QuillAudits

Audit Report
August, 2022

For

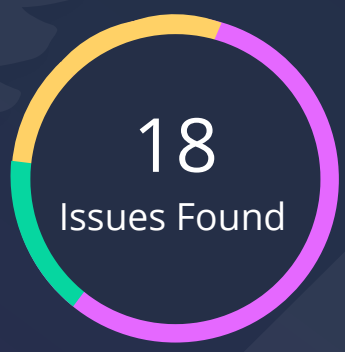
ChainCollection

Table of Content

Table of Content

Executive Summary

Project Name	ChainCollection -NFTController
Overview	The First ZERO-FEES Multi-Chain NFT, NFT Games and Metaverse Marketplace
Timeline	25 April, 2022 to 16 May, 2022
Method	Manual Review, Functional Testing, Automated Testing, etc
Scope of Audit	The scope of this audit was to analyse ChainCollection codebase for quality, security, and correctness. *https://github.com/adilghani/chaincollection-contracts/tree/main* 1st Commit for initial Audit: d095337ddac3043139fc1e275a15400d6c713207 *https://github.com/adilghani/chaincollection-contracts/commit/f4af6c258d93ce36386df213bf6196da8613f7f1*
Fixed in	f4af6c258d93ce36386df213bf6196da8613f7f1
Commit	2nd Commit for revised Audit: e6452ced3d246098810849c4449f4885aa8c490f
Fixed in	*https://github.com/adilghani/chaincollection-contracts/commit/57338213b6cfbaddf47557d596315099581cad94*
Commit	57338213b6cfbaddf47557d596315099581cad94



High Medium

Low Informational

	High	Medium	Low	Informational
Open Issues	0	0	0	0
Acknowledged Issues	0	1	2	3
Partially Resolved Issues	0	0	0	0
Resolved Issues	0	4	4	4

Types of Severities

High

A high severity issue or vulnerability means that your smart contract can be exploited. Issues on this level are critical to the smart contract's performance or functionality, and we recommend these issues be fixed before moving to a live environment.

Medium

The issues marked as medium severity usually arise because of errors and deficiencies in the smart contract code. Issues on this level could potentially bring problems, and they should still be fixed.

Low

Low-level severity issues can cause minor impact and or are just warnings that can remain unfixed for now. It would be better to fix these issues at some point in the future.

Informational

These are severity issues that indicate an improvement request, a general question, a cosmetic or documentation error, or a request for information. There is low-to-no impact.

Types of Issues

Open

Security vulnerabilities identified that must be resolved and are currently unresolved.

Resolved

These are the issues identified in the initial audit and have been successfully fixed.

Acknowledged

Vulnerabilities which have been acknowledged but are yet to be resolved.

Partially Resolved

Considerable efforts have been invested to reduce the risk/impact of the security issue, but are not completely resolved.

Checked Vulnerabilities

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

Techniques and Methods

Throughout the audit of smart contract, care was taken to ensure:

- The overall quality of code.
- Use of best practices.
- Code documentation and comments match logic and expected behaviour.
- Token distribution and calculations are as per the intended behaviour mentioned in the whitepaper.
- Implementation of ERC-20 token standards.
- Efficient use of gas.
- Code is safe from re-entrancy and other vulnerabilities.

The following techniques, methods and tools were used to review all the smart contracts.

Structural Analysis

In this step, we have analysed the design patterns and structure of smart contracts. A thorough check was done to ensure the smart contract is structured in a way that will not result in future problems.

Static Analysis

Static analysis of smart contracts was done to identify contract vulnerabilities. In this step, a series of automated tools are used to test the security of smart contracts.

Code Review / Manual Analysis

Manual analysis or review of code was done to identify new vulnerabilities or verify the vulnerabilities found during the static analysis. Contracts were completely manually analysed, their logic was checked and compared with the one described in the whitepaper. Besides, the results of the automated analysis were manually verified.

Gas Consumption

In this step, we have checked the behaviour of smart contracts in production. Checks were done to know how much gas gets consumed and the possibilities of optimization of code to reduce gas consumption.

Tools and Platforms used for Audit

Remix IDE, Truffle, Truffle Team, Solhint, Mythril, Slither, Solidity statistic analysis.

Manual Testing

High Severity Issues

No issues found

Medium Severity Issues

1. Function should be internal

Line	Function - readyToSellTokenTo()

1858	`function readyToSellTokenTo(`
	`address _tokenAddr,`
	`uint256 _tokenId,`
	`uint256 _price,`
	`// uint256 _royalty,`
	`address _to,`
	`string memory _category,`
	`bool _withEther`
	`) public whenNotPaused notBanned(_tokenAddr, _tokenId)`

Description

This function can be called by anyone as it is public and the address parameter “_to” can be set to any arbitrary address instead of the address of the msg.sender. Making it public would lead to readyToSellTokenTo() being called directly and the address _to parameter can be set arbitrarily.

Remediation

readyToSellTokenTo() function should be made internal and not public as readyToSellTokenTo() is being used internally in readyToSellToken().

Status

Resolved

2. Missing zero check and same address check

Line	Function - constructor
1764	

```
constructor(
    address _nftAddress,
    address _quoteErc20Address,
    address payable _feeAddr,
    uint256 _feePercent,
    address _weth,
    address _usdToken
) public {
    require(_nftAddress != address(0) && _nftAddress != address(this));
    require(_quoteErc20Address != address(0) && _quoteErc20Address != address(this));
    require(_feePercent <= MAX_FEE_PERCENT);
    closedSeaNFT = IclosedSeaNft(_nftAddress);
    quoteErc20 = IERC20(_quoteErc20Address);
    feeAddr = _feeAddr;
    feePercent = _feePercent;
    weth = IWETH(_weth);
    usdToken = IERC20(_usdToken);
    _operators[_msgSender()] = true;
    emit FeeAddressTransferred(address(0), feeAddr);
    emit SetFeePercent(_msgSender(), 0, feePercent);
}
```

Description

There is missing zero check for both “usdToken” and “weth” addresses as well as same address checks. According to the functionality of the contract, these addresses can only be set once (during deployement) and if it’s set to zero or any non-existent address then there could be loss of funds. Moreover, these addresses can also be set as the same which may cause some functions to fail.

Remediation

Consider adding zero address and same address checks to avoid this issue.

Status

Resolved

3. Missing zero check

Line	Function - constructor and transferFeeAddr

1764, 2088	`function transferFeeAddress(address payable _feeAddr) public {`
	`  require(_msgSender() == feeAddr, 'FORBIDDEN');`
	`  feeAddr = _feeAddr;`
	`  emit FeeAddressTransferred(_msgSender(), feeAddr);`
	`}`

Description

Missing Zero check on FEEADDR_ in the constructor and the logic to update it afterwards states that only the feeAddress owner can call the function to change it.
In a scenario where FeeAddress is set to zero address or a Non-existing address will lead to lost of this functionality completely as well as the funds in the form of fee will be lost.

Remediation

Consider adding zero address check in the constructor and instead of using a require check in the transferFeeAddr function, a modifier should be used.

Status

Resolved

4. usdToken Decimal problem

1730

```
1723    IclosedSeaNft public closedSeaNFT;
1724    IERC20 public quoteErc20;
1725    IWETH public weth;
1726    IERC20 public usdToken;    // busd in bsc, usdt in etheruem, etc...
1727    address quoteTokenAndUSDPairAddress;    // weth - quoteToken pair
1728    address usdAndETHPair;    // usdToken - weth pair
1729    address payable public feeAddr;
1730    uint256 public feePercent;
1731    uint256 public seaAmountForExemptFee = 500 ether;
1732    uint256 public constant MAX_FEE_PERCENT = 10000;
```

2360, 2369

```
2356    function getQuoteTokenPrice() public view returns (uint) {
2357        if (quoteTokenAndUSDPairAddress != address(0)) {
2358            uint pairBalInUSD = usdToken.balanceOf(quoteTokenAndUSDPairAddress);
2359            uint pairBalInQuote = quoteErc20.balanceOf(quoteTokenAndUSDPairAddress);
2360            return pairBalInUSD.mul(1e18).div(pairBalInQuote);
2361        }
2362        return 0;
2363    }
2364
2365    function getETHPrice() public view returns (uint) {
2366        if (usdAndETHPair != address(0)) {
2367            uint pairBalInWETH = weth.balanceOf(usdAndETHPair);
2368            uint pairBalInUSD = usdToken.balanceOf(usdAndETHPair);
2369            return pairBalInUSD.mul(1e18).div(pairBalInWETH);
2370        }
2371        return 0;
2372    }
```

Description

The comment on line: 1730 says that usdToken could be any of BUSD or USDT. The getQuoteTokenPrice() and getETHPrice() both use 1e18 in order to multiply the usdToken balance and return value accordingly. But if USDT is used instead of BUSD, it would create a decimal problem, as USDT has 6 decimals. This would mean that it would need to be multiplied by 10^6 instead of 10^18.
Let's say the price is $100. For BUSD, it will be multiplying by 18 and you will get $100. But for USDT, this will give different result because instead of multiplying by 10^6, that price value will be multiplied by 10^18, which will be incorrect.

Remediation

It is advised to write a separate function for getting the price when usdToken is set to address of USDT or USDC stablecoin(which have 6 decimals) and use 10^6 instead on line: 2360 and line: 2369 in this case.

Status

Acknowledged

Comment: The dev team in the meeting said that they would be deploying only on one blockchain right now, and would change the code when they would be deploying on another chain(provided it uses usdToken with 6 decimals).

5. Seller can buy his own NFT

Line	Function - buyToken()
1802	

```
1802    function buyToken(address _tokenAddr, uint256 _tokenId) public payable whenNotPaused notBanned(_tokenAddr, _tokenId) {
1803        require(_msgSender() != address(0) && _msgSender() != address(this), 'Wrong msg sender');
1804        bytes32 key = _getKey(_tokenAddr, _tokenId);
1805        require(!_userBids[_msgSender()].contains(key), 'You must cancel your bid first');
1806
1807        (uint256 price, , ) = _asksMap.get(key);
1808        uint256 feeAmount = 0;
1809        address seller = _tokenSellers[key];
1810        uint256 userSeaTokenBalance = quoteErc20.balanceOf(seller);
1811
1812        // Royalty Data
1813        (address recipient, uint256 royaltyAmount) = getRoyaltyInfo(_tokenId, price);
1814
1815        if (_tokenAskedWithEther[key]) {
1816                require(msg.value >= price, 'pay amount insufficient');
1817                if (userSeaTokenBalance < seaAmountForExemptFee) {
1818                feeAmount = price.mul(feePercent).div(MAX_FEE_PERCENT);
1819                feeAddr.transfer(feeAmount); // fee to feeAddr
1820                }
```

Description

NFT owner can list his own token for sale- then buy his own token for sale. This can result in the seller fooling the royalty design easily by buying his own token and reaching the limit faster so that he can get royalty on his first actual sale itself.

Remediation

It is recommended to disallow this by adding necessary checks in buyToken() function.

Status

Resolved

Low Severity Issues

6. Non-transfer of NFT after listing possible

Line	Function - readyToSellTokenTo()
1871	

```
1873    function readyToSellTokenTo(
1874        address _tokenAddr,
1875        uint256 _tokenId,
1876        uint256 _price,
1877        address _to,
1878        string memory _category,
1879        bool _withEther
1880    ) internal whenNotPaused notBanned(_tokenAddr, _tokenId) {
1881        require(_msgSender() == IERC721(_tokenAddr).ownerOf(_tokenId), 'Only Token Owner can sell token');
1882        bytes32 key = _getKey(_tokenAddr, _tokenId);
1883        require(_price != 0, 'Price must be granter than zero');
1884        _asksMap.set(key, _price, _tokenId, _tokenAddr);
1885        _tokenSellers[key] = _to;
1886        _tokenCategories[key] = bytes32(bytes(_category));
1887        _tokenAskedWithEther[key] = _withEther;
1888        _userSellingTokens[_to].add(key);
1889        _categorySellingTokens[bytes32(bytes(_category))].add(key);
1890        emit Ask(_to, _tokenId, _price);
1891    }
```

Description

When a seller lists an NFT for sale, the NFT remains with the owner himself(instead of this getting transferred to the NFT Controller smart contract). According to the dev team, the seller of NFT will approve the smart contract.
But it is entirely possible that the seller removes that approval after listing his NFT on sale. This would result in buyToken function always failing as transfer of NFT to new owner/buyer will never be possible.
This could also result in seller exploiting this by listing his NFT for your marketplace as well as some other marketplace(even when he is no longer the owner)

Remediation

It is advised to follow a better approach which will be to transfer the NFT to the smart contract when the readyToSellToken() is called.

Status

Resolved

7. Improper Implementation

Line	Function - delBidByTokenInfoAndIndex

2171	`uint256 len = _tokenBids[key].length;`
	`for (uint256 i = _index; i < len - 1; i++) {`
	`  _tokenBids[key][i] = _tokenBids[key][i + 1];`
	`}`
	`_tokenBids[key].pop();`
	`}`

Description

The implementation of this logic results is that the index should be less than length. Also, the cases of non-existing indexes should be handled.

Remediation

If the index doesn't exist, the function should not return any number and there should be checks to handle the non-existing indexes.

Status

Resolved

8. Insufficient Tests Provided

Description

There was insufficient test coverage of the codebase provided to us When such a critical project does not provide all the details about what to expect from and functions and logic, it is not recommended from security perspective.

Remediation

It is advised that the team cover at least 80 percent of the test cases.

Status

Acknowledged

9. Issue with fee deduction

Line	Function - buyToken
1811, 1818	

```
if (_tokenAskedWithEther[key]) {
    require(msg.value >= price, 'pay amount insufficient');
        if (feeAmount != 0 && userSeaTokenBalance < seaAmountForExemptFee) {
        feeAddr.transfer(feeAmount); // 2% to feeAddr
        }
        // Royalty Implementation
        if(recipient != closedSeaNFT.ownerOf(_tokenId)){
            // distribute royalty to recipient from 98%
            payable(recipient).transfer(royaltyAmount); // transfer Royalty to
            payable(seller).transfer(price.sub(feeAmount.add(royaltyAmount)));
        }
```

Description

If an user is using Ether and their sea token balance is greater than the sea amount for exempt fee but the feeAmount is not equal to zero then the condition on line 1811 will return false but there is a possibility that the recipient is not the owner of the tokenId, and in that case, fee amount will still be deducted while transferring the token.
Hence, the logic of the contract which states that if an user has a certain amount of SeaTokenBalance will not have to pay the fee will fail.

Remediation

Consider checking the sea token balance again in the condition on the line “#1815”.

Status

Resolved

10. Return values not checked

Description

Slither static analysis showed that the return value of the following operations are not checked For example, remove() from EnumerableSet returns true if the value was removed from the set, that is if it was present. Failing to check the return value can lead to incorrect and false assumptions

ClosedSeaNFTController.buyToken(address,uint256)
(contracts/chaincollection.sol#1802-1857) ignores return value by _asksMap.remove(key) (contracts/chaincollection.sol#1850)

ClosedSeaNFTController.buyToken(address,uint256)
(contracts/chaincollection.sol#1802-1857) ignores return value by _userSellingTokens[_tokenSellers[key]].remove(key) (contracts/chaincollection.sol#1851)

ClosedSeaNFTController.buyToken(address,uint256)
(contracts/chaincollection.sol#1802-1857) ignores return value by _categorySellingTokens[_tokenCategories[key]].remove(key) (contracts/chaincollection.sol#1852)

ClosedSeaNFTController.setCurrentPrice(address,uint256,uint256)
(contracts/chaincollection.sol#1860-1866) ignores return value by _asksMap.set(key,_price,_tokenId,_tokenAddr) (contracts/chaincollection.sol#1864)

ClosedSeaNFTController.readyToSellTokenTo(address,uint256,uint256,address,string,bool)
(contracts/chaincollection.sol#1873-1891) ignores return value by _asksMap.set(key,_price,_tokenId,_tokenAddr) (contracts/chaincollection.sol#1884)

ClosedSeaNFTController.readyToSellTokenTo(address,uint256,uint256,address,string,bool)
(contracts/chaincollection.sol#1873-1891) ignores return value by _userSellingTokens[_to].add(key) (contracts/chaincollection.sol#1888)

ClosedSeaNFTController.readyToSellTokenTo(address,uint256,uint256,address,string,bool)
(contracts/chaincollection.sol#1873-1891) ignores return value by _categorySellingTokens[bytes32(bytes(_category))].add(key) (contracts/chaincollection.sol#1889)

ClosedSeaNFTController.cancelSellToken(address,uint256)
(contracts/chaincollection.sol#1893-1903) ignores return value by _asksMap.remove(key) (contracts/chaincollection.sol#1896)

ClosedSeaNFTController.cancelSellToken(address,uint256)
(contracts/chaincollection.sol#1893-1903) ignores return value by _userSellingTokens[_tokenSellers[key]].remove(key) (contracts/chaincollection.sol#1897)

ClosedSeaNFTController.cancelSellToken(address,uint256)
(contracts/chaincollection.sol#1893-1903) ignores return value by _categorySellingTokens[_tokenCategories[key]].remove(key) (contracts/chaincollection.sol#1898)

ClosedSeaNFTController.bidToken(address,uint256,uint256,bool)
(contracts/chaincollection.sol#2116-2147) ignores return value by _userBids[_to].set(key,_price,_tokenId,_tokenAddr) (contracts/chaincollection.sol#2144)

ClosedSeaNFTController.updateBidPrice(address,uint256,uint256)
(contracts/chaincollection.sol#2149-2175) ignores return value by _userBids[_to].set(key,_price,_tokenId,_tokenAddr) (contracts/chaincollection.sol#2172)

ClosedSeaNFTController.delBidByTokenInfoAndIndex(address,uint256,uint256)
(contracts/chaincollection.sol#2198-2207) ignores return value by _userBids[_tokenBids[key][_index].bidder].remove(key) (contracts/chaincollection.sol#2200)

ClosedSeaNFTController.sellTokenTo(address,uint256,address)
(contracts/chaincollection.sol#2261-2318) ignores return value by _asksMap.remove(key) (contracts/chaincollection.sol#2310)

ClosedSeaNFTController.sellTokenTo(address,uint256,address)
(contracts/chaincollection.sol#2261-2318) ignores return value by _userSellingTokens[_tokenSellers[key]].remove(key) (contracts/chaincollection.sol#2311)

ClosedSeaNFTController.sellTokenTo(address,uint256,address)
(contracts/chaincollection.sol#2261-2318) ignores return value by _categorySellingTokens[_tokenCategories[key]].remove(key) (contracts/chaincollection.sol#2312)

Reference

https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return

Remediation

It is advised to add the required boolean require checks for the same and review the business logic.

Status

Acknowledged

11. Violation of checks-effects-interactions pattern

Description

There is violation of checks-effects-interactions pattern throughout the contract which may give rise to potential attacks like Reentrancy attacks.

Remediation

It is advised to follow this pattern to avoid the danger of any reentrancy
attack. Also it is advised that the external call on line: 2270 of safeTransferFrom be done at the end of the function sellTokenTo() as it transfers the call of execution of the function to an external contract function that could be malicious.

Status

Resolved

Informational Issues

12. Unlocked pragma (pragma solidity ^0.8.0)

Description

Contracts should be deployed with the same compiler version and flags that they have been tested with thoroughly. Locking the pragma helps to ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.
In case of this contract, certain functionalities have been used that only came into existence after the version 0.8.4 and we have been informed that the contract was tested on compiler version 0.8.7, but in the contract itself 0.8.0's unlocked version is used.

Remediation

Here all the in-scope contracts have an unlocked pragma, it is recommended to lock the same (0.8.7). Moreover, we strongly suggest not to use experimental Solidity features (e.g., pragma experimental ABIEncoderV2) or third-party unaudited libraries. If necessary, refactor the current code base to only use stable features.

Status

Resolved

13. Multiple Pragmas used (pragma solidity ^0.6.0 <0.8.0)

Description

Imported contracts (by OpenZeppelin) are using solidity version till 0.8.0 and the main contract was tested with the functionalities of version 0.8.7 and it is not recommended to use these versions while deployment with locked pragmas.

Remediation

Use latest imports by OpenZeppelin

Status

Resolved

14. General Recommendation

Description

In the light of recent events, we conclude in our audit that certain libraries such as EnumerableMap and EnumerableSet are known to consume a lot of gas. We suggest to use them carefully.

Status

Acknowledged

15. Lack of following Solidity Naming Guidelines

Description

readyToSellTokenTo() and safeTransferQuoteToken() are internal functions. It's name should start from _ as per solidity naming guidelines such as _readyToSellTokenTo() and _safeTransferQuoteToken() for better code readability.

Also the static analysis from Slither showed incorrect following of Solidity naming guidelines for function parameters as shown in the screenshot below.

```
Parameter ClosedSeaNFTController.getRoyaltyInfo(uint256,uint256)._tokenId (contracts/chaincollection.sol#1793) is not in mixedCase
Parameter ClosedSeaNFTController.getRoyaltyInfo(uint256,uint256)._price (contracts/chaincollection.sol#1793) is not in mixedCase
Parameter ClosedSeaNFTController.buyToken(address,uint256)._tokenAddr (contracts/chaincollection.sol#1802) is not in mixedCase
Parameter ClosedSeaNFTController.buyToken(address,uint256)._tokenId (contracts/chaincollection.sol#1802) is not in mixedCase
Parameter ClosedSeaNFTController.setCurrentPrice(address,uint256,uint256)._tokenAddr (contracts/chaincollection.sol#1860) is not in mixedCase
Parameter ClosedSeaNFTController.setCurrentPrice(address,uint256,uint256)._tokenId (contracts/chaincollection.sol#1860) is not in mixedCase
Parameter ClosedSeaNFTController.setCurrentPrice(address,uint256,uint256)._price (contracts/chaincollection.sol#1860) is not in mixedCase
Parameter ClosedSeaNFTController.readyToSellToken(address,uint256,uint256,string,bool)._tokenAddr (contracts/chaincollection.sol#1869) is not in mixedCase
Parameter ClosedSeaNFTController.readyToSellToken(address,uint256,uint256,string,bool)._tokenId (contracts/chaincollection.sol#1869) is not in mixedCase
Parameter ClosedSeaNFTController.readyToSellToken(address,uint256,uint256,string,bool)._price (contracts/chaincollection.sol#1869) is not in mixedCase
Parameter ClosedSeaNFTController.readyToSellToken(address,uint256,uint256,string,bool)._category (contracts/chaincollection.sol#1869) is not in mixedCase
Parameter ClosedSeaNFTController.readyToSellToken(address,uint256,uint256,string,bool)._withEther (contracts/chaincollection.sol#1869) is not in mixedCase
Parameter ClosedSeaNFTController.readyToSellTokenTo(address,uint256,uint256,address,string,bool)._tokenAddr (contracts/chaincollection.sol#1874) is not in mixedCase
Parameter ClosedSeaNFTController.readyToSellTokenTo(address,uint256,uint256,address,string,bool)._tokenId (contracts/chaincollection.sol#1875) is not in mixedCase
Parameter ClosedSeaNFTController.readyToSellTokenTo(address,uint256,uint256,address,string,bool)._price (contracts/chaincollection.sol#1876) is not in mixedCase
Parameter ClosedSeaNFTController.readyToSellTokenTo(address,uint256,uint256,address,string,bool)._to (contracts/chaincollection.sol#1877) is not in mixedCase
Parameter ClosedSeaNFTController.readyToSellTokenTo(address,uint256,uint256,address,string,bool)._category (contracts/chaincollection.sol#1878) is not in mixedCase
Parameter ClosedSeaNFTController.readyToSellTokenTo(address,uint256,uint256,address,string,bool)._withEther (contracts/chaincollection.sol#1879) is not in mixedCase
Parameter ClosedSeaNFTController.cancelSellToken(address,uint256)._tokenAddr (contracts/chaincollection.sol#1893) is not in mixedCase
Parameter ClosedSeaNFTController.cancelSellToken(address,uint256)._tokenId (contracts/chaincollection.sol#1893) is not in mixedCase
Parameter ClosedSeaNFTController.getAskByCategoryLength(string)._category (contracts/chaincollection.sol#1909) is not in mixedCase
Parameter ClosedSeaNFTController.getAsksByCategoryAndPageDesc(string,uint256,uint256)._category (contracts/chaincollection.sol#2029) is not in mixedCase
Parameter ClosedSeaNFTController.getAsksByCategoryAndPageDesc(string,uint256,uint256)._page (contracts/chaincollection.sol#2029) is not in mixedCase
Parameter ClosedSeaNFTController.getAsksByCategoryAndPageDesc(string,uint256,uint256)._size (contracts/chaincollection.sol#2029) is not in mixedCase
Parameter ClosedSeaNFTController.getAsksByCategoryDesc(string)._category (contracts/chaincollection.sol#2066) is not in mixedCase
Parameter ClosedSeaNFTController.getCategoryOf(address,uint256)._tokenAddr (contracts/chaincollection.sol#2084) is not in mixedCase
Parameter ClosedSeaNFTController.getCategoryOf(address,uint256)._tokenId (contracts/chaincollection.sol#2084) is not in mixedCase
Parameter ClosedSeaNFTController.transferFeeAddress(address)._feeAddr (contracts/chaincollection.sol#2102) is not in mixedCase
Parameter ClosedSeaNFTController.setFeePercent(uint256)._feePercent (contracts/chaincollection.sol#2109) is not in mixedCase
Parameter ClosedSeaNFTController.bidToken(address,uint256,uint256,bool)._tokenAddr (contracts/chaincollection.sol#2116) is not in mixedCase
Parameter ClosedSeaNFTController.bidToken(address,uint256,uint256,bool)._tokenId (contracts/chaincollection.sol#2116) is not in mixedCase
Parameter ClosedSeaNFTController.bidToken(address,uint256,uint256,bool)._price (contracts/chaincollection.sol#2116) is not in mixedCase
Parameter ClosedSeaNFTController.bidToken(address,uint256,uint256,bool)._withEther (contracts/chaincollection.sol#2116) is not in mixedCase
Parameter ClosedSeaNFTController.updateBidPrice(address,uint256,uint256)._tokenAddr (contracts/chaincollection.sol#2149) is not in mixedCase
Parameter ClosedSeaNFTController.updateBidPrice(address,uint256,uint256)._tokenId (contracts/chaincollection.sol#2149) is not in mixedCase
Parameter ClosedSeaNFTController.updateBidPrice(address,uint256,uint256)._price (contracts/chaincollection.sol#2149) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
```

Remediation

It is advised to follow the solidity naming conventions thoroughly.

Reference

Status

Acknowledged

16. Non-usage of inherited contract

Line	Function
1698	

```
1698  contract ClosedSeaNFTController is ERC721Holder, Ownable, Pausable {
1699      using SafeMath for uint256;
1700      using SafeERC20 for IERC20;
1701      using Address for address;
1702      using EnumerableMap for EnumerableMap.Bytes32ToUintMap;
1703      using EnumerableSet for EnumerableSet.UintSet;
1704      using EnumerableSet for EnumerableSet.Bytes32Set;
1705
1706      struct AskEntry {
1707          address tokenAddr;
1708          uint256 tokenId;
1709          uint256 price;
1710          bool withEther;
1711      }
1712
```

Description

Contract inherits ERC721Holder contract but is never holding any NFTs at any point of time.

Remediation

It is advised to remove redundant contract getting inherited if not in use.

Status

Resolved

Comment: New logic of the contract utilizes this to store ERC.

17. Public function could be declared external

Description

Public functions that are never called by the contract should be declared external to save gas

```
buyToken(address,uint256) should be declared external:
        - ClosedSeaNFTController.buyToken(address,uint256) (contracts/chaincollection.sol#1802-1857)
setCurrentPrice(address,uint256,uint256) should be declared external:
        - ClosedSeaNFTController.setCurrentPrice(address,uint256,uint256) (contracts/chaincollection.sol#1860-1866)
readyToSellToken(address,uint256,uint256,string,bool) should be declared external:
        - ClosedSeaNFTController.readyToSellToken(address,uint256,uint256,string,bool) (contracts/chaincollection.sol#1869-1871)
cancelSellToken(address,uint256) should be declared external:
        - ClosedSeaNFTController.cancelSellToken(address,uint256) (contracts/chaincollection.sol#1893-1903)
getAskLength() should be declared external:
        - ClosedSeaNFTController.getAskLength() (contracts/chaincollection.sol#1905-1907)
getAskByCategoryLength(string) should be declared external:
        - ClosedSeaNFTController.getAskByCategoryLength(string) (contracts/chaincollection.sol#1909-1911)
getAsks() should be declared external:
        - ClosedSeaNFTController.getAsks() (contracts/chaincollection.sol#1913-1923)
getAsksDesc() should be declared external:
        - ClosedSeaNFTController.getAsksDesc() (contracts/chaincollection.sol#1925-1942)
getAsksByPage(uint256,uint256) should be declared external:
        - ClosedSeaNFTController.getAsksByPage(uint256,uint256) (contracts/chaincollection.sol#1944-1961)
getAsksByPageDesc(uint256,uint256) should be declared external:
        - ClosedSeaNFTController.getAsksByPageDesc(uint256,uint256) (contracts/chaincollection.sol#1963-1996)
getAsksByUser(address) should be declared external:
        - ClosedSeaNFTController.getAsksByUser(address) (contracts/chaincollection.sol#1998-2008)
getAsksByUserDesc(address) should be declared external:
        - ClosedSeaNFTController.getAsksByUserDesc(address) (contracts/chaincollection.sol#2010-2027)
getAsksByCategoryAndPageDesc(string,uint256,uint256) should be declared external:
        - ClosedSeaNFTController.getAsksByCategoryAndPageDesc(string,uint256,uint256) (contracts/chaincollection.sol#2029-2064)
getAsksByCategoryDesc(string) should be declared external:
        - ClosedSeaNFTController.getAsksByCategoryDesc(string) (contracts/chaincollection.sol#2066-2082)
getAsksByCategoryDesc(string) should be declared external:
        - ClosedSeaNFTController.getAsksByCategoryDesc(string) (contracts/chaincollection.sol#2066-2082)
getCategoryOf(address,uint256) should be declared external:
        - ClosedSeaNFTController.getCategoryOf(address,uint256) (contracts/chaincollection.sol#2084-2092)

transferFeeAddress(address) should be declared external:
        - ClosedSeaNFTController.transferFeeAddress(address) (contracts/chaincollection.sol#2102-2107)
setFeePercent(uint256) should be declared external:
        - ClosedSeaNFTController.setFeePercent(uint256) (contracts/chaincollection.sol#2109-2114)
bidToken(address,uint256,uint256,bool) should be declared external:
        - ClosedSeaNFTController.bidToken(address,uint256,uint256,bool) (contracts/chaincollection.sol#2116-2147)
updateBidPrice(address,uint256,uint256) should be declared external:
        - ClosedSeaNFTController.updateBidPrice(address,uint256,uint256) (contracts/chaincollection.sol#2149-2175)
sellTokenTo(address,uint256,address) should be declared external:
        - ClosedSeaNFTController.sellTokenTo(address,uint256,address) (contracts/chaincollection.sol#2261-2318)
cancelBidToken(address,uint256) should be declared external:
        - ClosedSeaNFTController.cancelBidToken(address,uint256) (contracts/chaincollection.sol#2320-2334)
getBidsLength(address,uint256) should be declared external:
        - ClosedSeaNFTController.getBidsLength(address,uint256) (contracts/chaincollection.sol#2336-2339)
getBids(address,uint256) should be declared external:
        - ClosedSeaNFTController.getBids(address,uint256) (contracts/chaincollection.sol#2341-2344)
getUserBids(address) should be declared external:
        - ClosedSeaNFTController.getUserBids(address) (contracts/chaincollection.sol#2346-2354)
getQuoteTokenPrice() should be declared external:
        - ClosedSeaNFTController.getQuoteTokenPrice() (contracts/chaincollection.sol#2356-2363)
getETHPrice() should be declared external:
        - ClosedSeaNFTController.getETHPrice() (contracts/chaincollection.sol#2365-2372)
checkRoyalty(address) should be declared external:
        - ClosedSeaNFTController.checkRoyalty(address) (contracts/chaincollection.sol#2432-2434)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
```

Remediation

It is advised to declare the functions external as indicated in the screenshot above.

Status

Resolved

18. Important Note

The dev team was asked to deflatten or separate the contract due to inefficiency of testing the contracts in Remix as it resulted in crashing of Remix on doing transactions. But the dev team was unable to provide the required separated contracts resulting in inadequate functionality testing combined with time constraint for the audit. It is advised to take a note of this and it is recommended that the team carries out additional functionality testing for the same before the final launch.

Status

Acknowledged

Functional Testing

- ✓ should be able to buy and sell token
- ✓ Should be able to bid, update the bidding price and cancel bid
- ✓ Should be able to cancel sale of token
- ✓ Should be able to set fee address and rate
- ✓ Should be able to set price for token
- ✓ Should be able to ban and release token
- ✓ Should be able to pause and unpause functionalities
- ✓ Should revert if WETH and USD token are either same or the zero address
- ✓ Should revert if fee address is a zero address
- ✓ Should revert if seller is not owner
- ✓ Should revert if owner tries to bid
- ✓ Should revert if the Token is not in sell book
- ✓ Should revert if the bidder tries to buy without cancelling the bid

Automated Tests

No major issues were found. Some false positive errors were reported by the tools. Other issues reported have been added above in the report. All the other issues have been categorized above according to their level of severity.

Closing Summary

In this report, we have considered the security of the ChainCollection. We performed our audit according to the procedure described above.

Some issues of Medium, Low and informational severity were found, Some suggestions and best practices are also provided in order to improve the code quality and security posture. In the end, ChainCollection Team Resolved all issues.

Disclaimer

QuillAudits smart contract audit is not a security warranty, investment advice, or an endorsement of the ChainCollection Platform. This audit does not provide a security or correctness guarantee of the audited smart contracts.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them. Securing smart contracts is a multistep process. One audit cannot be considered enough. We recommend that the ChainCollection team put in place a bug bounty program to encourage further analysis of the smart contract by other third parties.

About QuillAudits

QuillAudits is a secure smart contracts audit platform designed by QuillHash Technologies. We are a team of dedicated blockchain security experts and smart contract auditors determined to ensure that Smart Contract-based Web3 projects can avail the latest and best security solutions to operate in a trustworthy and risk-free ecosystem.

500+
Audits Completed

$15B
Secured

500K
Lines of Code Audited

Follow Our Journey

Audit Report
August, 2022

For

ChainCollection

QuillAudits

Canada, India, Singapore, United Kingdom

audits.quillhash.com

audits@quillhash.com