auditace.tech

AuditAce
FROM INCEPTION TO SUCCESS

Smart Contract Audit

FOR

StakingNFTRarity

DATED : 17 November 23'

AUDIT SUMMARY

Project name – StakingNFTRarity

Date: 17 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	2	3	2
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/0xBfd4c168C1F353e3B54D0BbB649111504791A125#code

Token Information

Token Address: -

Name: -

Symbol: -

Decimals: -

Network: Binance smart chain

Contract Type: Staking

Owner: -

Deployer: -

Token Supply: -

Checksum: b391b4a737a5cdfd0edf7de92a1b2764

Testnet version:
The tests were performed using the contract deployed on the Binance smart chain Testnet, which can be found at the following address:
https://testnet.bscscan.com/address/0xBfd4c168C1F353e3B54D0BbB649111504791A125#code

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	2
Low-Risk	3
Gas Optimization / Suggestions	2

POINTS TO NOTE

Owner can set Daily APR per NFT

Owner can set collection size

Owner can set nft contract address

Owner can set token contract address

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

1- Add Tiers (passed):

https://testnet.bscscan.com/tx/0xf718a597bdd1890828676fbcf1955ee95280e61c9fcf948c7f0c11f192776d3e

2- Set Collection Size (passed):

https://testnet.bscscan.com/tx/0xb1c08d53a9896d1f5310cd130424471c87d3f5ddaa12b82fca4e3196e8fcb226

3- Renounce Ownership (passed):

https://testnet.bscscan.com/tx/0x78e14e8dc0f72def9b6d1dc55ab6b0374ffbf0fb81f1adac4b907fdc6542aa03

4- Reset Tiers (passed):

https://testnet.bscscan.com/tx/0x5b8d58bfb98d0ca36cf097e0c0044372e9d911b65fccff21a3817e0f5c518b17

5- Set Token Contract Address (passed):

https://testnet.bscscan.com/tx/0x3bb22730caaedb6ed2584c58d55d3f82b53851f67d1d934fc37d5908498f8deb

6- Set Collection Size (passed):

https://testnet.bscscan.com/tx/0xb1c08d53a9896d1f5310cd130424471c87d3f5ddaa12b82fca4e3196e8fcb226

MANUAL TESTING

Centralization – Owner can change the NFT Contract address.
Severity: Medium
function: SetNFTContractAddress
Status: Open

Overview:
If the owner changes this address after the user has staked the NFT then at the Time of Unstaking, the User will not get the NFT that he staked.

```
function setNftContractAddress(address _nftAddress)
external onlyOwner {
    nftContract = IERC721(_nftAddress);
 }
```

MANUAL TESTING

Centralization – Owner can change the staking token.
Severity: Medium
function: SetTokenContractAddress
Status: Open

Overview:
The owner can change the staking token, in this case, the user will not get the token that they deposited, instead they will get another token set by the owner.

```
function setTokenContractAddress(
address _newTokenAddress
  ) external onlyOwner {
    tokenContract = IERC20(_newTokenAddress);
  }
```

MANUAL TESTING

Severity: Low

subject: Missing Events

Status: Open

Overview:

They serve as a mechanism for emitting and recording data onto the blockchain, making it
transparent and easily accessible.

```
 function setCollectionSize(uint256 _collectionSize) external onlyOwner {
    collectionSize = _collectionSize;
  }

function setEnabled(bool _enabled) external onlyOwner {
    enabled = _enabled;
  }

function setNftContractAddress(address _nftAddress) external onlyOwner {
    nftContract = IERC721(_nftAddress);
  }

function setTokenContractAddress(
address _newTokenAddress
  ) external onlyOwner {
    tokenContract = IERC20(_newTokenAddress);
  }
```

MANUAL TESTING

```
function claim(uint256 _stakeId) external nonReentrant {
 StakeDetail storage stakeDetail =
idTogetCurrentStakeDetail[_stakeId];
uint256 currentInterest = InterestById(_stakeId);
require(
 stakeDetail.status == StakeStatus.Staked,
"Stake is already withdrawn"
 );
require(
 stakeDetail.staker == msg.sender,
"You are not the staker of this stake"
 );
if (currentInterest > 0) {
 tokenContract.transfer(msg.sender, currentInterest);
 stakeDetail.claimedAmount = stakeDetail.claimedAmount.add(
 currentInterest
 );
 }
 }
```

```
 function unstake(uint256[] memory _stakeIds) external
nonReentrant {
for(uint256 i; i < _stakeIds.length; i++) {
uint256 _stakeId = _stakeIds[i];
 StakeDetail storage stakeDetail = idToStakeDetail[_stakeId];
require(
 stakeDetail.status == StakeStatus.Staked,
"Stake is already claimed"
 );
require(
```

MANUAL TESTING

```
 stakeDetail.staker == msg.sender,
"You are not the staker of this stake"
 );

 nftContract.transferFrom(
address(this),
 msg.sender,
 stakeDetail.stakedNFTId
 );
uint256 currentInterest = getCurrentInterestById(_stakeId);
if (currentInterest > 0) {
 tokenContract.transfer(msg.sender, currentInterest);
 stakeDetail.claimedAmount = stakeDetail.claimedAmount.add(
 currentInterest
 );
 }
 stakeDetail.status = StakeStatus.Withdrawn;
uint256 remainingStakeIds = countStakingIds(msg.sender);
if (remainingStakeIds == 0) {
 stakeHolderCount--;
 isStakeHolder[msg.sender] = false;
 }
 }
 }
```

MANUAL TESTING

Centralization – Missing Visibility

Severity: Low

function: Uint256

Status: Open

Overview:

It's simply saying that no visibility was specified, so it's going with the default. This has been related to security issues in contracts.

*uint256 ONE_DAY_IN_SECONDS = 24 * 60 * 60;*
*uint256 constant ONE_HOUR_IN_SECONDS = 60 * 60;*

Suggestion:

You can easily silence the warning by adding the Uint256 public/private.

MANUAL TESTING

Centralization – Missing Zero Address
Severity: Low
function: setTokenContractAddress and CountStakingIDs
Status: Open

Overview:
functions can take a zero address as a parameter (0x00000...). If a function parameter of address type is not properly validated by checking for zero addresses, there could be serious consequences for the contract's functionality.

```
function setTokenContractAddress(
address _newTokenAddress
 ) external onlyOwner {
   tokenContract = IERC20(_newTokenAddress);
 }
```

```
function countStakingIds(address _owner) public view returns (uint256) {
uint256[] storage stakeIds = addressToIds[_owner];
uint256 stakingIds = 0;
for (uint256 i = 0; i < stakeIds.length; i++) {
if (idToStakeDetail[stakeIds[i]].status == StakeStatus.Staked) {
      stakingIds++;
    }
  }
return stakingIds;
 }
```

Suggestion:
It is suggested that the address should not be zero or dead.

MANUAL TESTING

Severity: Optimization
subject: floating Pragma Solidity version
Status: Open

Overview:
It is considered best practice to pick one compiler version and stick with it. With a floating pragma, contracts may accidentally be deployed using an outdated.

pragma solidity ^0.8.6;

Suggestion:
Adding the latest constant version of solidity is recommended, as this prevents the unintentional deployment of a contract with an outdated compiler that contains unresolved bugs.

MANUAL TESTING

Severity: Optimization

subject: Remove unused code.

Status: Open

Overview:

Unused variables are allowed in Solidity, and they do. not pose a direct security issue. It is the best practice. though to avoid them.

```
event RequestClaimed(address indexed staker, uint256 indexed stakeId);

event Claimed(
address indexed staker,
uint256 indexed stakeId,
uint256 amount,
uint256 nftId
  );
```

Suggestion:

It is recommended that not to use unused code.

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