Cyberscope

Audit Report

Creath Governance Token

September 2023

SHA256 03d3b75c81d189464e95c2464cb347a1c4e7ce7a1ef1c0fc0620a1b141a4313a

Audited by © cyberscope

Table of Contents

Review

Audit Updates

Initial Audit	20 Sep 2023

Source Files

Filename	SHA256
CreathTreasury.sol	7c5096d327f0aaf4dffef08562e79a615f01a132ee6c5b6f1398ddd96851250a
CreathMarketplace.sol	b7f200f7d398acd558098b4792c721428accc1c8bbb233d9d143ea204fe8060a
CreathArtTradable.sol	691edf8148911146c04c4369f5f33a5b3eb9f7fa2731b4fa38e702377bc42da2
CreathArtFactory.sol	ddc52325e70c217b92be961bb7a4126edef2e4abd9944d2f5e323eaa6f015ea9
Creath.sol	03d3b75c81d189464e95c2464cb347a1c4e7ce7a1ef1c0fc0620a1b141a4313a

Overview

The Creath platform represents an advanced NFT infrastructure, offering a multifaceted marketplace which make possible the purchase of NFT listings. Alongside this, it features a dedicated factory mechanism for the creation of new NFT contracts tailored to specific criteria. To bolster its financial operations, the ecosystem incorporates a robust treasury module, ensuring the safe accumulation and distribution of funds.

Creath contract

The Creath contract serves as the foundational layer for the Creath Marketplace. This contract, inheriting from the `ERC721URIStorage` and `Ownable` standards, is equipped with functionalities that empower the contract owner with distinct privileges. Specifically, the owner can mint new Non-Fungible Tokens (NFTs) and assign them to a designated `_beneficiary` address. Each minted NFT is associated with a unique token ID and a specified URI that provides detailed information about the token. Additionally, the contract owner retains the authority to burn NFTs based on their specific `_tokenId`. The contract also incorporates mechanisms to check the existence of a token, verify approvals, ensuring seamless interactions within the Creath ecosystem. As a result the contract integrates with the marketplace, allowing for streamlined trading and management of NFTs.

CreathArtFactory Conctract

The CreathArtFactory contract, is facilitating the creation and management of NFT contracts. This contract, inheriting from the `Ownable` standard, grants the owner exclusive privileges to deploy new NFT contracts through the `createNFTContract` function. When invoked, this function establishes a new NFT contract with specified `_name` and `_symbol` parameters, while also associating it with the predefined `marketplace` address. Beyond creation, the `CreathArtFactory` contract offers functionalities to register NFT contracts via the `registerTokenContract` method. This function ensures that only genuine ERC721 compliant contracts are registered, enhancing the security and integrity of the platform. Additionally, the owner can disable any registered NFT contract using the `disableTokenContract` function, providing a mechanism for quality control and contract lifecycle management. Throughout these operations, relevant events such as `ContractCreated` and `ContractDisabled` are emitted, ensuring transparency and traceability within the ecosystem.

CreathArtTradable Contract

The CreathArtTradable contract, is facilitating the creation and management of individual NFT contracts. This contract, inheriting from both `ERC721URIStorage` and `Ownable` , is equipped with the capability to mint new NFTs to a designated beneficiary address, by the mint function. Each minted NFT is uniquely identified by a token ID, which is systematically incremented to ensure distinctiveness.

While it shares many functionalities with the Creath contract, a distinguishing feature of the `CreathArtTradable` contract is its initialization process. This contract mandates the specification of parameters such as `_name` , `_symbol` , and `_marketplace` during its initialization. Upon initialization, the `CreathArtTradable` contract requires specific parameters, namely `_name` , `_symbol` , and `_marketplace` , to be defined. These parameters set the stage for the NFT's identity. Additioanlly, the contract's owner, granted exclusive privileges, can mint new NFTs and also burn them if necessary, by using the burn function.

CreathMarketplace Contract

The `CreathMarketplace` contract is using the `initialize` function which sets the `platformFee` , `feeReceipient` , and the `paymentToken` parameters of the marketplace.

The marketplace is structured around four core functionalities:

1. ListItem Function The `listItem` function facilitates the process of listing an NFT for sale. It incorporates checks to ensure that only ERC721 compliant NFT contracts are eligible for listing. Before listing, the function verifies that the NFT hasn't been previously listed. Once these checks are passed, the NFT's price and associated artist are set, marking it as available for purchase.
2. CancelListing Function The `cancelListing` function offers the flexibility to cancel the NFTs from the marketplace. Upon invocation, the function erases the

NFT's price and artist details from the contract's records, effectively delisting it from the marketplace.

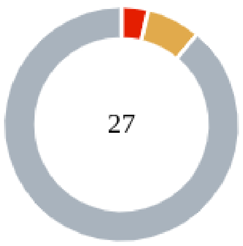
3. UpdateListing Function The `updateListing` function is designed to provide the ability to adjust the price of the already listed NFTs. By setting the new desired price as a parameter, the price of the NFTs can be adjusted .
4. BuyItem Function Potential buyers can use the `buyItem` function to purchase a listed NFT. The function first calculates a `feeAmount` based on the set `platformFee` . This fee is then transferred to the `feeReceipient` address. The remaining amount, after deducting the fee, is sent to the NFT's associated `artist` address. Finally, ownership of the NFT is transferred to the buyer, completing the transaction.

In addition to these core functionalities, the contract is equipped with administrative functions that grant the owner the capability to modify key parameters. Specifically, the owner can update the `platformFee` , `feeReceipient` , and the `paymentToken` , ensuring that the marketplace remains adaptable to evolving requirements and conditions.

CreathTreasury Conctract

The `CreathTreasury` is the contract where tokens are accumulated. Designed with robust security measures, the contract incorporates functionalities that allows the withdrawal of both tokens and native tokens to designated addresses. These withdrawal functions can only be invoked by authorized entities, which the owner have set. The contract is equipped with mechanisms to authorize new addresses and modify existing roles, ensuring that the treasury operations remain flexible and secure.

Findings Breakdown



●	Critical	1
●	Medium	2
●	Minor / Informative	24

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	2	0	0	0
●	Minor / Informative	24	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ⚪ Minor / Informative

Severity	Code	Description	Status
🔴	IPU	Ineffective Price Update	Unresolved
🟠	ZPL	Zero Price Listing	Unresolved
🟠	MAV	Missing Address Validation	Unresolved
⚪	CCR	Contract Centralization Risk	Unresolved
⚪	RES	Redundant Event Statement	Unresolved
⚪	PBV	Percentage Boundaries Validation	Unresolved
⚪	RCC	Redundant Condition Check	Unresolved
⚪	MVN	Misleading Variables Naming	Unresolved
⚪	RNRM	Redundant No Reentrant Modifier	Unresolved
⚪	EIS	Excessively Integer Size	Unresolved
⚪	PDNP	Potential Duplicate NFT Parameters	Unresolved
⚪	RSW	Redundant Storage Writes	Unresolved
⚪	OSI	Override Specification Inconsistency	Unresolved
⚪	RF	Redundant Function	Unresolved

●	MEE	Missing Events Emission	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

IPU - Ineffective Price Update

Criticality	Critical
Location	CreathMarketplace.sol#L2002
Status	Unresolved

Description

The contract contains the `updateListing` function whose purpose is to update the listing price of a specific NFT. However, while the function retrieves the current price of the NFT into the `listedItem` variable and then sets this variable to `_newPrice`, it fails to subsequently update the actual mapping listings `[_nftAddress][_tokenId]` with this new price value. As a result, the value of `listings[_nftAddress][_tokenId]` remains unchanged, rendering the price update ineffective.

```
function updateListing(
        address _nftAddress,
        address _artist,
        uint256 _tokenId,
        uint256 _newPrice
    ) external onlyOwner nonReentrant isListed(_nftAddress,
_tokenId) {
        uint listedItem = listings[_nftAddress][_tokenId];

        listedItem = _newPrice;
        emit ItemUpdated(
            _artist,
            _nftAddress,
            _tokenId,
            _newPrice
        );
    }
```

Recommendation

It is recommended that the team refactor the `updateListing` function to ensure the proper updating of the `listings[_nftAddress][_tokenId]` mapping. To achieve the intended purpose of updating the price, the mapping `listings[_nftAddress][_tokenId]` should directly be set to the `_newPrice` passed as a parameter. This will ensure that the price update is effective and the contract behaves as expected.

ZPL - Zero Price Listing

Criticality	Medium
Location	CreathMarketplace.sol#L1923,1960
Status	Unresolved

Description

The contract contains the `listItem` function which allows the listing of an NFT for sale. This function utilizes the `notListed` modifier, which checks if the listing price of the NFT is less than or equal to zero, indicating that the NFT hasn't been listed with a zero price. However, the `listItem` function does not explicitly prevent the `_price` parameter from being set to zero. As a result the `notListed` modifier's condition allows for the NFT to be listed again if its price is zero. This means that an NFT can be listed multiple times with a zero price, which potential contradicts the actual functionality of the contract.

```
modifier notListed(
        address _nftAddress,
        uint256 _tokenId
    ) {
        uint listing = listings[_nftAddress][_tokenId];
        require(listing <= 0, "Creath Marketplace:already
listed");
        _;
    }

 function listItem(
        address _nftAddress,
        address _artist,
        uint256 _tokenId,
        uint256 _price
    ) external onlyOwner notListed(_nftAddress, _tokenId) {
        ...
        listings[_nftAddress][_tokenId] = _price;

        artists[_nftAddress][_tokenId] = _artist;
        ...
    }
```

Recommendation

It is recommended to reconsider the purpose of the `listItem` function. If the intended functionality is to prevent NFTs from being listed for with a zero price, then the `listItem` function should be enhanced to include a check ensuring that the `_price` parameter passed is greater than zero. This will prevent NFTs from being listed with a zero price and uphold the integrity of the marketplace by ensuring that all listed NFTs have a valid price.

MAV - Missing Address Validation

Criticality	Medium
Location	CreathMarketplace.sol#L1978,2051
Status	Unresolved

Description

The contract contains the `listItem` function that allows for the listing of an NFT for sale. This function accepts an `_artist` address parameter, which represents the artist's address to which payments should be sent upon the sale of the NFT. However, the `listItem` function does not contain any checks to prevent the `_artist` address from being set to the zero address (0x0). Subsequently, when the `_buyItem` function is invoked to purchase an NFT, the contract sends a portion of the `paymentToken` to the artist's address. If the artist's address was inadvertently set to the zero address during the listing process, the `withdrawToken` function will transfer the `paymentToken` to the zero address, resulting in a permanent loss of those tokens.

```
function listItem(
        address _nftAddress,
        address _artist,
        uint256 _tokenId,
        uint256 _price
    ) external onlyOwner notListed(_nftAddress, _tokenId) {
        ...
        artists[_nftAddress][_tokenId] = _artist;
        ...
        );
    }

  function _buyItem(
        address _nftAddress,
        uint256 _tokenId,
        address buyer
    ) private {
        uint listedItem = listings[_nftAddress][_tokenId];

        uint256 feeAmount = (listedItem.mul(platformFee)).div(100);
        ...
        address artist = artists[_nftAddress][_tokenId];
        ...
        if
(IERC165Upgradeable(_nftAddress).supportsInterface(INTERFACE_ID_ERC
721)) {
            IERC721Upgradeable(_nftAddress).transferFrom(owner(),
buyer, _tokenId);
        }
        ...
        delete (listings[_nftAddress][_tokenId]);
    }
```

Recommendation

It is recommended to reconsider the functionality of both the `listItem` and `_buyItem` functions. If the intended behavior is to prevent the artist's address from being set to the zero address during the listing process, then the `listItem` function should be enhanced to include a check ensuring that the `_artist` address is not the zero address. Alternatively, if the intended functionality is to allow the zero address to be set as the artist, then the `_buyItem` function should be modified to ensure that the `paymentToken` is only transferred to the artist if the artist's address is not the zero address. This dual approach will prevent unintentional loss of tokens and ensure the integrity of the payment process.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	Creath.sol#L1775CreathArtTradable.sol#L1776CreathMarketplace.sol#L1992
Status	Unresolved

Description

The contract is designed with a high degree of centralization, granting the owner significant authority over its operations. Specifically, the owner has the power to set and modify the `paymentToken` of the contract, determining which token will be used for purchasing NFTs. This level of control allows the owner to potentially switch the payment token at will, which could disrupt users' expectations and operations.

Furthermore, the owner possesses the capability to burn any token from any user. This means that even if an NFT is purchased, the owner can unilaterally decide to burn it, depriving the user of their acquired asset. Such centralized control not only poses risks to the users but also deviates from the decentralized nature of blockchain and smart contracts.

```
   function burn(uint256 _tokenId) external onlyOwner{
        _burn(_tokenId);
    }

    function updatePaymentToken(address newPaymentToken)
external onlyOwner{
        paymentToken = IERC20Upgradeable(newPaymentToken);
    }
```

Recommendation

It is recommended to address these centralization concerns by evaluating the feasibility of integrating critical configurations and functionality directly into the contract's codebase. By doing so, the contract would become more self-reliant, reducing its dependence on external configurations and the associated risks. This would also ensure that the contract operates in a more predictable and transparent manner, aligning with the principles of decentralization. Additionally, to further safeguard users' assets, the team should consider implementing checks that prevent the owner from burning NFTs that have been purchased by users. This protective measure would enhance trust in the contract and protect users from potential arbitrary actions by the owner.

RES - Redundant Event Statement

Criticality	Minor / Informative
Location	CreathTreasury.sol#L1463
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The `MarketplaceSet` event statement is not used in the contract's implamantation.

```
event MarketplaceSet( address _address);
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it. It is recommend removing the unused event statement from the contract..

PBV - Percentage Boundaries Validation

Criticality	Minor / Informative
Location	CreathMarketplace.sol#L1930,2040,2076
Status	Unresolved

Description

The contract is using the variables `platformFee` for calculations. However, this variable is used in multiplication operations and if the `platformFee` is set to a value greater than `100`, it could lead to incorrect calculations. The owner has the ability to set the `platformFee` variable to a value greater than `100` within the `initialize` function or by invoking the `updatePlatformFee` function. This could potentially cause unintended behavior within the contract's operations. It's crucial to ensure that the `platformFee` is appropriately validated to prevent potential miscalculations.

Additionally, during the contract initialization, the contract does not prevent the owner to set the platformFee.

```
    function initialize(
        address payable _feeRecipient,
        address _token,
        uint16 _platformFee)
        public
        initializer
    {
        ...
        _platformFee;
        ...
    }

    uint256 feeAmount = (listedItem.mul(platformFee)).div(100);

    function updatePlatformFee(uint16 _platformFee) external
onlyOwner {
        platformFee = _platformFee;
        emit UpdatePlatformFee(_platformFee);
    }
```

Recommendation

It is recommended to ensure that the values of `platformFee` cannot exceed the value of `100`. This can be achieved by adding checks whenever these variables are set.

RCC - Redundant Condition Check

Criticality	Minor / Informative
Location	CreathMarketplace.sol#L2055
Status	Unresolved

Description

The contract is using the `_buyItem` function that contains an if statement checking if the provided `_nftAddress` supports the ERC721 interface using the condition `IERC165Upgradeable(_nftAddress).supportsInterface(INTERFACE_ID_ERC721)`. This check is intended to ensure that the ERC721 interface is supported during the purchase of an item. However, this check is redundant, since an item can only be listed for sale if it already supports the ERC721 interface, as verified in the `listItem` function. Consequently, if an item cannot be listed due to the absence of ERC721 support, it inherently means it cannot be bought either. This renders the interface check within the `_buyItem` function superfluous.

```
function listItem(
        address _nftAddress,
        address _artist,
        uint256 _tokenId,
        uint256 _price
    ) external onlyOwner notListed(_nftAddress, _tokenId) {
        if
(IERC165Upgradeable(_nftAddress).supportsInterface(INTERFACE_ID_ERC
721)) {
            IERC721Upgradeable nft =
IERC721Upgradeable(_nftAddress);
            require(
                nft.isApprovedForAll(owner(), address(this)),
                "item not approved"
            );
        } else {
            revert("Creath Marketplace:invalid nft address");
        }
    ...
    }

    function _buyItem(
        address _nftAddress,
        uint256 _tokenId,
        address buyer
    ) private {
        ...
        if
(IERC165Upgradeable(_nftAddress).supportsInterface(INTERFACE_ID_ERC
721)) {
            IERC721Upgradeable(_nftAddress).transferFrom(owner(),
buyer, _tokenId);
        }
```

Recommendation

It is recommended to remove the redundant if statement from the `_buyItem` function. Since the support for the ERC721 interface is already verified in the `listItem` function, there is no need to recheck it when the NFT is being bought. This will lead to a more streamlined contract, optimized gas usage, and enhanced clarity in the codebase.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Location	CreathMarketplace.sol#L2008,2038
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

Specifically, the contract contains the `listedItem` variable which is derived from the `listings[_nftAddress][_tokenId]` mapping. This mapping is intended to store the price of the NFT. However, the variable name `listedItem` is misleading as it suggests that the variable might represent an item that is listed, rather than its actual purpose, which is to represent the price of the NFT.

```
uint listedItem = listings[_nftAddress][_tokenId];
```

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. It is recommended to rename the `listedItem` variable to a more descriptive and accurate name that reflects that the actual purpose of that variable is to represent price. This will ensure clarity and reduce potential confusion.

RNRM - Redundant No Reentrant Modifier

Criticality	Minor / Informative
Location	CreathMarketplace.sol#L1992,2027
Status	Unresolved

Description

The contract uses the `nonReentrant` modifier to the `cancelListing` and `updateListing` functions, which suggests an intention to prevent potential reentrancy attacks. However, neither of these functions deals with the transfer of the native token or any other value. As such, the risk of reentrancy attacks in these specific functions is minimal to non-existent.

```
   function cancelListing(address _nftAddress, address _artist,
uint256 _tokenId)
        external
        onlyOwner
        nonReentrant
        isListed(_nftAddress, _tokenId)
    {
        _cancelListing(_nftAddress,_artist, _tokenId);
    }

    function buyItem(
        address _nftAddress,
        uint256 _tokenId
    )
        external
        nonReentrant
        isListed(_nftAddress, _tokenId)
    {
        _buyItem(_nftAddress, _tokenId, msg.sender);
    }
```

Recommendation

To address this finding and enhance code simplicity and clarity, it is recommended to remove the unnecessary `nonReentrant` modifier from the `cancelListing` and `updateListing` functions. By removing the modifier, the code becomes more streamlined and easier to comprehend, reducing the gas consuption.

EIS - Excessively Integer Size

Criticality	Minor / Informative
Location	CreathMarketplace.sol#L1881,1896,1933,2076
Status	Unresolved

Description

The contract is using a bigger unsigned integer data type that the maximum size that is required. By using an unsigned integer data type larger than necessary, the smart contract consumes more storage space and requires additional computational resources for calculations and operations involving these variables. This can result in higher transaction costs, longer execution times, and potential scalability bottlenecks.

Specifically using a `uint16` data type for the `platformFee` variable. However, the maximum potential value for the `platformFee` is `100`. As a result, a `uint8` data type, which can represent values from 0 to 255, would be more appropriate and efficient in terms of storage.

```
uint16 platformFee
uint16 public platformFee;
```

Recommendation

To address the inefficiency associated with using an oversized unsigned integer data type, it is recommended to accurately determine the required size based on the range of values the variable needs to represent. It is recommended to use a `uint8` for the `platformFee` variable, since the maximum value of the `platformFee` could be set up to `100`.

PDNP - Potential Duplicate NFT Parameters

Criticality	Minor / Informative
Location	CreathArtFactory.sol#L1859
Status	Unresolved

Description

The contract is utilizing the `createNFTContract` function that allows the deployment of a new `CreathArtTradable` contract. This function initializes the new NFT contract with the provided `_name`, `_symbol`, and the `marketplace` address. However, the contract does not incorporate any mechanism to prevent the creation of an NFT contract with identical parameters to one that already exists. Consequently, there is the possibility for multiple NFT contracts to be created with the same `_name`, `_symbol`, and `marketplace` values.

```
function createNFTContract(string memory _name, string memory
_symbol)
        external
        onlyOwner
        returns (address)
    {
        CreathArtTradable nft = new CreathArtTradable(
            _name,
            _symbol,
            marketplace
        );
        exists[address(nft)] = true;
        nft.transferOwnership(owner());
        emit ContractCreated(_msgSender(), address(nft));
        return address(nft);
    }
```

Recommendation

It is recommended to reconsider the intended functionality of the `createNFTContract` function. If the primary objective of the function is to ensure unique NFT contracts, then the function should incorporate a mechanism to check and prevent the creation of contracts with duplicate characteristics. This can be achieved by verifying the uniqueness of the `_name` and `_symbol` parameters before the contract creation process.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	CreathArtFactory.sol#L1851
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract updates the address the `marketplace` variable even if its current address is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
    function updateMarketplace(address _marketplace) external
onlyOwner {
        marketplace = _marketplace;
    }
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

OSI - Override Specification Inconsistency

Criticality	Minor / Informative
Location	Creath.sol#L1810CreathArtTradable.sol#L1792
Status	Unresolved

Description

The contract is using an `override` within the `isApprovedForAll` function. However, the `override` keyword does not specify which contracts are being overridden. As a result, if the contract is compiled using version `0.8.12` of Solidity or higher, it will encounter compilation errors due to the stricter requirements for specifying overridden contracts.

```
    function isApprovedForAll(address _owner, address operator)
        override
    ....
    }
```

Recommendation

It is recommended to specify the `contracts` that will need to be overridden. If the contract is considered for deployment using a Solidity version of `0.8.12` or above, then the `override(ERC721, IERC721)` syntax should be used instead of the simple `override`, if the intended contracts to be overridden is the `ERC721` and `IERC721`. This will ensure that the contract adheres to the newer Solidity requirements and can be compiled without issues.

RF - Redundant Function

Criticality	Minor / Informative
Location	Creath.sol#L1780
Status	Unresolved

Description

The contract is using the `_extractIncomingTokenId` function which is declared as `internal`. However, even though this function is declared as `internal`, it is not utilized or called anywhere within the contract. As a result, this function is redundant and does not influence or affect any functionality inside the contract.

```
    function _extractIncomingTokenId() internal pure returns
(uint256) {
        // Extract out the embedded token ID from the sender
        uint256 _receiverTokenId;
        uint256 _index = msg.data.length - 32;
        assembly {_receiverTokenId := calldataload(_index)}
        return _receiverTokenId;
    }
```

Recommendation

It is recommended to consider removing the `_extractIncomingTokenId` function from the contract. If the current implementation does not require the functionality of this function, then it could be removed to optimize gas usage and streamline the contract's codebase.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Creath.sol#L1775CreathArtFactory.sol#L1851CreathMarketplace.sol#L1952CreathTreasury.sol#L1505
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
    function burn(uint256 _tokenId) external onlyOwner{
        _burn(_tokenId);
    }
```

```
    function updateMarketplace(address _marketplace) external
onlyOwner {
        marketplace = _marketplace;
    }
```

```
    function updatePaymentToken(address newPaymentToken)
external onlyOwner{
        paymentToken = IERC20Upgradeable(newPaymentToken);
    }
```

```
    function updateMarketplace(address _marketplace) external
onlyRole("admin"){
        _setupRole("marketplace", _marketplace);
        marketplace = _marketplace;
    }
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	CreathMarketplace.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	CreathMarketplace.sol#L298,308,318,328,338,348,348,358,368,368CreathArtTradable.sol#L25CreathArtFactory.sol#L25Creath.sol#L25
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
AddressSlot storage r
BooleanSlot storage r
Bytes32Slot storage r
Uint256Slot storage r
StringSlot storage r
string storage store
BytesSlot storage r
bytes storage store
Counter storage counter
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	CreathArtTradable.sol#L1752Creath.sol#L1746
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
marketplace
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	CreathTreasury.sol#L309,1456,1505,1517,1528CreathMarketplace.sol#L232,903,907,951,973,976,991,1023,1027,1088,1111,1114,1260,1286,1289,1292,1374,1906,1931,1932,1933,1961,1962,1963,1964,1989,2003,2004,2005,2006,2023,2024,2076,2086,2094CreathArtTradable.sol#L1635,1761,1776,1784,1791CreathArtFactory.sol#L1550,1761,1776,1784,1851,1859 Creath.sol#L1635,1755,1775,1794,1802,1809
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
address private ADMIN;
(address _marketplace)
(address _token, address _to, uint _amount)
(uint _amount, address _to)
...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	CreathTreasury.sol#L67,92,121,148,158,173,183,222,427,438,453,462,475,488,527,611,618,626,635,663,670,678,689,699,783,796,832,843,885,896,934,947,977,1004,1029,1036,1067,1405CreathMarketplace.sol#L318,328,338,348,358,368,512,537,566,593,603,667,839,850,857,903,907,942,973,976,1111,1114,1194,1201,1211,1225,1232,1247,1286,1289,1484,1503,1518,1527,1540,1553,1592CreathArtTradable.sol#L35,43,62,69,77,86,114,121,129,140,150,234,247,283,294,336,385,398,428,480,487,496,511,518,700,725,735,754,764,781,791,806,816,831,855,867,1635CreathArtFactory.sol#L35,43,62,69,77,86,114,121,129,140,150,234,247,283,294,336,385,398,428,480,487,496,511,518,588,613,623,642,652,669,679,694,704,719,743,755,1550Creath.sol#L35,43,62,69,77,86,114,121,129,140,150,234,247,283,294,336,385,398,428,480,487,496,511,518,700,725,735,754,764,781,791,806,816,831,855,867,1635,1780
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value,
recipient may have reverted");
    }

function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

function functionCallWithValue(address target, bytes memory
data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value,
"Address: low-level call with value failed");
    }

...
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	CreathTreasury.sol#L745,748,760,764,765,766,767,768,769,775CreathArtTradable.sol#L196,199,211,215,216,217,218,219,220,226CreathArtFactory.sol#L196,199,211,215,216,217,218,219,220,226Creath.sol#L196,199,211,215,216,217,218,219,220,226
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
denominator := div(denominator, twos)
inverse *= 2 - denominator * inverse
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	CreathMarketplace.sol#L1175
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
bytes32 slot
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	CreathArtTradable.sol#L1748,1749,1791,1812CreathArtFactory.sol#L1748,1749,1812Creath.sol#L1809,1830
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
string memory _name
string memory _symbol
address _owner, address operator
address _owner = ERC721.ownerOf(tokenId);
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	CreathTreasury.sol#L1475,1502,1507,1529CreathMarketplace.sol#L1940, 2090CreathArtTradable.sol#L1752CreathArtFactory.sol#L1752,1842,1852 Creath.sol#L1746
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
ADMIN = _admin;
...
feeReceipient = _feeRecipient
feeReceipient = _platformFeeRecipient
...
marketplace = _marketplace;
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	CreathTreasury.sol#L239,706,1010CreathMarketplace.sol#L300,310,320,330,340,350,360,370,684CreathArtTradable.sol#L157,461,872,1585CreathArtFactory.sol#L157,461,760,1500Creath.sol#L157,461,872,1585,1784
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }

assembly {
...
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

assembly {
                ptr := add(buffer, add(32, length))
            }


...
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	CreathTreasury.sol#L7,254,317,398,543,571,602,648,990,1077,1104,1195,1443CreathArtTradable.sol#L7,53,99,441,528,555,640,887,917,945,953,984,1118,1126,1148,1177,1645,1720CreathArtFactory.sol#L7,53,99,441,528,775,805,833,841,872,1006,1014,1036,1065,1092,1560,1636,1720,1821Creath.sol#L7,53,99,441,528,555,640,887,917,945,953,984,1118,1126,1148,1177,1645,1720
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.0;
pragma solidity ^0.8.0;
pragma solidity ^0.8.1;
...
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	CreathTreasury.sol#L7,254,317,398,543,571,602,648,990,1077,1104,1195,1443CreathMarketplace.sol#L7,35,43,177,241,381,410,429,452,699,867,959,999,1096,1268,1382,1463,1608,1824CreathArtTradable.sol#L7,53,99,441,528,555,640,887,917,945,953,984,1118,1126,1148,1177,1645,1720CreathArtFactory.sol#L7,53,99,441,528,775,805,833,841,872,1006,1014,1036,1065,1092,1560,1636,1720,1821Creath.sol#L7,53,99,441,528,555,640,887,917,945,953,984,1118,1126,1148,1177,1645,1720
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.1;
pragma solidity ^0.8.0;
pragma solidity ^0.8.0;
pragma solidity ^0.8.0;
...
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

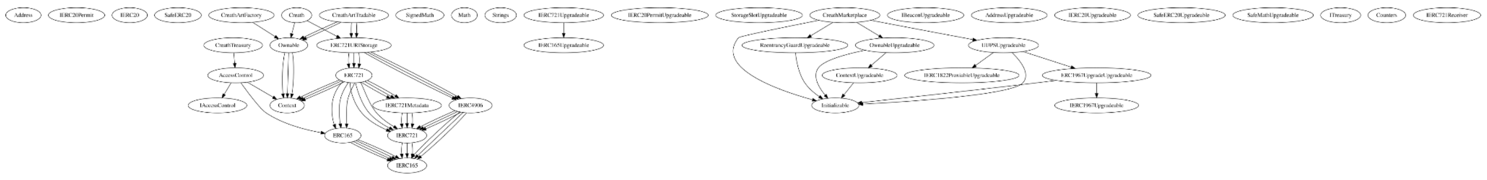
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
CreathTreasury	Implementation	AccessControl		
		Public	✓	-
	isAuthorized	Public		-
	updateAdmin	External	✓	onlyRole
	updateMarketplace	External	✓	onlyRole
	withdrawToken	Public	✓	onlyAuthorized
	withdraw	Public	✓	onlyAuthorized
		External	Payable	-
ITreasury	Interface			
	withdrawToken	External	✓	-
CreathMarketplace	Implementation	Initializable, UUPSUpgradeable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	_authorizeUpgrade	Internal	✓	onlyOwner

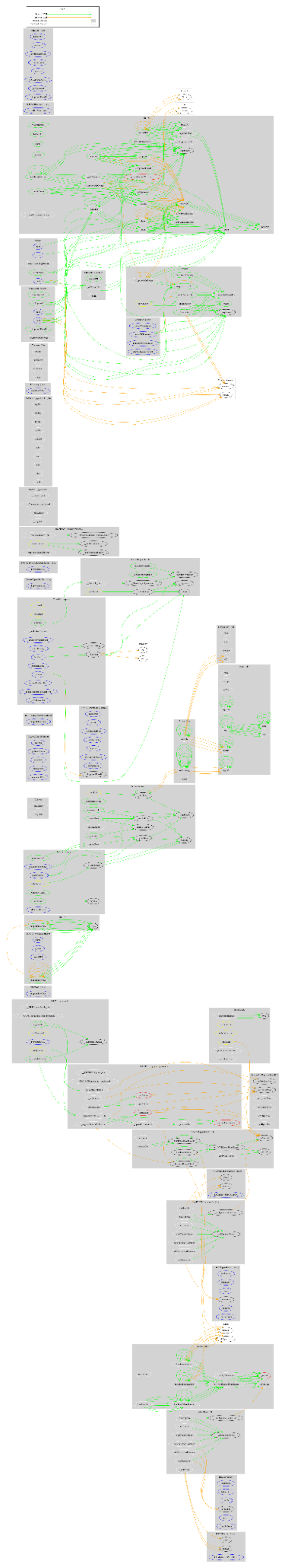
	updatePaymentToken	External	✓	onlyOwner
	listItem	External	✓	onlyOwner notListed
	cancelListing	External	✓	onlyOwner nonReentrant isListed
	updateListing	External	✓	onlyOwner nonReentrant isListed
	buyItem	External	✓	nonReentrant isListed
	_buyItem	Private	✓	
	updatePlatformFee	External	✓	onlyOwner
	updatePlatformFeeRecipient	External	✓	onlyOwner
	getCollectorData	External		-
	_cancelListing	Private	✓	
CreathArtTradable	Implementation	ERC721URIStorage, Ownable		
		Public	✓	ERC721
	mint	External	✓	onlyOwner
	burn	External	✓	onlyOwner
	isApproved	Public		-
	isApprovedForAll	Public		-
	_isApprovedOrOwner	Internal		
CreathArtFactory	Implementation	Ownable		

		Public	✓	-
	updateMarketplace	External	✓	onlyOwner
	createNFTContract	External	✓	onlyOwner
	registerTokenContract	External	✓	onlyOwner
	disableTokenContract	External	✓	onlyOwner
Creath	Implementation	ERC721URIStorage, Ownable		
		Public	✓	ERC721
	mint	External	✓	onlyOwner
	burn	External	✓	onlyOwner
	_extractIncomingTokenId	Internal		
	exists	External		-
	isApproved	Public		-
	isApprovedForAll	Public		-
	_isApprovedOrOwner	Internal		

Inheritance Graph



Flow Graph



Summary

The Creath project is a comprehensive NFT ecosystem that encompasses a marketplace for buying the listed NFTs, alongside a factory for generating new NFT contracts based on specific parameters. Additionally, it integrates a treasury system to securely manage and distribute funds, ensuring a seamless and secure transaction experience for users within the platform.

Disclaimer

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io