Cryptoart.com Audit Report

Prepared by Cyfrin

Version 2.0

Lead Auditors

Dacian

Hans

April 24, 2025

Contents

1 About Cyfrin **2**

2 Disclaimer **2**

3 Risk Classification **2**

4 Protocol Summary **2**

5 Audit Scope **2**

6 Executive Summary **2**

7 Findings **5**

7.1 Medium Risk 5

7.1.1 Collector can add `CreatorStory`, corrupting the provenance of an artwork 5

7.2 Low Risk 6

7.2.1 Allow custom Creator and Collector names to be emitted in `IStory` events to build artwork provenance 6

7.2.2 Prevent code-injection inside `string` fields when emitting `IStory` events or setting `tokenURI` fields 6

7.2.3 Allow users to increment their nonce to void their signatures 7

7.2.4 `IERC7160` specification requires `hasPinnedTokenURI` to revert for non-existent `tokenId` 7

7.2.5 `IERC7160` specification requires `pinTokenURI` to revert for non-existent `tokenId` 7

7.2.6 Inconsistent pause functionality allows certain state-changing operations when contract is paused 8

7.3 Informational 9

7.3.1 Protocol vulnerable to cross-chain signature replay 9

7.3.2 Signatures have no expiration deadline 9

7.3.3 Consider limiting max royalty to prevent large amount or all of the sale fee being taken as royalty 9

7.3.4 `MintType` is almost never enforced 9

7.3.5 Remove unused constant `CryptoartNFT::ROYALTY_BASE` 10

7.4 Gas Optimization 11

7.4.1 Prefer named return parameters, especially for `memory` returns 11

7.4.2 Use named constants to indicate purpose of magic numbers 11

7.4.3 Remove obsolete `onlyTokenOwner` from `_transferToNftReceiver` 12

7.4.4 In `tokenURI` avoid copying entire `_tokenURIs[tokenId]` from `storage` into `memory` 12

7.4.5 `burn` should delete `tokenURI` related data and emit `TokenUriUnpinned` event 12

7.4.6 To prevent duplicate ids in `_batchBurn`, enforce ascending order instead of nested `for` loops 13

1 About Cyfrin

Cyfrin is a Web3 security company dedicated to bringing industry-leading protection and education to our partners and their projects. Our goal is to create a safe, reliable, and transparent environment for everyone in Web3 and DeFi. Learn more about us at cyfrin.io.

2 Disclaimer

The Cyfrin team makes every effort to find as many vulnerabilities in the code as possible in the given time but holds no responsibility for the findings in this document. A security audit by the team does not endorse the underlying business or product. The audit was time-boxed and the review of the code was solely on the security aspects of the solidity implementation of the contracts.

3 Risk Classification

	Impact: High	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4 Protocol Summary

Cryptoart.com is a protocol which uses pairable NFTs backed by physical artwork, allowing NFT holders to:

- pair (redeem) their NFT to receive a physical artwork corresponding to their NFTs
- unpair (unredeem) their NFT by destroying the physical artwork
- trade NFTs, both paired and unpaired
- burn NFTs, trading them for new ones which require burning in order to mint
- build on-chain "provenance" around an artwork via the `IStory` interface

5 Audit Scope

The scope of this audit was limited to one file plus its associated interfaces and dependencies: `src/CryptoartNFT.sol`

6 Executive Summary

Over the course of 2 days, the Cyfrin team conducted an audit on the Cryptoart.com smart contracts provided by Cryptoart.com. In this period, a total of 18 issues were found.

The findings consist of 1 Medium and 6 Lows, with the remainder being informational and gas optimizations.

The single Medium finding resulted in the owner of an NFT being able to corrupt the provenance of an artwork by adding to the `CreatorStory`, which only a Creator should be able to do.

The 6 Lows were a mix of findings related to signature handling, inconsistent pausing of functionality and minor ERC specification violations.

Centralization Risks

The protocol is centralized by nature featuring:

- upgradeable contract which allows the contract owner to change the implementation
- off-chain services which users must interact with in order to perform core protocol functions

Summary

Project Name	Cryptoart.com
Repository	cryptoart-smart-contracts
Commit	1dc1068fbe98...
Fix Commit	86142422c65e...
Audit Timeline	Apr 21st - Apr 22nd, 2025
Methods	Manual Review

Issues Found

Critical Risk	0
High Risk	0
Medium Risk	1
Low Risk	6
Informational	5
Gas Optimizations	6
Total Issues	18

Summary of Findings

[M-1] Collector can add `CreatorStory`, corrupting the provenance of an artwork	Resolved
[L-1] Allow custom Creator and Collector names to be emitted in `IStory` events to build artwork provenance	Resolved
[L-2] Prevent code-injection inside `string` fields when emitting `IStory` events or setting `tokenURI` fields	Acknowledged
[L-3] Allow users to increment their nonce to void their signatures	Resolved
[L-4] `IERC7160` specification requires `hasPinnedTokenURI` to revert for non-existent `tokenId`	Resolved
[L-5] `IERC7160` specification requires `pinTokenURI` to revert for non-existent `tokenId`	Resolved
[L-6] Inconsistent pause functionality allows certain state-changing operations when contract is paused	Resolved

[I-1] Protocol vulnerable to cross-chain signature replay	Resolved
[I-2] Signatures have no expiration deadline	Resolved
[I-3] Consider limiting max royalty to prevent large amount or all of the sale fee being taken as royalty	Resolved
[I-4] `MintType` is almost never enforced	Resolved
[I-5] Remove unused constant `CryptoartNFT::ROYALTY_BASE`	Resolved
[G-1] Prefer named return parameters, especially for `memory` returns	Resolved
[G-2] Use named constants to indicate purpose of magic numbers	Resolved
[G-3] Remove obsolete `onlyTokenOwner` from `_transferToNftReceiver`	Resolved
[G-4] In `tokenURI` avoid copying entire `_tokenURIs[tokenId]` from `storage` into `memory`	Resolved
[G-5] `burn` should delete `tokenURI` related data and emit `TokenUriUnpinned` event	Resolved
[G-6] To prevent duplicate ids in `_batchBurn`, enforce ascending order instead of nested `for` loops	Resolved

7 Findings

7.1 Medium Risk

7.1.1 Collector can add `CreatorStory`, corrupting the provenance of an artwork

Description: The purpose of the `IStory` interface is to allow 3 different entities (Admin, Creator and Collectors) to add "Stories" about a given artwork (NFT) which describes the provenance of the artwork. In the art world the "provenance" of an item can affect its status and price, so the `IStory` interface aims to facilitate an on-chain record of an artwork's "provenance".

`IStory` is designed to work like this:

- Creator/Admin can add a `CollectionsStory` for when a collection is added to a contract
- Creator of an artwork can add a `CreatorStory`
- Collector of an artwork can add one or more `Story` about their experience while holding the artwork

The `IStory` interface specification requires that `addCreatorStory` is only called by the creator:

```
/// @notice Function to let the creator add a story to any token they have created
/// @dev This function MUST implement logic to restrict access to only the creator
function addCreatorStory(uint256 tokenId, string calldata creatorName, string calldata story) external;
```

But in the CryptoArt implementation of the `IStory` interface, the current token owner can always emit `CreatorStory` events:

```
function addCreatorStory(uint256 tokenId, string calldata, /*creatorName*/ string calldata story)
    external
    onlyTokenOwner(tokenId)
{
    emit CreatorStory(tokenId, msg.sender, msg.sender.toHexString(), story);
}
```

Impact: As an NFT is sold or transferred to new owners, each subsequent owner can continue to add new `CreatorStory` events even though they aren't the Creator of the artwork. This corrupts the provenance of the artwork by allowing Collectors to add to the `CreatorStory` as if they were the Creator.

Recommended Mitigation: Only the Creator of an artwork should be able to emit the `CreatorStory` event. Currently the on-chain protocol does not record the address of the creator; this could either be added or `onlyOwner` could be used where the contract owner acts as a proxy for the creator.

CryptoArt: Fixed in commit 94bfc1b.

Cyfrin: Verified.

7.2 Low Risk

7.2.1 Allow custom Creator and Collector names to be emitted in `IStory` events to build artwork provenance

Description: The `IStory` interface is designed to allow custom names to be emitted for the Creator and Collector events. Here is an example where a Creator has used the custom name `lytke`.

But in CryptoArt's implementation of `IStory` interface, custom names are not allowed and it is always the caller's hex string that will be set:

```
function addCollectionStory(string calldata, /*creatorName*/ string calldata story) external onlyOwner {
    emit CollectionStory(msg.sender, msg.sender.toHexString(), story);
}

/// @inheritdoc IStory
function addCreatorStory(uint256 tokenId, string calldata, /*creatorName*/ string calldata story)
    external
    onlyTokenOwner(tokenId)
{
    emit CreatorStory(tokenId, msg.sender, msg.sender.toHexString(), story);
}

/// @inheritdoc IStory
function addStory(uint256 tokenId, string calldata, /*collectorName*/ string calldata story)
    external
    onlyTokenOwner(tokenId)
{
    emit Story(tokenId, msg.sender, msg.sender.toHexString(), story);
}
```

Impact: Custom names should be allowed as they form part of the "provenance" of an artwork; the value of an artwork is often based on who the creator was and if it has been held by significant collectors in the past. Proper custom names are a lot easier to remember and tell a story about rather than 0x1343335...Artworks with custom names will be able to build a better story around them resulting in improved "provenance".

CryptoArt: Fixed in commit 77f34a4.

Cyfrin: Verified.

7.2.2 Prevent code-injection inside `string` fields when emitting `IStory` events or setting `tokenURI` fields

Description: An attack vector which spans the intersection between web3 and web2 is when users can associate arbitrary metadata strings with NFTs and those strings are later processed or displayed on a website.

In particular the `IStory::Story` event:

- is emitted by a non-trusted entity, the current holder of the artwork
- emits two arbitrary string parameters, `collectorName` and `story`
- these string parameters are designed to be displayed to users and may be processed by web2 apps

Recommended Mitigation: The most important validation is for non-trusted user-initiated functions, eg:

- When a Creator emits `CreatorStory` or a Collector emits `Story`, revert if the `name` and `story` strings contain any unexpected special characters
- When minting tokens revert if `TokenURISet::uriWhenRedeemable` and `uriWhenNotRedeemable` contain any unexpected special characters - though this must be done using off-chain components controlled by the protocol so risk here is minimal
- In off-chain code don't trust any user-supplied strings but sanitize them or check them for unexpected special characters

CryptoArt: Acknowledged; mitigation handled off-chain via URI validation pre-signing, Story string sanitization/encoding post-event.

7.2.3 Allow users to increment their nonce to void their signatures

Description: Currently users are unable to void their signatures by incrementing their nonce, since `NoncesUpgradeable::_useNonce` is `internal` and only called during actions which verify user signatures.

A user may want to invalidate a previous signature to prevent it from being used but is unable to.

Impact: Users are unable to invalidate previous signatures before they are used.

Recommended Mitigation: Expose `NoncesUpgradeable::_useNonce` via a `public` function that allows users to increment their own nonce.

CryptoArt: Fixed in commit cf82aeb.

Cyfrin: Verified.

7.2.4 `IERC7160` specification requires `hasPinnedTokenURI` to revert for non-existent `tokenId`

Description: Per the specification of `IERC7160`:

```
/// @notice Check on-chain if a token id has a pinned uri or not
/// @dev This call MUST revert if the token does not exist
function hasPinnedTokenURI(uint256 tokenId) external view returns (bool pinned);
```

But the implementation of `hasPinnedTokenURI` doesn't revert for tokens which don't exist, instead it will simply return `false` or even return `true` if a token was burned when the value was true since burning doesn't delete `_hasPinnedTokenURI` (another issue has been created to track this):

```
function hasPinnedTokenURI(uint256 tokenId) external view returns (bool) {
    return _hasPinnedTokenURI[tokenId];
}
```

Recommended Mitigation: Use the `onlyIfTokenExists` modifier:

```
-    function hasPinnedTokenURI(uint256 tokenId) external view returns (bool) {
+    function hasPinnedTokenURI(uint256 tokenId) external view onlyIfTokenExists(tokenId) returns
↪   (bool) {
```

CryptoArt: Fixed in commit 56d0e22.

Cyfrin: Verified.

7.2.5 `IERC7160` specification requires `pinTokenURI` to revert for non-existent `tokenId`

Description: Per the specification of `IERC7160`:

```
/// @notice Pin a specific token uri for a particular token
/// @dev This call MUST revert if the token does not exist
function pinTokenURI(uint256 tokenId, uint256 index) external;
```

But the implementation of `pinTokenURI` doesn't revert for tokens which don't exist, since `_tokenURIs[tokenId].length` will always equal 2 even for non-existent `tokenId`:

```
// mapping value always has fixed array size of 2
mapping(uint256 tokenId => string[2] tokenURIs) private _tokenURIs;

function pinTokenURI(uint256 tokenId, uint256 index) external onlyOwner {
    if (index >= _tokenURIs[tokenId].length) {
        revert Error.Token_IndexOutOfBounds(tokenId, index, _tokenURIs[tokenId].length - 1);
    }

    _pinnedURIIndex[tokenId] = index;

    emit TokenUriPinned(tokenId, index);
    emit MetadataUpdate(tokenId);
}
```

Recommended Mitigation: Use the `onlyIfTokenExists` modifier:

```diff
-    function pinTokenURI(uint256 tokenId, uint256 index) external onlyOwner {
+    function pinTokenURI(uint256 tokenId, uint256 index) external onlyIfTokenExists(tokenId) onlyOwner
↪  {
```

CryptoArt: Fixed in commit 0409ae4.

Cyfrin: Verified.

7.2.6 Inconsistent pause functionality allows certain state-changing operations when contract is paused

Description: The `CryptoartNFT` contract implements a pause mechanism using OpenZeppelin's `PausableUpgradeable` contract. However, the pause functionality is inconsistently applied across the contract's functions. While minting and burning operations are properly protected with the `whenNotPaused` modifier, several other state-changing functions remain accessible even when the contract is paused, including token transfers, metadata management, and story-related functions.

The following state-changing functions lack the `whenNotPaused` modifier:

1. Token transfers and approvals (inherited from ERC721)
2. Metadata management functions:
    - `updateMetadata`
    - `pinTokenURI`
    - `markAsRedeemable`
3. Story-related functions:
    - `addCollectionStory`
    - `addCreatorStory`
    - `addStory`
    - `toggleStoryVisibility`

Impact: When the contract is paused (typically during emergencies or upgrades), users can still perform various state-changing operations that might be undesirable during a pause period. It could lead to unexpected state changes during contract upgrades or emergency situations.

Recommended Mitigation: Add the `whenNotPaused` modifier to all state-changing functions to ensure consistent behavior when the contract is paused. For example:

Cryptoart: Fixed in commit e7d7e5b.

Cyfrin: Verified.

7.3 Informational

7.3.1 Protocol vulnerable to cross-chain signature replay

Description: As signatures do not include`chainId`, signature verification is vulnerable to cross-chain replay.

Impact: Although the protocol plans to deploy cross-chain in the future, the specification of this audit is to only consider deployment to one chain. Hence this finding is only Informational as this attack path is not possible when the protocol is only deployed on one chain.

Recommended Mitigation: Include `block.chainid` as a signature parameter.

CryptoArt: Fixed in commit 1e25f8c.

Cyfrin: Verified.

7.3.2 Signatures have no expiration deadline

Description: Signatures which have no expiration parameter effectively grant a lifetime license. Consider adding an expiration parameter to the signature that if used after that time results in the signature being invalid.

CryptoArt: Fixed in commit a93977d.

Cyfrin: Verified.

7.3.3 Consider limiting max royalty to prevent large amount or all of the sale fee being taken as royalty

Description: Currently `updateRoyalties` and `setTokenRoyalty` allow the contract owner to set a royalty up to `10_000` which would take the entire sale fee as a royalty. Consider limiting these functions to set the max royalty to something more reasonable like 1000 (10%).

CryptoArt: Fixed in commit 1d1125e.

Cyfrin: Verified.

7.3.4 `MintType` is almost never enforced

Description: The contract has an enumeration `MintType` which defines several types of mints:

```
enum MintType {
    OpenMint,
    Whitelist,
    Claim,
    Burn
}
```

But there are never any checks for these mint types, for example:

- there is no check for `MintType.Whitelist` and no corresponding whitelist enforcement
- the `claim` function doesn't enforce input `data.mintType == MintType.Claim`
- similarly `burnAndMint` doesn't enforce input `data.mintType == MintType.Burn`

The only place input `data.mintType` is used is in `_validateSignature` to validate that the input parameter matches what was signed, but there is no other validation that the correct mint types are being used for the correct operations.

CryptoArt: Fixed in commit deaf964.

Cyfrin: Verified.

7.3.5 Remove unused constant `CryptoartNFT::ROYALTY_BASE`

Description: The `CryptoartNFT` contract defines a constant `ROYALTY_BASE` with a value of 10,000 that is never used in the contract. This constant is intended to represent the denominator for royalty percentage calculations (where 10,000 = 100%), but it's not referenced anywhere in the contract's implementation.

Recommended Mitigation: Remove the unused constant to improve code clarity and reduce deployment gas costs.

Cryptoart: Fixed in commit 0c0dd8c.

Cyfrin: Verified.

7.4 Gas Optimization

7.4.1 Prefer named return parameters, especially for `memory` returns

Description: Prefer named return parameters, especially for memory returns. For example `tokenURIs` can be refactored to remove local variables and explicit return:

```
function tokenURIs(uint256 tokenId)
    external
    view
    override
    onlyIfTokenExists(tokenId)
    returns (uint256 index, string[2] memory uris, bool isPinned)
{
    index = _getTokenURIIndex(tokenId);
    uris = _tokenURIs[tokenId];
    isPinned = _hasPinnedTokenURI[tokenId];
}
```

CryptoArt: Fixed in commit bdd28fa.

Cyfrin: Verified.

7.4.2 Use named constants to indicate purpose of magic numbers

Description: Use named constants to indicate purpose of magic numbers. For example in reference to the value of the `_tokenURIs` mapping:

- instead of using literal `2`, use existing named constant `URIS_PER_TOKEN`:

```
CryptoartNFT.sol
72:    mapping(uint256 tokenId => string[2] tokenURIs) private _tokenURIs;
358:        returns (uint256, string[2] memory, bool)
361:        string[2] memory uris = _tokenURIs[tokenId];
698:        string[2] memory uris = _tokenURIs[tokenId];
```

- when setting uris in `updateMetadata` and `_setTokenURIs`, use named constants for the indexes:

```
function updateMetadata(uint256 tokenId, string calldata newRedeemableURI, string calldata
↪   newNotRedeemableURI)
    external
    onlyOwner
    onlyIfTokenExists(tokenId)
{
    _tokenURIs[tokenId][URI_REDEEMABLE_INDEX] = newRedeemableURI;
    _tokenURIs[tokenId][URI_NOT_REDEEMABLE_INDEX] = newNotRedeemableURI;
    emit MetadataUpdate(tokenId); // ERC4906
}
```

This can also save gas for example in `pinTokenURI`, instead of using `_tokenURIs[tokenId].length` just use the constant `URIS_PER_TOKEN` since it never changes:

```
function pinTokenURI(uint256 tokenId, uint256 index) external onlyOwner {
    if (index >= URIS_PER_TOKEN) {
        revert Error.Token_IndexOutOfBounds(tokenId, index, URIS_PER_TOKEN - 1);
    }
```

CryptoArt: Fixed in commit 97ef0ad.

Cyfrin: Verified.

7.4.3 Remove obsolete `onlyTokenOwner` from `_transferToNftReceiver`

Description: Since `_transferToNftReceiver` calls `ERC721Upgradeable::safeTransferFrom`, the `onlyTokenOwner` modifier is obsolete and inefficient as:

- `safeTransferFrom` calls `transferFrom`
- `transferFrom` calls `_update` passing `_msgSender()` as the last `auth` parameter
- `_update` calls `_checkAuthorized` since the `auth` parameter was valid
- `_checkAuthorized` calls `_isAuthorized` which verifies the caller is either the token's owner or someone who the token owner has approved

CryptoArt: Fixed in commit 75e179b.

Cyfrin: Verified.

7.4.4 In `tokenURI` avoid copying entire `_tokenURIs[tokenId]` from `storage` into `memory`

Description: `tokenURI` only uses the "pinned" URI index so there's no reason to copy both token URIs from `storage` to `memory`. Simply use a `storage` reference like this:

```diff
    function tokenURI(uint256 tokenId)
        public
        view
        override(ERC721Upgradeable)
        onlyIfTokenExists(tokenId)
        returns (string memory)
    {
-       string[2] memory uris = _tokenURIs[tokenId];
+       string[2] storage uris = _tokenURIs[tokenId];
        string memory uri = uris[_getTokenURIIndex(tokenId)];

        if (bytes(uri).length == 0) {
            revert Error.Token_NoURIFound(tokenId);
        }

        return string.concat(_baseURI(), uri);
    }
```

CryptoArt: Fixed in commit 591fed0.

Cyfrin: Verified.

7.4.5 `burn` should delete `tokenURI` related data and emit `TokenUriUnpinned` event

Description: The `burn` function should delete `tokenURI` related data and emit `TokenUriUnpinned` event:

```diff
    function burn(uint256 tokenId) public override whenNotPaused {
        // require sender is owner or approved has been removed as the internal burn function already
        ↪  checks this
        ERC2981Upgradeable._resetTokenRoyalty(tokenId);
        ERC721BurnableUpgradeable.burn(tokenId);
        emit Burned(tokenId);
+       emit TokenUriUnpinned(tokenId);
+       delete _tokenURIs[tokenId];
+       delete _pinnedURIIndex[tokenId];
+       delete _hasPinnedTokenURI[tokenId];
    }
```

This provides a gas refund as part of the burn and also removes token data that should no longer exist. It also prevents `hasPinnedTokenURI` from returning `true` for a burned token since that function doesn't check for valid token id (another issue has been created to track this).

CryptoArt: Fixed in commit b554763.

Cyfrin: Verified.

7.4.6 To prevent duplicate ids in `_batchBurn`, enforce ascending order instead of nested `for` loops

Description: In `_batchBurn` to prevent duplicate ids, instead of using nested `for` loops it is more efficient to enforce ascending order of ids using only 1 `for` loop.

Additionally, the duplicate id check can be completely removed since if there is a duplicate id the second `burn` call will revert. For example this test added to `test/unit/BurnOperationsTest.t.sol`:

```
    function test_DoubleBurn() public {
        // Mint a token to user1
        mintNFT(user1, TOKEN_ID, TOKEN_PRICE, TOKEN_PRICE);
        testAssertions.assertTokenOwnership(nft, TOKEN_ID, user1);

        // First burn should succeed
        vm.prank(user1);
        nft.burn(TOKEN_ID);

        // Second burn should revert since token no longer exists
        vm.prank(user1);
        // vm.expectRevert();
        nft.burn(TOKEN_ID);
    }
```

Results in:

```
     [4294] TransparentUpgradeableProxy::fallback(1)
        [3940] CryptoartNFT::burn(1) [delegatecall]
           ← [Revert] ERC721NonexistentToken(1)
        ← [Revert] ERC721NonexistentToken(1)
     ← [Revert] ERC721NonexistentToken(1)
```

CryptoArt: Fixed in commit 3c39fb8.

Cyfrin: Verified.