled (StakingNFTRarity.sol#799) is not in mixedCase
Parameter StakingNFTRarity.setNftContractAddress(address)._nftAddress (StakingNFTRarity.sol#803) is not in mixedCase
Parameter StakingNFTRarity.addTiers(uint8[])._tiers (StakingNFTRarity.sol#811) is not in mixedCase
Parameter StakingNFTRarity.setTokenContractAddress(address)._newTokenAddress (StakingNFTRarity.sol#818) is not in mixedCase
Parameter StakingNFTRarity.getStakeDetail(uint256)._id (StakingNFTRarity.sol#824) is not in mixedCase
Parameter StakingNFTRarity.stake(uint256[])._nftIds (StakingNFTRarity.sol#829) is not in mixedCase
Parameter StakingNFTRarity.countStakingIds(address)._owner (StakingNFTRarity.sol#860) is not in mixedCase
Parameter StakingNFTRarity.claim(uint256)._stakeId (StakingNFTRarity.sol#871) is not in mixedCase
Parameter StakingNFTRarity.unstake(uint256[])._stakeIds (StakingNFTRarity.sol#913) is not in mixedCase
Parameter StakingNFTRarity.getCurrentInterestById(uint256)._id (StakingNFTRarity.sol#948) is not in mixedCase
Parameter StakingNFTRarity.getCurrentTotalInterestOfAddress(address)._address (StakingNFTRarity.sol#971) is not in mixedCase
Parameter StakingNFTRarity.getStakingIds(address)._address (StakingNFTRarity.sol#983) is not in mixedCase
Parameter StakingNFTRarity.transfer(address,uint256)._recipient (StakingNFTRarity.sol#989) is not in mixedCase
Parameter StakingNFTRarity.transfer(address,uint256)._amount (StakingNFTRarity.sol#990) is not in mixedCase
Parameter StakingNFTRarity.getNFTsOfOwner(address)._addr (StakingNFTRarity.sol#996) is not in mixedCase
Variable StakingNFTRarity.ONE_DAY_IN_SECONDS (StakingNFTRarity.sol#770) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
StakingNFTRarity.ONE_HOUR_IN_SECONDS (StakingNFTRarity.sol#771) is never used in StakingNFTRarity (StakingNFTRarity.sol#717-1026)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-state-variable
INFO:Detectors:
StakingNFTRarity.ONE_DAY_IN_SECONDS (StakingNFTRarity.sol#770) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Slither:StakingNFTRarity.sol analyzed (9 contracts with 93 detectors), 72 result(s) found
```

Result => A static analysis of contract's source code has been performed using slither,

No major issues were found in the output

FUNCTIONAL TESTING

Add Tiers (Passed) -
https://testnet.bscscan.com/tx/0xc2e1bd67f4218e4a00d2bd70fd5b1e3545a2ec0b7cebab0a76f4a012bff21746

Set Collection Size (Passed) -
https://testnet.bscscan.com/tx/0x6c06058552084c381ade7b2bd85fdfeebb9abb0b1df3b6203681286bb808fa73

Set Daily APR Per NFT (Passed) -
https://testnet.bscscan.com/tx/0x1ce68d6ecb05c6d343e917de5e325b220062fdf11aed103db7a012cd201b87d4

Set Enabled (Passed) -
https://testnet.bscscan.com/tx/0x642e9e17d1f554b621faa575be7747bc05732b2a4705473c381f9615e99b4933

Reset Tiers (Passed) -
https://testnet.bscscan.com/tx/0x2a026d61da3cf249c3bd210e7cdba0aa050b9a26b0debb2b6f89508f37d6ec4b

Renounce Ownership (Passed) -
https://testnet.bscscan.com/tx/0xd5e9c00db4ec04cf138da62238195bfd426c634efd1775d9d152b23dfc46c58f

DISCLAIMER

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment. Team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed. The Auditace team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Auditace receive a payment to manipulate those results or change the awarding badge that we will be adding in our website. Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token. The Auditace team disclaims any liability for the resulting losses.

ABOUT AUDITACE

We specializes in providing thorough and reliable audits for Web3 projects. With a team of experienced professionals, we use cutting-edge technology and rigorous methodologies to evaluate the security and integrity of blockchain systems. We are committed to helping our clients ensure the safety and transparency of their digital assets and transactions.

https://auditace.tech/

https://t.me/Audit_Ace

https://twitter.com/auditace_

https://github.com/Audit-Ace