#Hashlock.

Mad Cartels (NFT)

SMART CONTRACT

Security Audit

Performed on Contracts:

madCartelNFT.sol
MD5 : 39e5d941c26268be6ea90c6c42e2d89f
OperatorFilterer.sol
MD5: e427016321faf240b195ee97bc03b32b
IOperatorFilterRegistry.sol
MD5: 7231704f1485ae005918633b96c954c7
DefaultOperatorFilterer.sol
MD5: 3db46a8b8c0ee95b7ce3ac9eef7f1caf

Platform

ERC-721

hashlock.com.au

MARCH 2023

Table of Contents

CAUTION

THIS DOCUMENT IS A SECURITY AUDIT REPORT AND MAY CONTAIN CONFIDENTIAL INFORMATION. THIS INCLUDES IDENTIFIED VULNERABILITIES AND MALICIOUS CODE WHICH COULD BE USED TO COMPROMISE THE PROJECT. THIS DOCUMENT SHOULD ONLY BE FOR INTERNAL USE UNTIL ISSUES ARE RESOLVED. ONCE VULNERABILITIES ARE REMEDIATED, THIS REPORT CAN BE MADE PUBLIC. THE CONTENT OF THIS REPORT IS OWNED BY HASHLOCK PTY LTD FOR USE OF THE CLIENT.

Executive Summary

The Mad Cartels team partnered with Hashlock to conduct a security audit of their ERC721 smart contracts prior to their project's launch. Hashlock manually and proactively reviewed the code in order to ensure the project's team and community that it is ready for mainnet deployment.

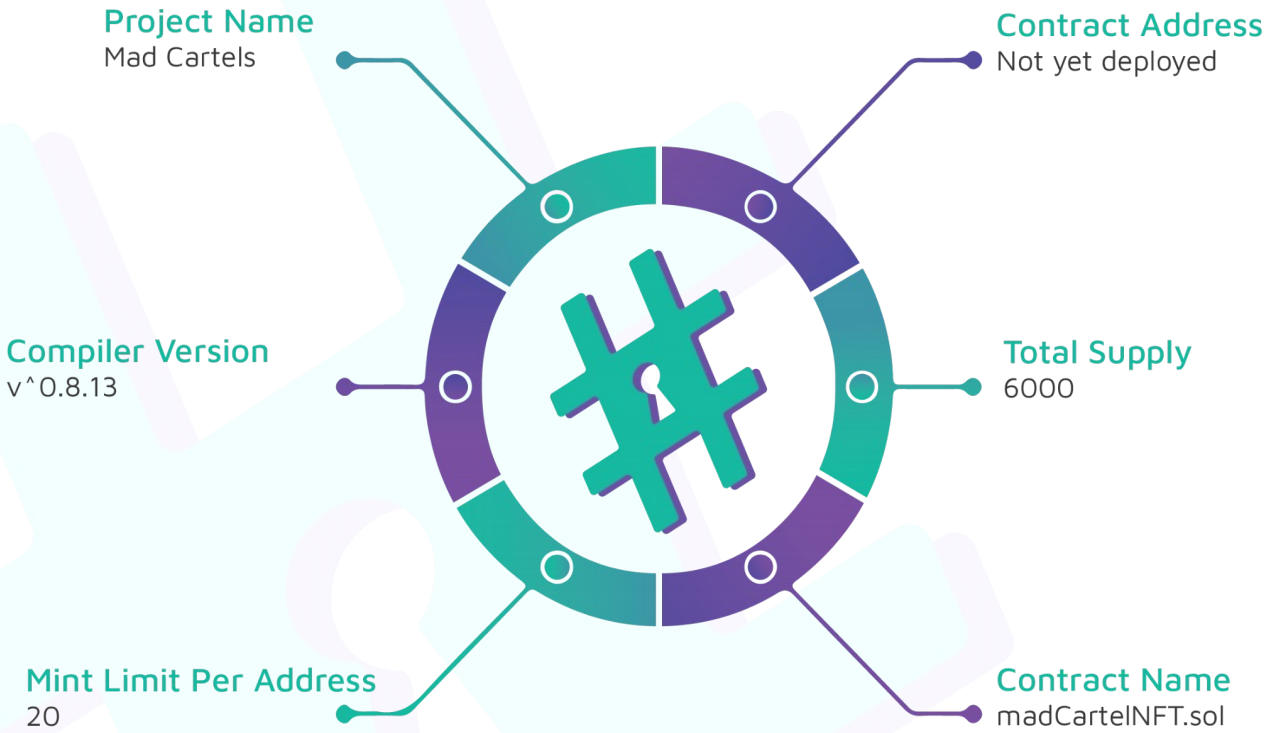
Project Context

Mad Cartels is an upcoming NFT project aiming to build an engaged community via an exciting roadmap. The MadCartels team engaged Hashlock to ensure their NFT contract is ready for Launch.

Project Name: Mad Cartels
Contract Address: N/A
Compiler Version: ^0.8.19
Max Supply: 6000

Logo:

Visualised Context:



Project Artwork:

Roadmap

PHASE 1 - Mint the revolutionary 3D Mafia character NFT

PHASE 2 - Mad Cartels send the first gift to all its NFT holders. Each holder can claim one free safe house key as an NFT that can be traded on the secondary market.

PHASE 3 - Mad Cartels take photos from Mafia characters & each holder can claim one free 2D image as an NFT.

PHASE 4 - Discover the highly anticipated limited edition of 400 3D NFT auctions of Genesis characters held by Mad Cartels, exclusive to NFT holders only.

PHASE 5 - Mad Cartels offer one free special 2D photo & 2 special ranches as NFTs for every Genesis holder.

PHASE 6 - Mad Cartels provide 2 free Capos as NFTs for every Genesis holder.

PHASE 7 - Mad Cartels gift a new car as an NFT to its NFT holders.

PHASE 8 - Mad Cartels offer one luxurious Limousine car as an NFT for every Genesis holder.

PHASE 9 - Now that the black market is open and running, Mad Cartels send a free weapon as an NFT to each holder.

PHASE 10 - Time for Genesis holders to be awarded a free limited edition Golden weapon as an NFT.

PHASE 11 - Get ready for the ultimate drop, Metaverse lands re-dropped to all holders as NFTs to embark on the Mad Cartels long-term mission!

PHASE 12 - Introducing a New Ground-breaking Mafia Deal.

PHASE 13 - Unravel the unprecedented benefits of the Mafia Cartels Token.

Mad Cartels' Project Leadership

Title	Name	Photo	Links
Co-Founder	Ali Bagheri		Twitter
Co-Founder	Samin Javadian		Twitter

Audit scope

We at Hashlock audited the solidity code within the Mad Cartels Project, the scope of works included a comprehensive review of the smart contracts listed below. We tested the smart contracts to check for their security and efficiency. These tests were undertaken through both manual and software assisted analysis.

Description	**Project Review and Security Analysis Report for Mad Cartels Protocol Smart Contracts and other factors.**
Platform	**Ethereum / Solidity**
Audit Date	**March 10th, 2023**
Contract 1	madCartelNFT.sol
Contract 1 MD5 Hash	39e5d941c26268be6ea90c6c42e2d89f
Contract 2	OperatorFilterer.sol
Contract 2 MD5 Hash	e427016321faf240b195ee97bc03b32b
Contract 3	IOperatorFilterRegistry.sol
Contract 3 MD5 Hash	7231704f1485ae005918633b96c954c7
Contract 4	DefaultOperatorFilterer.sol
Contract 4 MD5 Hash	3db46a8b8c0ee95b7ce3ac9eef7f1caf

Security Rating

After our audit and analysis, we found the smart contracts to be **"Secure"**. The contracts all follow simple logic, with correct and detailed ordering. They use a series of interfaces, and the protocol uses a list of Open Zeppelin contracts. We have identified two high and multiple medium and low severity vulnerabilities that will need to be addressed before mainnet launch.

The 'Hashlocked' rating is reserved for fully decentralised, highly secure projects with little owner influence.

All issues uncovered during automated and manual analysis were meticulously reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in the Function list section and all identified issues can be found in the Audit overview section.

All vulnerabilities initially identified have now been acknowledged or resolved.

Initially Hashlock found:

0 Critical severity vulnerabilities

2 High severity vulnerabilities

2 Medium severity vulnerabilities

10 Low severity vulnerabilities

Caution: *Hashlock's audits do not guarantee a project's success or ethics, and are not liable or responsible for security. Always conduct independent research about any project before interacting.*

Standardised Checks

Main Category	Subcategory	Result
General Code Checks	Solidity/compiler version stated	Passed
	Consistent pragma version across each contract	Reviewed
	Outdated Solidity Version	Reviewed
	Overflow/underflow	Passed
	Correct checks, effects, interaction order	Passed
	Lack of check on input parameters	Passed
	Function input parameters check bypass	Passed
	Correct Access control	Passed
	Built in emergency features	Passed
	Correct event logs	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	Passed
	Fallback function misuse	Reviewed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	delegatecall() vulnerabilities	Passed
	Other programming issues	Reviewed
Code Specification	Correctly declared function visibility	Passed
	Correctly declared variable storage location	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	"Out of Gas" Issue	Passed
	High consumption 'for/while' loop	Passed

	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Tokenomics Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Preliminary Audit Result: FAILED

Revised Audit Result: PASSED

Intended Smart Contract Functions

Claimed Behaviour	Actual Behaviour
madCartelNFT.sol - Uses Merkle Proofs for whitelisting - Max supply of 6000 - Cost of 1 ether - Max mint amount of 10 - Max minted per address of 20 - Has presale minting - Owner can withdraw funds earned from NFT sales	**Contract achieves this functionality.**
OperatorFilterer.sol, IOperatorFilterer.sol and DefaultOperatorFilterer.sol - Inherited abstract contract - `onlyAllowedOperator` modifier for `transferFrom` and `safeTransferFrom` methods. - `onlyAllowedOperatorApproval` modifier for `approve` and `setApprovalForAll` methods. - automatically subscribes to the default OpenSea subscription.	**Contract achieves this functionality.**

Function List

madCarteINFT.sol

Functions

	Functions	Visibility	Observation	Conclusion
1	constructor	write	Passed	No Issue
2	_baseURI	internal	Passed	No Issue
4	onlyOwner	modifier	Passed	No Issue
7	renounceOwnership	write	access only Owner	No Issue
8	transferOwnership	write	access only Owner	No Issue
9	_transferOwnership	internal	Passed	No Issue
10	presaleMint	public	Passed	Reviewed
11	mint	public	Passed	Reviewed
12	approve	public	Passed	No Issue
13	pause	public	access only Owner	Reviewed
14	renounceOwnership	public	access only Owner	No Issue
15	reveal	public	access only Owner	No Issue
16	safeTransferFrom	public	Passed	No Issue
17	setApprovalForAll	public	Passed	No Issue
18	setBaseURI	public	access only Owner	No Issue
19	setCost	public	access only Owner	No Issue
20	setNotRevealedURI	public	access only Owner	No Issue
21	setPresale	public	access only Owner	No Issue
22	setPresaleRoot	public	access only Owner	No Issue
23	transferFrom	public	Passed	No Issue
24	transferOwnership	public	access only Owner	No Issue
25	withdraw	public	onlyOwner	No Issue

26	toBytes32	internal	Passed	No Issue
27	isAllowListed	internal	Passed	Reviewed

Code Quality

This audit scope involves the solidity smart contracts of the MadCartel's project, as outlined in the Audit Scope section. All contracts, libraries and interfaces have since been refactored to follow standard best practices and to help avoid unnecessary complexity that increases the likelihood of exploitation.

All code is mostly well commented on and closely follows best practice nat-spec styling. All comments are correctly aligned with code functionality.

Audit Resources

We were given the Mad Cartels Protocol's smart contract code in the form of a zip file. As mentioned above, code parts are mostly well commented. The logic is straightforward, and therefore it is somewhat easy to quickly comprehend the programming flow as well as the complex code logic. The comments are helpful in understanding the overall architecture of the protocol.

Dependencies

As per our observation, the libraries used in this smart contracts infrastructure are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

Severity Definitions

Significance	Description
Critical	Critical vulnerabilities are of immediate concern, and allow hackers or malicious code to steal funds, take ownership of the contract, or other devastating actions.
High	High vulnerabilities are not attacked as easily, but still have significant implications on the protocol.
Medium	Medium level difficulties should be solved before deployment, but won't result in loss of funds.
Low	Low level vulnerabilities are areas that lack best practices that may cause small complications in the future.
Gas	Gas Optimisations, issues and inefficiencies

Audit Findings

Critical

No critical vulnerabilities were found.

High

1) Users can lose funds to the contract when minting.

In the case a user accidentally sends excess ether, the call to mint function will accept it and the users extra funds will be sent to the contract. There is no method of the user regaining access to those funds.

Recommendation: use "==" instead, that way if a user accidentally sends extra funds the call will revert.

Status: Resolved

2) Logic in contract is not as intended by protocol.

The protocol's desired specification was to have a max supply of 6,000, whereas the contract (As of 05/06/23) has a max supply of 10,000.

Recommendation: Update variable to match desired specification.

Status: Acknowledged

Medium

1) Compiler error

madCartelNFT.sol

```
188     function walletOfOwner(address _owner)
189         public
190         view
191         returns (uint256[] memory)
192     {
193         uint256 ownerTokenCount = balanceOf(_owner);
194         uint256[] memory tokenIds = new uint256;
195         for (uint256 i; i < ownerTokenCount; i++) {_exists
196             tokenIds[i] = tokenOfOwnerByIndex(_owner, i);
197         }
198         return tokenIds;
199     }
```

```
TypeError: Contract or array type
expected.
-->
MadCartels/madCartelNFT.sol:194:37:
|
194 | uint256[] memory tokenIds =
new uint256;
| ^^^^^^^^^^
```

Contract cannot deploy with critical compiler error. Array is not implemented correctly.

Recommendation:
uint256[] memory tokenIds = new uint256[](ownerTokenCount);

Status: Resolved

2) There is little to no test coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	0	0	0	0	
madCartelNFT.sol	0	0	0	0	... 313,314,319
contracts/on-chainRoyalties/	0	0	0	0	
DefaultOperatorFilterer.sol	100	100	0	100	
IOperatorFilterRegistry.sol	100	100	100	100	
OperatorFilterer.sol	0	0	0	0	... 61,69,72,73
contracts/on-chainRoyalties/lib/	100	100	100	100	
Constants.sol	100	100	100	100	
All files	0	0	0	0	

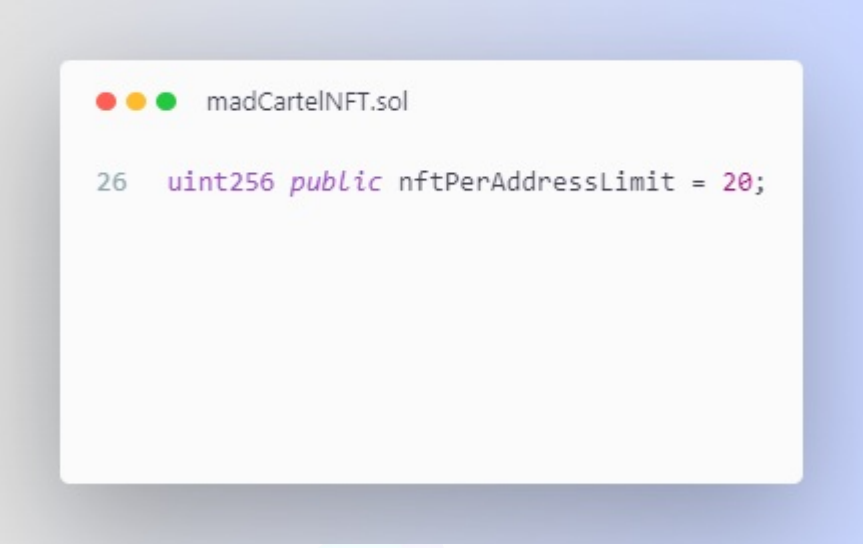
Full test coverage ensures that the code works as intended under all possible circumstances, reducing the risk of errors or unexpected behaviour.

Recommendation: Write tests for your contracts

Status: Acknowledged

Low

1) NFTPerAddressLimit variable is misleading



This state variable and its function is misleading. Addresses can have more than 20 NFT's if they mint 20 and another address sends them more.

It seems that the intended function is to limit the mint amount per address, not the amount each address can hold.

Recommendation: Change the name to mintLimitPerAddress, or change the functionality if you wish to actually limit the amount each address can hold.

Status: **Resolved**

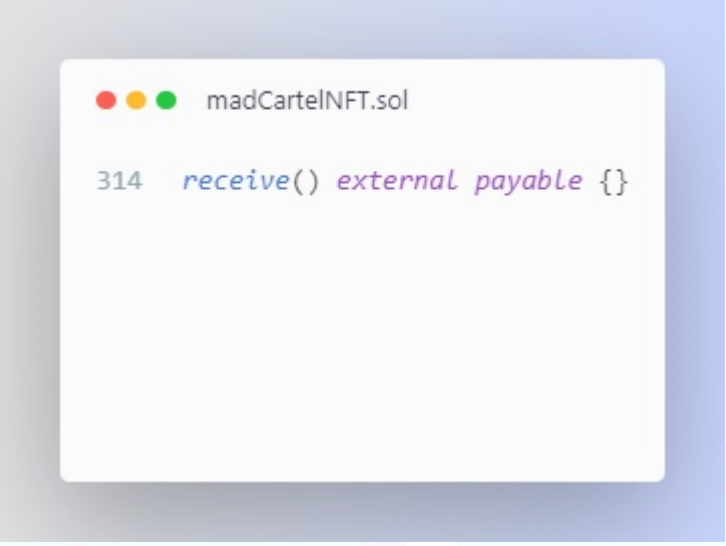
2) Not most recent solidity version

Smart contracts use up to date, but not the most recent solidity version.

Recommendation: Use ^0.8.18 as best practice.

Status: Resolved

3) Unused receive function



Smart contracts should not have unused receive or fallback functions.

Recommendation: Remove unused receive function.

Status: Resolved

4) Make variables that can't be changed constant to save on gas

Since maxSupply cannot be changed, set it to constant.
Other variables that can be set to constant: baseExtension, maxMintAmount, nftPerAddressLimit. There are no functions to change these state variables.

Recommendation: Set it to: `uint256 public constant maxSupply = 10000;`

Status: Resolved

5) Use OpenZeppelin's modules

OpenZeppelin Contracts help you minimise risk by using battle-tested libraries of smart contracts. The contract contains logic from Pausable.sol and ERC721URIStorage.sol unnecessarily, as these can be inherited, reducing code length and complexity.

Recommendation: Inherit OpenZeppelin's Pausable.sol and ERC721URIStorage.sol contracts.

Status: Acknowledged

6) Use more suitable name for require

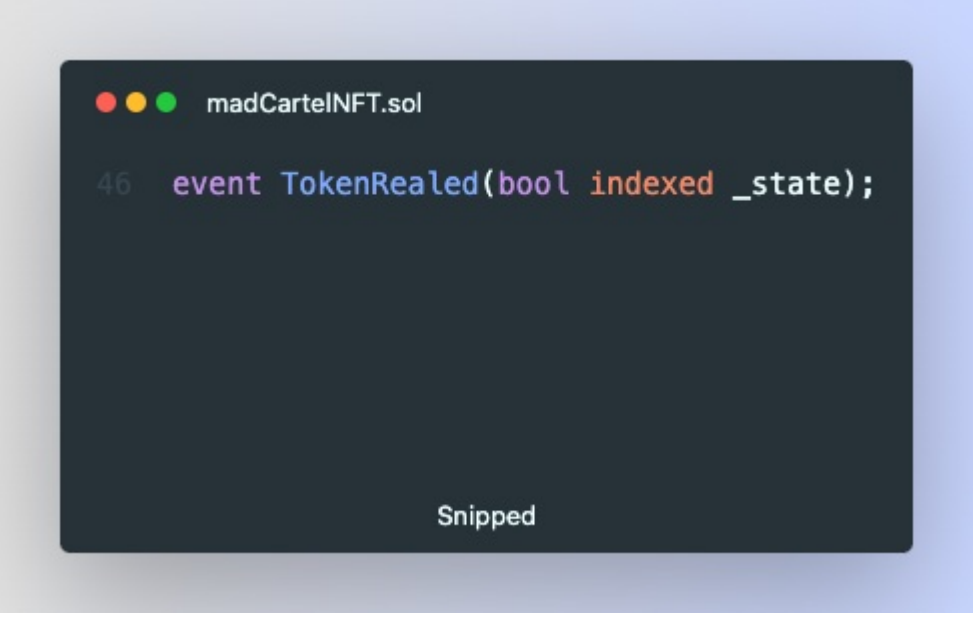
The current implementation makes the codebase more difficult to read and debug.

Recommendation: Consider using something like this instead:

```
(bool success, ) = payable(owner()).call{value: toWithdraw}("");
    require(success, "withdraw failed”);
```

Status: Resolved

7) Spelling error

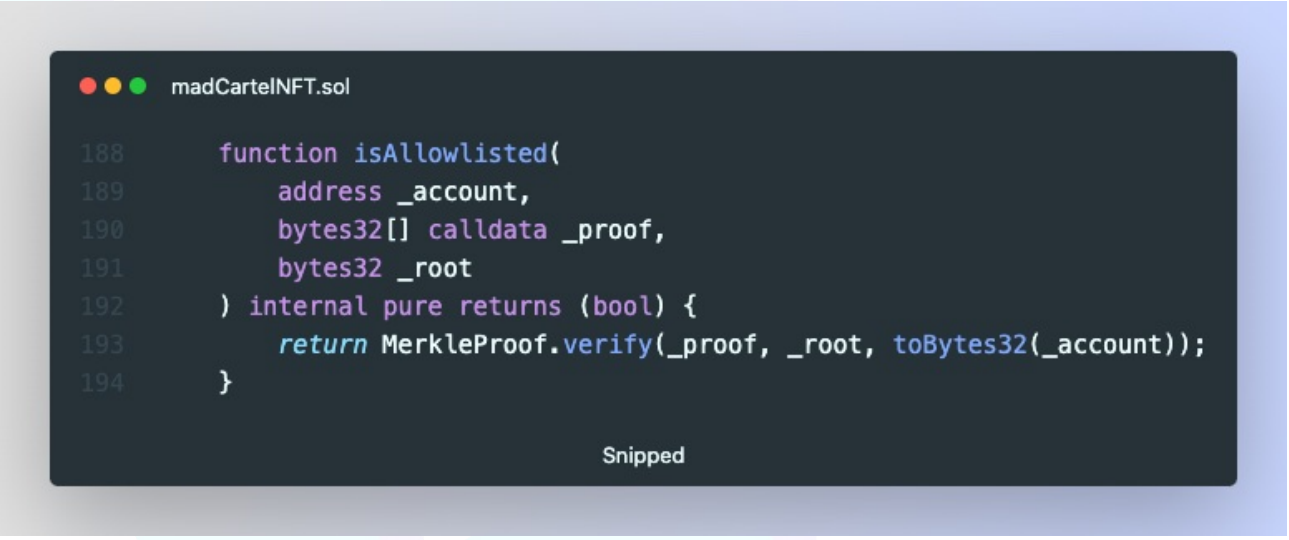


Spelling error on Revealed. Need to change on 'emit' too.

Recommendation: Should be TokenRevealed

Status: Resolved

8) No need of including presaleRoot as a parameter



PresaleRoot is passed into the isAllowListed function as a parameter however it is a state variable that does not change when used in mint function so can be used directly.

Recommendation: Insert the presale state variable directly into the merkleProof.verify and remove it as a parameter.

Status: Resolved

9) Unnecessary check that increases complexity with owner restrictions

madCarteINFT.sol

```
101         // Check if the sender is not the contract owner
102         if (msg.sender != owner()) {
103             // Check if the sender is on the allowlist with the given Merkle proof and presale root
104             require(
105                 isAllowlisted(msg.sender, proof, presaleRoot),
106                 "MC: user is not allowlisted"
107             );
108             // Get the number of tokens minted by the sender
109             uint256 ownerMintedCount = addressMintedBalance[msg.sender];
110             // Check if ownerMintedCount + _mintAmount is less than or equal to nftPerAddressLimit
111             require(
112                 ownerMintedCount + _mintAmount <= nftPerAddressLimit,
113                 "MC: max NFT per address exceeded"
114             );
115         }
```

Snipped

Owner can be added to the Allow list to avoid the above if function

Recommendation: Add owner to the Allow list and get rid of the above if function

Status: Acknowledged

10)Informational: There is no presale limit or treasury reserve implemented

NFT projects usually have a set amount of NFTs reserved for the treasury for giveaways, the team, etc. This reserved amount can also be minted first to ensure the contract works as it should on the mainnet. The smart contract currently also has no limit on how many NFTs can be sold during the presale.

Recommendation: Consider setting a presale limit and reserved amount for the project treasury.

Status: Acknowledged

Centralisation

The project values security and utility over decentralisation.

The owner executable functions within the protocol increase security and functionality but depend highly on private key security, and internal team responsibility.

Conclusion

After Hashlocks analysis, the Mad Cartels NFT project has a sound and well tested code base, after the resolution of our findings. Overall, the code is now correctly ordered and follows industry best practices.

The project does sacrifice decentralisation for security, as there is a large amount of access control, as well as the manual release of rewards. Hashlock believes this is a sound decision.

Our Methodology

Hashlock strives to maintain a transparent working process and to make our audits a collaborative effort. The objective of our security audits are to improve the quality of systems and upcoming projects we review and to aim for sufficient remediation to help protect users and project leaders. Below is the methodology we use in our security audit process.

Manual Code Review:

In manually analysing all of the code, we seek to find any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behaviour when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our methodologies include manual code analysis, user interface interaction, and whitebox penetration testing. We consider the project's website, specifications, and whitepaper (if available) to attain a high level understanding of what functionality the smart contract under review contains. We then communicate with the developers and founders to gain insight into their vision for the project. We install and deploy the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We undergo a robust, transparent process for analysing potential security vulnerabilities and seeing them through to successful remediation. When a potential issue is discovered, we immediately create an issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is vast because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyse the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinised by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the contracts details are made public.

Disclaimers

Hashlock's Disclaimer

Hashlock's team has analysed these smart contracts in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in the smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Hashlock is not responsible for the safety of any funds, and is not in any way liable for the security of the project.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to attacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

About Hashlock

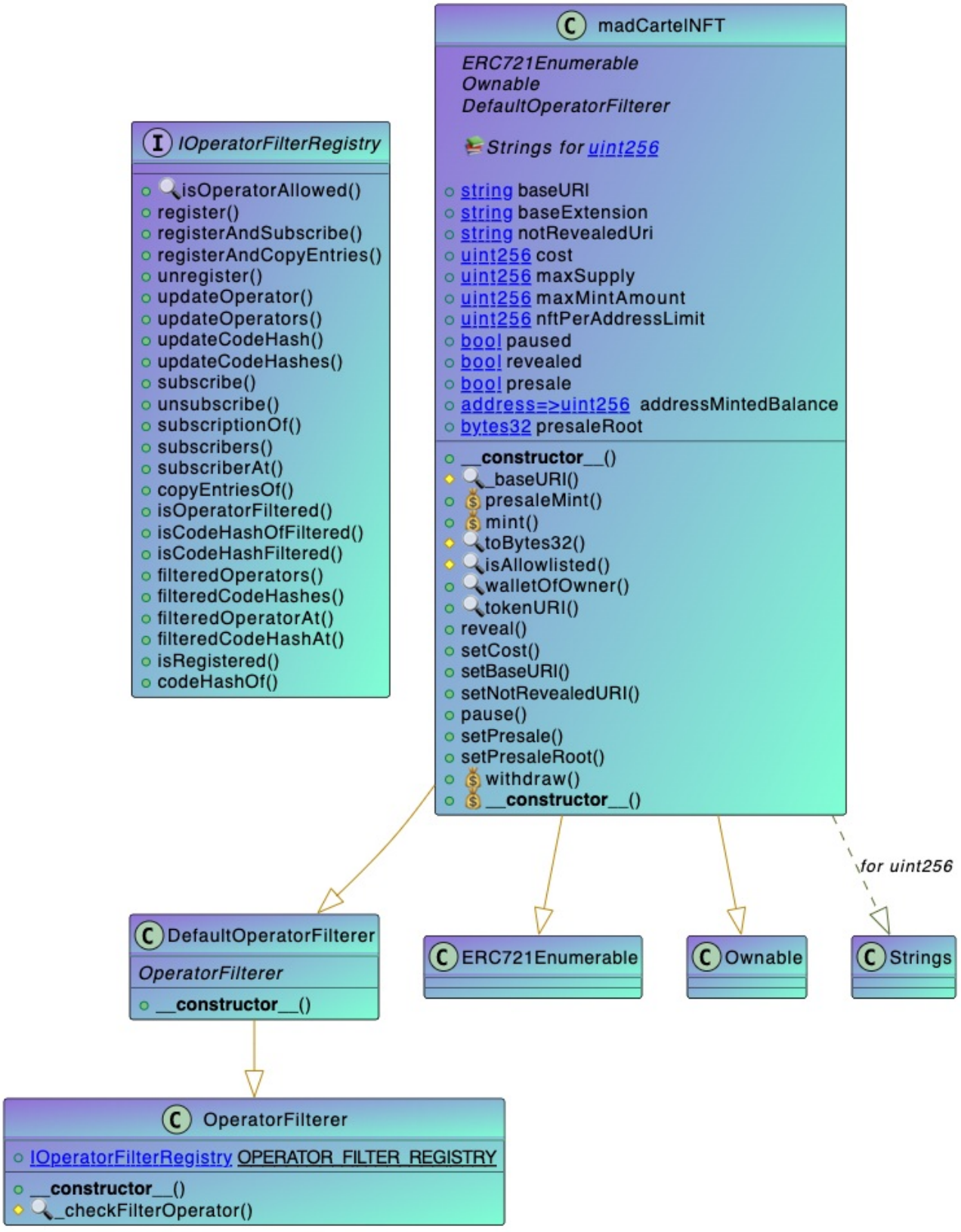
Hashlock is an Australian based company aiming to help facilitate the successful widespread adoption of distributed ledger technology. Our key services all have a focus on security, as well as projects that focus on streamlined adoption in the business sector.

Hashlock is excited to continue to grow its partnerships with developers and other web3 oriented companies to collaborate on secure innovation, helping businesses and decentralised entities alike.

Website: hashlock.com.au
Contact: info@hashlock.com.au

Appendix

Code Flow Diagram



Slither Results Log

```
Reentrancy in madCartelNFT.mint(uint256) (contracts/madCartelNFT.sol#123-157):
        External calls:
        - _safeMint(msg.sender,supply + i) (contracts/madCartelNFT.sol#155)
                - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#436-
447)
        State variables written after the call(s):
        - addressMintedBalance[msg.sender] ++ (contracts/madCartelNFT.sol#153)
        madCartelNFT.addressMintedBalance (contracts/madCartelNFT.sol#30) can be used in cross function reentrancies:
        - madCartelNFT.addressMintedBalance (contracts/madCartelNFT.sol#30)
        - madCartelNFT.mint(uint256) (contracts/madCartelNFT.sol#123-157)
        - madCartelNFT.presaleMint(uint256,bytes32[]) (contracts/madCartelNFT.sol#75-117)
Reentrancy in madCartelNFT.presaleMint(uint256,bytes32[]) (contracts/madCartelNFT.sol#75-117):
        External calls:
        - _safeMint(msg.sender,supply + i) (contracts/madCartelNFT.sol#115)
                - IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#436-
447)
        State variables written after the call(s):
        - addressMintedBalance[msg.sender] ++ (contracts/madCartelNFT.sol#113)
        madCartelNFT.addressMintedBalance (contracts/madCartelNFT.sol#30) can be used in cross function reentrancies:
        - madCartelNFT.addressMintedBalance (contracts/madCartelNFT.sol#30)
        - madCartelNFT.mint(uint256) (contracts/madCartelNFT.sol#123-157)
        - madCartelNFT.presaleMint(uint256,bytes32[]) (contracts/madCartelNFT.sol#75-117)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-1

madCartelNFT.walletOfOwner(address).i (contracts/madCartelNFT.sol#193) is a local variable never initialized
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#uninitialized-local-variables

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#429-451) ignores return value
by IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#436-447)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
```

```
madCartelNFT.presaleMint(uint256,bytes32[]) (contracts/madCartelNFT.sol#75-117) compares to a boolean constant:
        -require(bool,string)(presale == true,MC: presale is over) (contracts/madCartelNFT.sol#82)
madCartelNFT.mint(uint256) (contracts/madCartelNFT.sol#123-157) compares to a boolean constant:
        -require(bool,string)(presale == false,MC: presale is active) (contracts/madCartelNFT.sol#127)
madCartelNFT.tokenURI(uint256) (contracts/madCartelNFT.sol#204-229) compares to a boolean constant:
        -revealed == false (contracts/madCartelNFT.sol#213)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#boolean-equality

Different versions of Solidity are used:
        - Version used: ['^0.8.0', '^0.8.1', '^0.8.13', '^0.8.17']
        - ^0.8.0 (node_modules/@openzeppelin/contracts/access/Ownable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/ERC721Enumerable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Enumerable.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Context.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/Strings.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/cryptography/MerkleProof.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/ERC165.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/introspection/IERC165.sol#4)
        - ^0.8.0 (node_modules/@openzeppelin/contracts/utils/math/Math.sol#4)
        - ^0.8.1 (node_modules/@openzeppelin/contracts/utils/Address.sol#4)
        - ^0.8.13 (contracts/madCartelNFT.sol#3)
        - ^0.8.13 (contracts/on-chainRoyalties/DefaultOperatorFilterer.sol#2)
        - ^0.8.13 (contracts/on-chainRoyalties/IOperatorFilterRegistry.sol#2)
        - ^0.8.13 (contracts/on-chainRoyalties/OperatorFilterer.sol#2)
        - ^0.8.17 (contracts/on-chainRoyalties/lib/Constants.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-directives-are-used
```

```
madCartelNFT.constructor(string,string,string,bytes32)._name (contracts/madCartelNFT.sol#52) shadows:
        - ERC721._name (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#24) (state variable)
madCartelNFT.constructor(string,string,string,bytes32)._symbol (contracts/madCartelNFT.sol#53) shadows:
        - ERC721._symbol (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#27) (state variable)
madCartelNFT.walletOfOwner(address)._owner (contracts/madCartelNFT.sol#189) shadows:
        - Ownable._owner (node_modules/@openzeppelin/contracts/access/Ownable.sol#21) (state variable)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#429-451) has external calls in
side a loop: IERC721Receiver(to).onERC721Received(_msgSender(),from,tokenId,data) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#436-447)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation/#calls-inside-a-loop

Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).retval (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#436)' in ERC7
21._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#429-451) potentially used before d
eclaration: retval == IERC721Receiver.onERC721Received.selector (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#437)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#438)' in ERC7
21._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#429-451) potentially used before d
eclaration: reason.length == 0 (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#439)
Variable 'ERC721._checkOnERC721Received(address,address,uint256,bytes).reason (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#438)' in ERC7
21._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#429-451) potentially used before d
eclaration: revert(uint256,uint256)(32 + reason,mload(uint256)(reason)) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#444)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#pre-declaration-usage-of-local-variables

Reentrancy in madCartelNFT.withdraw() (contracts/madCartelNFT.sol#297-303):
        External calls:
        - (os) = address(owner()).call{value: toWithdraw}() (contracts/madCartelNFT.sol#299)
        Event emitted after the call(s):
        - Withdraw(toWithdraw) (contracts/madCartelNFT.sol#302)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

ERC721._checkOnERC721Received(address,address,uint256,bytes) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#429-451) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#443-445)
Address._revert(bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#231-243) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Address.sol#236-239)
Strings.toString(uint256) (node_modules/@openzeppelin/contracts/utils/Strings.sol#18-38) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Strings.sol#24-26)
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/Strings.sol#30-32)
MerkleProof._efficientHash(bytes32,bytes32) (node_modules/@openzeppelin/contracts/utils/cryptography/MerkleProof.sol#215-222) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/cryptography/MerkleProof.sol#217-221)
Math.mulDiv(uint256,uint256,uint256) (node_modules/@openzeppelin/contracts/utils/math/Math.sol#55-135) uses assembly
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/math/Math.sol#66-70)
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/math/Math.sol#86-93)
        - INLINE ASM (node_modules/@openzeppelin/contracts/utils/math/Math.sol#100-109)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
```

```
Low level call in madCartelNFT.withdraw() (contracts/madCartelNFT.sol#297-303):
        - (os) = address(owner()).call{value: toWithdraw}() (contracts/madCartelNFT.sol#299)
Low level call in Address.sendValue(address,uint256) (node_modules/@openzeppelin/contracts/utils/Address.sol#60-65):
        - (success) = recipient.call{value: amount}() (node_modules/@openzeppelin/contracts/utils/Address.sol#63)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#128-137):
        - (success,returndata) = target.call{value: value}(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#135)
Low level call in Address.functionStaticCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#155-162):
        - (success,returndata) = target.staticcall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#160)
Low level call in Address.functionDelegateCall(address,bytes,string) (node_modules/@openzeppelin/contracts/utils/Address.sol#180-187):
        - (success,returndata) = target.delegatecall(data) (node_modules/@openzeppelin/contracts/utils/Address.sol#185)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls

Contract madCartelNFT (contracts/madCartelNFT.sol#16-309) is not in CapWords
Parameter madCartelNFT.presaleMint(uint256,bytes32[])._mintAmount (contracts/madCartelNFT.sol#76) is not in mixedCase
Parameter madCartelNFT.mint(uint256)._mintAmount (contracts/madCartelNFT.sol#123) is not in mixedCase
Parameter madCartelNFT.isAllowlisted(address,bytes32[],bytes32)._account (contracts/madCartelNFT.sol#176) is not in mixedCase
Parameter madCartelNFT.isAllowlisted(address,bytes32[],bytes32)._proof (contracts/madCartelNFT.sol#177) is not in mixedCase
Parameter madCartelNFT.isAllowlisted(address,bytes32[],bytes32)._root (contracts/madCartelNFT.sol#178) is not in mixedCase
Parameter madCartelNFT.walletOfOwner(address)._owner (contracts/madCartelNFT.sol#189) is not in mixedCase
Parameter madCartelNFT.setCost(uint256)._newCost (contracts/madCartelNFT.sol#244) is not in mixedCase
Parameter madCartelNFT.setBaseURI(string)._newBaseURI (contracts/madCartelNFT.sol#253) is not in mixedCase
Parameter madCartelNFT.setNotRevealedURI(string)._notRevealedURI (contracts/madCartelNFT.sol#262) is not in mixedCase
Parameter madCartelNFT.pause(bool)._state (contracts/madCartelNFT.sol#271) is not in mixedCase
Parameter madCartelNFT.setPresale(bool)._state (contracts/madCartelNFT.sol#280) is not in mixedCase
Parameter madCartelNFT.setPresaleRoot(bytes32)._root (contracts/madCartelNFT.sol#289) is not in mixedCase
Function ERC721.__unsafe_increaseBalance(address,uint256) (node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol#503-505) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions

madCartelNFT.baseExtension (contracts/madCartelNFT.sol#20) should be constant
madCartelNFT.maxMintAmount (contracts/madCartelNFT.sol#25) should be constant
madCartelNFT.maxSupply (contracts/madCartelNFT.sol#24) should be constant
madCartelNFT.nftPerAddressLimit (contracts/madCartelNFT.sol#26) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
contracts/madCartelNFT.sol analyzed (18 contracts with 84 detectors), 114 result(s) found
```

Solidity Static Analysis

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in OperatorFilterer._checkFilterOperator(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 67:4:

Miscellaneous

Constant/View/Pure functions:

IOperatorFilterRegistry.register(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 14:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.registerAndSubscribe(address,address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 19:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.registerAndCopyEntries(address,address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 25:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.unregister(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 32:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.updateOperator(address,address,bool) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 37:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.updateOperators(address,address[],bool) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 42:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.updateCodeHash(address,bytes32,bool) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 47:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.updateCodeHashes(address,bytes32[],bool) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 52:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.subscribe(address,address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 61:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.unsubscribe(address,bool) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 66:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.subscriptionOf(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 71:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.subscribers(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 77:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.subscriberAt(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 83:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.copyEntriesOf(address,address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 88:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.isOperatorFiltered(address,address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 93:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.isCodeHashOfFiltered(address,address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 98:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.isCodeHashFiltered(address,bytes32) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 103:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.filteredOperators(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 108:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.filteredCodeHashes(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 114:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.filteredOperatorAt(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 121:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.filteredCodeHashAt(address,uint256) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 128:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.isRegistered(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 133:4:

Constant/View/Pure functions:

IOperatorFilterRegistry.codeHashOf(address) : Potentially should be constant/view/pure but is not. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 138:4:

Constant/View/Pure functions:

OperatorFilterer._checkFilterOperator(address) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 67:4:

No return:

IOperatorFilterRegistry.isOperatorAllowed(address,address): Defines a return type but never explicitly returns a value.
Pos: 9:4:

No return:
IOperatorFilterRegistry.subscriptionOf(address): Defines a return type but never explicitly returns a value.
Pos: 71:4:

No return:
IOperatorFilterRegistry.subscribers(address): Defines a return type but never explicitly returns a value.
Pos: 77:4:

No return:
IOperatorFilterRegistry.subscriberAt(address,uint256): Defines a return type but never explicitly returns a value.
Pos: 83:4:

No return:
IOperatorFilterRegistry.isOperatorFiltered(address,address): Defines a return type but never explicitly returns a value.
Pos: 93:4:

No return:
IOperatorFilterRegistry.isCodeHashOfFiltered(address,address): Defines a return type but never explicitly returns a value.
Pos: 98:4:

No return:
IOperatorFilterRegistry.isCodeHashFiltered(address,bytes32): Defines a return type but never explicitly returns a value.
Pos: 103:4:

No return:
IOperatorFilterRegistry.filteredOperators(address): Defines a return type but never explicitly returns a value.
Pos: 108:4:

No return:
IOperatorFilterRegistry.filteredCodeHashes(address): Defines a return type but never explicitly returns a value.
Pos: 114:4:

No return:

IOperatorFilterRegistry.filteredOperatorAt(address,uint256): Defines a return type but never explicitly returns a value.
Pos: 121:4:

No return:

IOperatorFilterRegistry.filteredCodeHashAt(address,uint256): Defines a return type but never explicitly returns a value.
Pos: 128:4:

No return:

IOperatorFilterRegistry.isRegistered(address): Defines a return type but never explicitly returns a value.
Pos: 133:4:

No return:

IOperatorFilterRegistry.codeHashOf(address): Defines a return type but never explicitly returns a value.
Pos: 138:4:

Solhint Linter

Linter results:

```
MadCartels/OperatorFilterer.sol:20:28: Error: Parse error: mismatched input '('
expecting {';', '='}
```

```
MadCartels/OperatorFilterer.sol:20:45: Error: Parse error: extraneous input ')'
expecting {';', '='}
```

```
MadCartels/OperatorFilterer.sol:73:41: Error: Parse error: mismatched input '('
expecting {';', '='}
```

Linter results:

```
MadCartels/madCartelNFT.sol:3:1: Error: Compiler version ^0.8.13 does not
satisfy the r semver requirement
```

```
MadCartels/madCartelNFT.sol:16:1: Error: Contract name must be in CamelCase
```

```
MadCartels/madCartelNFT.sol:49:5: Error: Explicitly mark visibility in function
(Set ignoreConstructors to true if using solidity >=0.7.0)
```

```
MadCartels/madCartelNFT.sol:52:30: Error: Code contains empty blocks
```

```
MadCartels/madCartelNFT.sol:283:23: Error: Avoid to use low level calls.
```

```
MadCartels/madCartelNFT.sol:292:32: Error: Code contains empty blocks
```

Linter results:

```
MadCartels/DefaultOperatorFilterer.sol:2:1: Error: Compiler version ^0.8.13 does
not satisfy the r semver requirement
```

```
MadCartels/DefaultOperatorFilterer.sol:16:5: Error: Explicitly mark visibility
in function (Set ignoreConstructors to true if using solidity >=0.7.0)
```

```
MadCartels/DefaultOperatorFilterer.sol:16:71: Error: Code contains empty blocks
```

Linter results:

```
MadCartels/IOperatorFilterRegistry.sol:2:1: Error: Compiler version ^0.8.13 does
not satisfy the r semver requirement
```