Cyberscope

Audit Report

Endemic

August 2023

Github	https://github.com/Endemic-NFT/ethereum-contracts
Commit	87164b9a6c787b3290f3f05f01e211c8e076166a
Audited by	© cyberscope

Table of Contents

Review

Repository	https://github.com/Endemic-NFT/ethereum-contracts
Commit	87164b9a6c787b3290f3f05f01e211c8e076166a

Audit Updates

Initial Audit	25 Aug 2023

Source Files

Filename	SHA256
RoyaltiesProvider.sol	6ec0562120de9594a676234f0913fef9581fac9ec4a4e3fcadf1ea605fb33ec5
PaymentManager.sol	406f322f40ce5d0ed2a1be571267e3bfc50dee346e557484b2e2f9590c5aa456
EndemicExchange.sol	9f0722e1b7cfd84778d1d82472efd27c6b1a6a9439e3446564df968fd7915387
EndemicCollectionFactory.sol	a7e8b560b8bf7e7827e12a012d5f5c8a2a265465b446fccf1861e11372f43de4
Collection.sol	222a06f2649b181fb9e9bf5aa217d063143a7719fd106a618e5d928db5699a45
mixins/MintApproval.sol	e51aa46efe31557f7b18accfd9bd6cc68963971c6ae573d50553fb390ebe077e
mixins/EndemicSale.sol	2017136118ecae17120d9043336f66c1bbcbf74eaf181f0fef9f07536fe509bf
mixins/EndemicOffer.sol	78a96323807a441ac66d255c97e4b6b9bbf47e751ae8661c1510fb7778d72e36
mixins/EndemicNonceManager.sol	8d6b4184746ce4d39e5214b333625e638cb2a4d0df6bc11d769205ceb38dd479
mixins/EndemicFundsDistributor.sol	b050118a5f5fbf3a4064c5144251768064c9d086fbf77b3a8b268256aacf947a
mixins/EndemicExchangeCore.sol	a2bd3e90e6f038491c3508b60f107d9da9690bffbdf47cddf6fe6f10062ac501
mixins/EndemicEIP712.sol	34a9aac1b6fa138fedb1b9c517ee6f0e49009ec534160f165e408a1584537eb2

mixins/ERC721Base.sol	5c7a227fd2f8cdf11d4d2f85ba506e492e6 4e463b1844f7ebc4e5bb75593101e
mixins/CollectionRoyalties.sol	b758072d00702353dc387b5fd2ee7405b0 1b5d99fa952803dd3981b21dc1e127
mixins/CollectionFactory.sol	a525bfd71536b0f5e4591710a4948984ae5 c69d72b03b8d80a18581ae7633265
mixins/auction/EndemicReserveAuction.sol	cffc25769d715c9405a997d807189478586 066ca192a50065cdb483ef994b1e2
mixins/auction/EndemicDutchAuction.sol	30fef4448f1ad11d6e10248f7c57da5a6c90 44ebd0c563044e5be69cada1926d
interfaces/IRoyaltiesProvider.sol	b0652d231e7a8c6bebaeb5b0cfbf6b921d 4037efe8a3e23aa91f6b6f7deeeaac
interfaces/IPaymentManager.sol	7a399fdd43d886f97f8b6650ea07725987ff b907dd4873203fbc59d38200aea6
interfaces/IERC721A.sol	4413c48ad3cc872156bbb4daa93f49b349 44d5cc21d06bc1f6f08fc3cb6a5dd9
interfaces/IERC2981Royalties.sol	bac456a95a7f22055f526017518f76951e8 c81755eff9fbc63544f0d30a9bce4
interfaces/ICollectionInitializer.sol	7e6c1e13898e708511c50cfe8f8454a67af 0dd4c3ba1ee559aeb0212b07b83fb
access/AdministratedUpgradable.sol	a2fafa9b4075f35936db364303a77dfb1efa 99d143733afd8cee22ebb9c2f6e5

Overview

The Cyberscope team audited five contracts within the Endemic ecosystem: EndemicCollectionFactory, Collection, EndemicExchange, PaymentManager and RoyaltiesProvider. The Endemic ecosystem presents a collection of thoughtfully designed smart contracts that form a sturdy foundation for the generation, trading, and control of Non-Fungible Tokens (NFTs). These contracts empower creators, collectors, and users in the blockchain space. The `EndemicCollectionFactory` contract serves as a portal for effortlessly initializing new NFT contracts, while the `Collection` contract offers a versatile framework for crafting and managing NFT collections, supported by extensive ERC-721 compatibility and royalty assistance. The `EndemicExchange` contract introduces a comprehensive platform that brings together various auction and sale methods, ensuring a dynamic and adaptable NFT trading experience. The `PaymentManager` and `RoyaltiesProvider` contracts further enhance this ecosystem by enabling fee management across diverse payment methods and equitable distribution of royalties, respectively. With a focus on resilience and user-oriented functionality, the Endemic ecosystem establishes itself as a cornerstone for the evolving NFT landscape.

Roles

Collection Contract

CollectionFactory

The CollectionFactory role has authority over the following functions:

- `function initialize(address creator, string memory name, string memory symbol, uint256 royalties, address administrator)`

Owner

The Owner role has authority over the following functions:

- `function mint(address recipient, string calldata tokenCID, uint8 v, bytes32 r, bytes32 s, uint256 nonce)`
- `function batchMint(address recipient, string[] calldata tokenCIDs, uint8 v, bytes32 r, bytes32 s, uint256 nonce)`
- `function mintAndApprove(address recipient, string calldata tokenCID, address operator, uint8 v, bytes32 r, bytes32 s, uint256 nonce)`
- `function batchMintAndApprove(address recipient, string[] calldata tokenCIDs, address operator, uint8 v, bytes32 r, bytes32 s, uint256 nonce)`
- `function setRoyalties(address recipient, uint256 value)`
- `function __CollectionRoyalties_init(address recipient, uint256 royalties)`

Administrator

The Administrator role has authority over the following functions:

- `function renounceAdministration()`
- `function transferAdministration(address newAdmin)`
- `function toggleMintApproval()`
- `function updateMintApprover(address newMintApprover)`

User

The User role can interact with the following functions:

- `function tokenURI(uint256 tokenId)`
- `function supportsInterface(bytes4 interfaceId)`
- `function royaltyInfo(uint256, uint256 value)`
- `function totalSupply()`

EndemicCollectionFactory Contract

Minter

The Minter role has authority over the following functions:

- `function createToken(DeployParams calldata params)`

Owner

The Owner role has authority over the following functions:

- `function initialize()`
- `function createTokenForOwner(OwnedDeployParams calldata params)`
- `function updateImplementation(address newImplementation)`
- `function updateCollectionAdministrator(address newCollectionAdministrator)`

EndemicExchange Contract

Owner

The Owner role has authority over the following functions:

- `function __EndemicExchange_init(address _royaltiesProvider, address _paymentManager, address _feeRecipientAddress, address _approvedSettler)`
- `function updateConfiguration(address _royaltiesProvider, address _paymentManager, address _feeRecipientAddress, address _approvedSettler)`

PaymentManager Contract

Owner

The Owner role has authority over the following functions:

- `function __PaymentManager_init(uint256 makerFee, uint256 takerFee)`
- `function updateSupportedPaymentMethod(address paymentMethodAddress, bool isEnabled)`
- `function updatePaymentMethodFees(address paymentMethodAddress, uint256 makerFee, uint256 takerFee)`

User

The User role can interact with the following functions:

- `function getPaymentMethodFees(address paymentMethodAddress)`
- `function isPaymentMethodSupported(address paymentMethodAddress)`
- `function royaltyInfo(uint256, uint256 value)`
- `function bidForDutchAuction(uint8 v, bytes32 r, bytes32 s, DutchAuction calldata auction)`
- `function getCurrentPrice(uint256 startingPrice, uint256 endingPrice, uint256 startingAt, uint256 duration)`
- `function cancelNonce(uint256 nonce)`
- `function finalizeReserveAuction(ReserveAuction calldata auction, ReserveAuction calldata bid)`
- `function acceptNftOffer(uint8 v, bytes32 r, bytes32 s, Offer calldata offer)`
- `function acceptCollectionOffer(uint8 v, bytes32 r, bytes32 s, Offer calldata offer, uint256 tokenId)`
- `function buyFromSale(uint8 v, bytes32 r, bytes32 s, Sale calldata sale)`

RoyaltiesProvider Contract

Owner

The Owner role has authority over the following functions:

- `function __RoyaltiesProvider_init(uint256 royaltiesLimit)`
- `function setRoyaltiesLimit(uint256 newLimit)`

NftOwner

The Owner role has authority over the following functions:

- `function setRoyaltiesForToken(address nftContract, uint256 tokenId, address feeRecipient, uint256 fee)`
- `function setRoyaltiesForCollection(address nftContract, address feeRecipient, uint256 fee)`

User

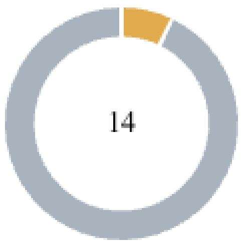
The User role can interact with the following functions:

- `function calculateRoyaltiesAndGetRecipient(address nftContract, uint256 tokenId, uint256 amount)`

Test Deployment

Contract	Explorer
Collection	https://testnet.bscscan.com/address/0x14e285c38e0586217236C51fB4faa14adbFFC818#code
EndemicCollectionFactory	https://testnet.bscscan.com/address/0xEB8516627288ddE8E6c3d55B63004E637e2cD929#code
EndemicExchange	https://testnet.bscscan.com/address/0xa56D376c25C82646c472FDa28031A7586EF3Dbe2#code
PaymentManager	https://testnet.bscscan.com/address/0x7EA7f13579DD3527dA2d3A9d8480A3f7D9867359#code
RoyaltiesProvider	https://testnet.bscscan.com/address/0x09a960f03CB691fB4e748f820EA2D4Db5349fA0C#code

Findings Breakdown



●	Critical	0
●	Medium	1
●	Minor / Informative	13

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	13	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	ZD	Zero Division	Unresolved
●	PIB	Possible Insufficient Balance	Unresolved
●	RSP	Redundant Struct Property	Unresolved
●	VTO	Variable Type Optimization	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	MU	Modifiers Usage	Unresolved
●	SWO	Storage Write Optimization	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	CR	Code Repetition	Unresolved
●	UIC	Unused Imported Contract	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L19	Stable Compiler Version	Unresolved

ZD - Zero Division

Criticality	Medium
Location	mixins/auction/EndemicDutchAuction.sol#L175
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert.

The `duration` variable is provided by an off-chain source. Assuming its value will never be zero might lead to divisions by zero.

```
int256 currentPriceChange = (totalPriceChange *
    int256(secondsPassed)) / int256(duration);
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements.

PIB - Possible Insufficient Balance

Criticality	Minor / Informative
Location	mixins/EndemicExchangeCore.sol#L158
Status	Unresolved

Description

As part of the bidding process, the contract verifies the sufficiency of the buyer's - contract allowance. However, it's crucial to note that the allowance itself may not necessarily reflect the buyer's actual available funds. Therefore, although the allowance check may pass successfully, the buyer's account balance might still be insufficient to complete the transaction, leading to a potential transaction revert.

```
function _requireSufficientErc20Allowance(
    uint256 sufficientAmount,
    address paymentMethodAddress,
    address buyer
) internal view {
    IERC20 ERC20PaymentToken = IERC20(paymentMethodAddress);

    uint256 contractAllowance = ERC20PaymentToken.allowance(
        buyer,
        address(this)
    );
    if (contractAllowance < sufficientAmount) {
        revert UnsufficientCurrencySupplied();
    }
}
```

Recommendation

The team is advised to take these segments into consideration and modify the validation process. In addition to checking the allowance, the contract should also explicitly verify whether the buyer's account balance covers the intended transaction amount. This dual verification ensures that both the allowance and the available balance are adequate for the transaction to proceed smoothly, minimizing the risk of unexpected reversions and enhancing the contract's reliability during the bidding process.

RSP - Redundant Struct Property

Criticality	Minor / Informative
Location	PaymentManager.sol#L17
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 declares the mapping `feesByPaymentMethod`. The current implementation uses the `paymentMethodAddress` property both as the key for the mapping and as a property within the `PaymentMethodFees` struct. As a result, this property is redundant.

```
mapping(address => PaymentMethodFees) public feesByPaymentMethod;
struct PaymentMethodFees {
    address paymentMethodAddress;
    uint256 makerFee;
    uint256 takerFee;
}
```

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.

The team could remove the property from the struct. If it is used as an existence indicator then a boolean data type could be used instead.

VTO - Variable Type Optimization

Criticality	Minor / Informative
Location	RoyaltiesProvider.sol#L15,40 PaymentManager.sol#L11,19,20 mixins/EndemicExchangeCore.sol#L14
Status	Unresolved

Description

The contracts declare certain variables as `uint256`, even though their maximum value is capped at 10,000. By performing a mathematical analysis, it becomes evident that the highest value, 10,000, can be comfortably stored in a `uint16` type, as the logarithm base-2 of 10,000 results in approximately 13.29. As a result, the contracts reserve unnecessary storage space and consume more gas when using these variables.

```
uint256 public royaltyFeeLimit;
uint256 fee;
uint256 internal constant MAX_FEE = 10000;
uint256 makerFee;
uint256 takerFee;
```

Recommendation

To optimize the smart contract and improve resource utilization, it is strongly recommended to review and update the variable types used. Specifically, consider changing variables currently declared as `uint256` to `uint16` wherever applicable, given that the maximum value they store is 10,000. This adjustment aligns the variable types more closely with the actual data requirements, reducing unnecessary gas costs and optimizing storage efficiency.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	mixins/auction/EndemicDutchAuction.sol#L38 mixins/auction/EndemicReserveAuction.sol#L49 mixins/EndemicSale.sol#L50 mixins/EndemicOffer.sol#L50,71
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

The Endemic ecosystem and its contracts contain several functions that rely on off-chain data. Namely, the following functions:

- `bidForDutchAuction` : manages the bidding process using auction data generated off-chain
- `finalizeReserveAuction` : processes the conclusion of a reserve auction and transfers an NFT from the auction's creator to the winning bidder
- `acceptNftOffer` / `acceptCollectionOffer` : enables the acceptance of offers for NFTs
- `buyFromSale` : facilitates the purchase of an NFT from a sale

```
function bidForDutchAuction(
    uint8 v,
    bytes32 r,
    bytes32 s,
    DutchAuction calldata auction
) external payable nonReentrant { ... }
function finalizeReserveAuction(
    ReserveAuction calldata auction,
    ReserveAuction calldata bid
) external onlySupportedERC20Payments(auction.paymentErc20TokenAddress) {
... }
...
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	RoyaltiesProvider.sol#L101,120
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
checkOwner(nftContract)
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

SWO - Storage Write Optimization

Criticality	Minor / Informative
Location	PaymentManager.sol#L85
Status	Unresolved

Description

The current implementation of the `updatePaymentMethodFees` function replaces the entire PaymentMethodFees object when updating fee values. This approach results in unnecessary gas consumption due to object creation and copying.

```
feesByPaymentMethod[paymentMethodAddress] = PaymentMethodFees(
    paymentMethodAddress,
    makerFee,
    takerFee
);
```

Recommendation

The is advised to refactor the `updatePaymentMethodFees` function to directly mutate the existing PaymentMethodFees object by updating the makerFee and takerFee fields. This optimization will lead to reduced gas costs during fee updates and improve the overall efficiency of the contract.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	PaymentManager.sol#L69 mixins/EndemicFundsDistributor.sol#L157 mixins/auction/EndemicReserveAuction.sol#L101 mixins/CollectionRoyalties.sol#L24,48
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 modifies the state of certain variables without checking if their current state is equal to the provided argument. As a result, the contract performs redundant storage writes.

```
supportedPaymentMethods[paymentMethodAddress] = isEnabled
feeRecipientAddress = _feeRecipientAddress
approvedSettler = _approvedSettler
royaltiesRecipient = recipient
```

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.

CR - Code Repetition

Criticality	Minor / Informative
Location	EndemicExchange.sol#L34,35,36 Collection.sol#L122,144
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

The listed functions are included in the `updateConfiguration()` function. As a result, the contract duplicates code.

```
_updateDistributorConfiguration(_feeRecipientAddress);
_updateExchangeConfiguration(_royaltiesProvider, _paymentManager);
_updateApprovedSettler(_approvedSettler);
```

The `mintAndApprove` and `batchMintAndApprove` functions could reuse the `mint` and `batchMint` functions respectively. As a result, the contract duplicates code.

```
function mintAndApprove(
    address recipient,
    string calldata tokenCID,
    address operator,
    uint8 v,
    bytes32 r,
    bytes32 s,
    uint256 nonce
) external onlyOwner {
    // Check if mint approval is required
    if (mintApprovalRequired) {
        // Make sure that mint is approved
        _checkMintApproval(owner(), tokenCID, v, r, s, nonce);
    }

    // Mint token to the recipient
    _mintBase(recipient, tokenCID);

    // Approve operator to access tokens
    setApprovalForAll(operator, true);
}
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

UIC - Unused Imported Contract

Criticality	Minor / Informative
Location	EndemicCollectionFactory.sol#L11
Status	Unresolved

Description

The EndemicCollectionFactory contract imports the Collection contract but does not utilize any of its functions, variables, or features within its codebase. This indicates potential inefficiency and unnecessary complexity in the contract structure.

```
import "./Collection.sol";
```

Recommendation

The team is advised to remove the import statement for the unused contract to streamline the codebase and reduce unnecessary complexity. This will improve the contract's readability, maintainability, and potentially reduce gas costs associated with deploying and interacting with the contract on the blockchain.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	mixins/auction/EndemicReserveAuction.sol#L101 mixins/CollectionRoyalties.sol#L24,48 mixins/EndemicFundsDistributor.sol#L157
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The provided addresses should not be the zero address.

```
royaltiesRecipient = recipient
approvedSettler = _approvedSettler
feeRecipientAddress = _feeRecipientAddress
```

The `makerFee` and `takerFee` variables should be less than max fee which is 10,000.

```
feesByPaymentMethod[ZERO_ADDRESS] = PaymentMethodFees(
    ZERO_ADDRESS,
    makerFee,
    takerFee
);
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	RoyaltiesProvider.sol#L44 PaymentManager.sol#L23 mixins/EndemicSale.sol#L167 mixins/EndemicOffer.sol#L167 mixins/EndemicNonceManager.sol#L26 mixins/EndemicFundsDistributor.sol#L163 mixins/EndemicExchangeCore.sol#L191 mixins/EndemicEIP712.sol#L32 mixins/CollectionRoyalties.sol#L18 mixins/auction/EndemicDutchAuction.sol#L218 EndemicExchange.sol#L25,26,27,28,29,46,47,48,49 EndemicCollectionFactory.sol#L149
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 __RoyaltiesProvider_init(uint256 royaltiesLimit)
        external
        initializer
    {
        __Context_init_unchained();
        __Ownable_init_unchained();

        setRoyaltiesLimit(royaltiesLimit);
    }

...
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	mixins/EndemicExchangeCore.sol#L15 EndemicCollectionFactory.sol#L149
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
uint256 internal constant MIN_PRICE = 0.0001 ether
uint256[1000] private __gap
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	mixins/ERC721Base.sol#L2
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 ^0.8.4;
```

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**
RoyaltiesProvider	Implementation	OwnableUpgradeable		
	__RoyaltiesProvider_init	External	✓	initializer
	calculateRoyaltiesAndGetRecipient	External		-
	setRoyaltiesForToken	External	✓	-
	setRoyaltiesForCollection	External	✓	-
	setRoyaltiesLimit	Public	✓	onlyOwner
	checkOwner	Internal		
	calculateFeeForAmount	Internal		
PaymentManager	Implementation	OwnableUpgradeable		
	__PaymentManager_init	External	✓	initializer
	getPaymentMethodFees	External		-
	isPaymentMethodSupported	External		-
	updateSupportedPaymentMethod	External	✓	onlyOwner
	updatePaymentMethodFees	External	✓	onlyOwner

EndemicExchange	Implementation	EndemicDutchAuction, EndemicReserveAuction, EndemicOffer, EndemicSale, OwnableUpgradeable		
	__EndemicExchange_init	External	✓	initializer
	updateConfiguration	External	✓	onlyOwner
EndemicCollectionFactory	Implementation	Initializable, AccessControlUpgradeable		
	initialize	External	✓	initializer
	createToken	External	✓	onlyRole
	createTokenForOwner	External	✓	onlyRole
	updateImplementation	External	✓	onlyContract onlyRole
	updateCollectionAdministrator	External	✓	onlyRole
	_deployContract	Internal	✓	
Collection	Implementation	CollectionFactory, Initializable, ERC721Upgradeable, MintApproval, ERC721Base, CollectionRoyalties		
		Public	✓	CollectionFactory
	initialize	External	✓	onlyCollectionFactory initializer

	mint	External	✓	onlyOwner
	batchMint	External	✓	onlyOwner
	mintAndApprove	External	✓	onlyOwner
	batchMintAndApprove	External	✓	onlyOwner
	tokenURI	Public		-
	setRoyalties	External	✓	onlyOwner
	supportsInterface	Public		-
	_mintBase	Internal	✓	
	_batchMintBase	Internal	✓	
	_burn	Internal	✓	
	_baseURI	Internal		
MintApproval	Implementation	EIP712Upgradeable, AdministratedUpgradable		
	toggleMintApproval	External	✓	onlyAdministrator
	updateMintApprover	External	✓	onlyAdministrator
	_checkMintApproval	Internal	✓	
	_checkBatchMintApproval	Internal	✓	
	_prepareMessage	Private		
	_prepareBatchMessage	Private		

EndemicSale	Implementation	ReentrancyGuardUpgradeable, EndemicFundsDistributor, EndemicExchangeCore, EndemicEIP712, EndemicNonceManager		
	buyFromSale	External	Payable	nonReentrant
	_finalizeSale	Internal	✓	
	_verifySignature	Internal		
EndemicOffer	Implementation	ReentrancyGuardUpgradeable, EndemicFundsDistributor, EndemicExchangeCore, EndemicEIP712, EndemicNonceManager		
	acceptNftOffer	External	✓	nonReentrant onlySupportedERC20Payments
	acceptCollectionOffer	External	✓	nonReentrant onlySupportedERC20Payments
	_acceptOffer	Internal	✓	
	_verifySignature	Internal		
EndemicNonceManager	Implementation			
	cancelNonce	External	✓	-
	_invalidateNonce	Internal	✓	

EndemicFundsDistributor	Implementation			
	_distributeFunds	Internal	✓	
	_distributeEtherFunds	Internal	✓	
	_distributeErc20Funds	Internal	✓	
	_transferEtherFees	Internal	✓	
	_transferErc20Fees	Internal	✓	
	_transferEtherRoyalties	Internal	✓	
	_transferErc20Royalties	Internal	✓	
	_transferEtherFunds	Internal	✓	
	_transferErc20Funds	Internal	✓	
	_updateDistributorConfiguration	Internal	✓	
EndemicExchangeCore	Implementation			
	_calculateFees	Internal		
	_calculateOfferFees	Internal		
	_calculateTakerCut	Internal		
	_calculateCut	Internal		
	_requireSupportedPaymentMethod	Internal		
	_requireSufficientCurrencySupplied	Internal		
	_requireSufficientEtherSupplied	Internal		
	_requireSufficientErc20Allowance	Internal		
	_updateExchangeConfiguration	Internal	✓	

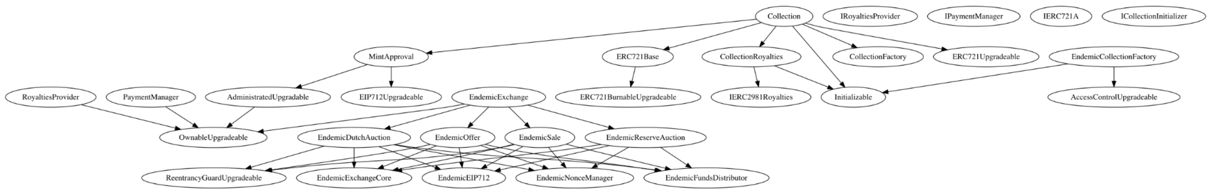
EndemicEIP712	Implementation			
	_buildDomainSeparator	Internal		
ERC721Base	Implementation	ERC721BurnableUpgradeable		
	totalSupply	External		-
	_burn	Internal	✓	
CollectionRoyalties	Implementation	Initializable, IERC2981Royalties		
	__CollectionRoyalties_init	Internal	✓	onlyInitializing
	royaltyInfo	External		-
	supportsInterface	Public		-
	_setRoyalties	Internal	✓	
CollectionFactory	Implementation			
		Public	✓	-
EndemicReserveAuction	Implementation	EndemicFundsDistributor, EndemicExchangeCore, EndemicEIP712, EndemicNonceManager		
	finalizeReserveAuction	External	✓	onlySupportedERC20Payments

	_updateApprovedSettler	Internal	✓	
	_calculateAuctionFees	Internal		
	_verifySignature	Internal		
EndemicDutchAuction	Implementation	ReentrancyGuardUpgradeable, EndemicFundsDistributor, EndemicExchangeCore, EndemicEIP712, EndemicNonceManager		
	bidForDutchAuction	External	Payable	nonReentrant
	getCurrentPrice	External		-
	_determinePriceByPaymentMethod	Internal		
	_calculateCurrentPrice	Internal		
	_verifySignature	Internal		
IRoyaltiesProvider	Interface			
	calculateRoyaltiesAndGetRecipient	External		-
IPaymentManager	Interface			
	getPaymentMethodFees	External		-
	isPaymentMethodSupported	External		-
	updateSupportedPaymentMethod	External	✓	-
	updatePaymentMethodFees	External	✓	-

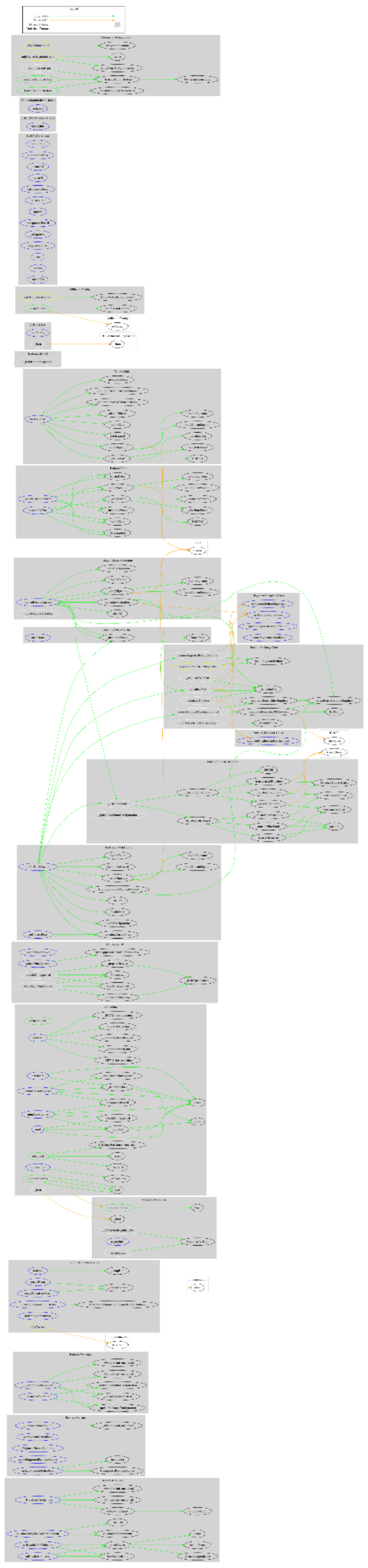
IERC721A	Interface			
	totalSupply	External		-
	supportsInterface	External		-
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	Payable	-
	safeTransferFrom	External	Payable	-
	transferFrom	External	Payable	-
	approve	External	Payable	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
	name	External		-
	symbol	External		-
	tokenURI	External		-
IERC2981Royalties	Interface			
	royaltyInfo	External		-
ICollectionInitializer	Interface			
	initialize	External	✓	-

AdministratedUpgradable	Implementation	OwnableUpgradeable		
	__Administrated_init	Internal	✓	onlyInitializing
	renounceAdministration	Public	✓	onlyAdministrator
	transferAdministration	Public	✓	onlyOwnerOrAdministrator
	_transferAdministration	Internal	✓	

Inheritance Graph



Flow Graph



Summary

Endemic contract implements an NFT, utility, and bet mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

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