Blaize.Security

August 31st, 2023 / V. 1.0

CRYPTOPIA
SMART CONTRACT AUDIT

TABLE OF CONTENTS

AUDIT RATING

SCORE **9.8**/10

The scope of the project includes Cryptopia:

contracts\source\tokens\ERC721\CryptopiaEarlyAccessShip
CryptopiaEarlyAccessShipTokenFactory.sol
CryptopiaEarlyAccessShipToken.sol
contracts\source\tokens
ERC721\CryptopiaERC721.sol
contracts\source\common\meta_transactions
ContentMixin.sol
EIP712Base.sol
NativeMetaTransaction.sol
contracts\source\tokens\ERC20\retriever
TokenRetriever.sol

Repository:

https://github.com/cryptopia-com/cryptopia-early-access-token

Branch: main

Initial commit:

- cca9003a2ffa1bfa07e851a1882b8b9f1b6c2e80

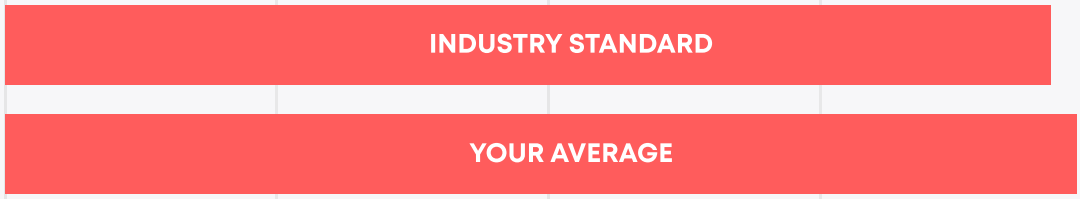
Final commit:

- dc3e1065962834d85c51841258d7193c8156bd31

TECHNICAL SUMMARY

During the audit, we examined the security of smart contracts for the Cryptopia protocol. Our task was to find and describe any security issues in the smart contracts of the platform. This report presents the findings of the security audit of the **Cryptopia** smart contracts conducted between **August 22nd, 2023** and **August 31st, 2023.**

Testable code



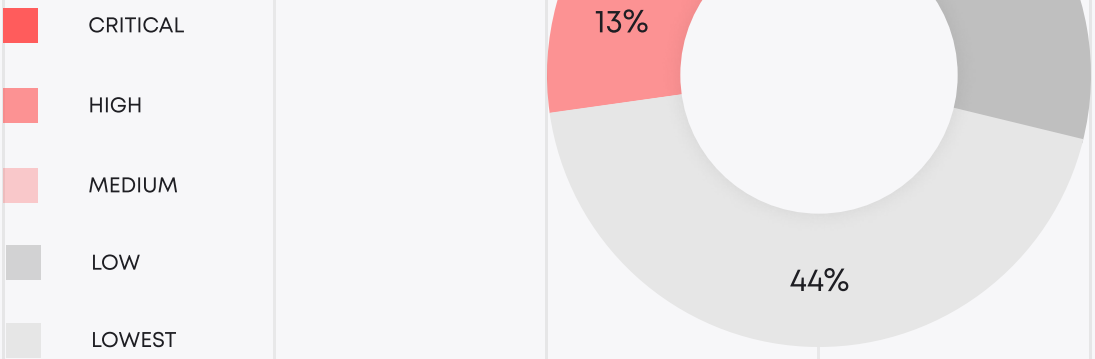
After the testing stage, code is 98% testable which is above the industry standard.

The audit scope includes all tests and scripts, documentation, and requirements presented by the **Cryptopia** team. The coverage is calculated based on the set of Hardhat framework tests and scripts from additional testing strategies, and includes testable code from manual and exploratory rounds.

However, to ensure the security of the contract, the **Blaize.Security** team suggests that the **Cryptopia** team follow post-audit steps:

1. launch **active protection** over the deployed contracts to have a system of early detection and alerts for malicious activity. We recommend the AI-powered threat prevention platform **VigiLens**, by the **CyVers** team.
2. launch a **bug bounty program** to encourage further active analysis of the smart contracts.

THE GRAPH OF VULNERABILITIES DISTRIBUTION:



The table below shows the number of the detected issues and their severity. A total of 18 problems were found. All of the issues were fixed or verified by the Cryptopia team.

	FOUND	FIXED/VERIFIED
Critical	0	0
High	2	2
Medium	3	3
Low	5	5
Lowest	8	5

SEVERITY DEFINITION

Critical

The system contains several issues ranked as very serious and dangerous for users and the secure work of the system. Requires immediate fixes and a further check.

High

The system contains a couple of serious issues, which lead to unreliable work of the system and migh cause a huge data or financial leak. Requires immediate fixes and a further check.

Medium

The system contains issues that may lead to medium financial loss or users' private information leak. Requires immediate fixes and a further check.

Low

The system contains several risks ranked as relatively small with the low impact on the users' information and financial security. Requires fixes.

Lowest

The system does not contain any issues critical to the secure work of the system, yet is relevant for best practices

AUDITING STRATEGY AND TECHNIQUES APPLIED/PROCEDURE

Blaize.Security auditors start the audit by developing an **auditing strategy** - an individual plan where the team plans methods, techniques, approaches for the audited components. That includes a list of activities:

Manual audit stage

- Manual line-by-line code by at least 2 security auditors with crosschecks and validation from the security lead;
- Protocol decomposition and components analysis with building an interaction scheme, depicting internal flows between the components and sequence diagrams;
- Business logic inspection for potential loopholes, deadlocks, backdoors;
- Math operations and calculations analysis, formula modeling;
- Access control review, roles structure, analysis of user and admin capabilities and behavior;
- Review of dependencies, 3rd parties, and integrations;
- Review with automated tools and static analysis;
- Vulnerabilities analysis against several checklists, including internal Blaize.Security checklist;
- Storage usage review;
- Gas (or tx weight or cross-contract calls or another analog) optimization;
- Code quality, documentation, and consistency review.

For advanced components:

- Cryptographical elements and keys storage/usage audit (if applicable);
- Review against OWASP recommendations (if applicable);
- Blockchain interacting components and transactions flow (if applicable);
- Review against CCSSA (C4) checklist and recommendations (if applicable);

Testing stage:

- Development of edge cases based on manual stage results for false positives validation;
- Integration tests for checking connections with 3rd parties;
- Manual exploratory tests over the locally deployed protocol;
- Checking the existing set of tests and performing additional unit testing;
- Fuzzy and mutation tests (by request or necessity);
- End-to-end testing of complex systems;

In case of any issues found during audit activities, the team provides detailed recommendations for all findings.

EXECUTIVE SUMMARY

Blaize Security team has conducted an audit for the Cryptopia protocol. The protocol consists of an ERC721 NFT collection, a factory for minting new tokens, and common contracts for EIP-217 signatures and meta transactions. Tokens represent ships with different characteristics and stats. Contracts allow users to mint new ships by paying a specific amount of ETH. Ships have different rarity and stats. Rarity affects the number of daily allocations the ship's owner receives in an ERC-20 token.

The audit aimed to analyze the smart contracts against the list of common vulnerabilities and auditors' internal checklist, validate the safety of funds, including ETH, NFTs, and ERC-20 tokens, and check the correctness of the minting process and allocation distributions. Additionally, auditors have reviewed the correctness of meta transactions and EIP-712 signatures and the implementation of ERC-712 itself.

Auditors discovered no critical issues. Two high-risk, three medium-risk, five low-risk, and several lowest-risk problems were detected. High-risk issues were connected to the obsolete ETH transfer method and the ability of anyone to withdraw tokens on the TokenReceiver implementation. Medium issues were regarding the absence of safeERC20, the ability to set mint timestamps in the future, and the usage of an unsafe mint function. Low- and lowest-risk issues focused on validating business logic and minor issues that didn't have a critical effect on the protocol's security.

However, there are several unresolved lowest-risk issues worth noting:

- The lowest-3 issue described the lack of ERC-20 allocations transfer to the protocol. Thus, the contract can't verify if the admin has provided sufficient ERC-20 tokens to be allocated to the users. Though the Cryptopia team has verified that they will transfer the necessary amount of tokens in advance, we still recommend adding corresponding validations in the contract or the script.

- The lowest-7 issue is connected to the upgradability of the ERC721 token. The issue was raised since tokens are usually implemented as a non-upgradable smart contract to ensure the safety of users' funds and metadata of the token. The Cryptopia team has ensured that the upgradability will be turned off later by renouncing the ownership of Proxy Admin. Nevertheless, the contract remains upgradable until then. This is why the Cryptopia team should be extra cautious with any possible upgrades of the CryptopiaEarlyAccessShipToken.sol.
- Lowest-8 issue. This issue describes the insecure randomness generation based on the on-chain data, which validators can predict or manipulate. Though the Cryptopia team left comments in the code assuring that such a level of randomness is enough for the protocol, auditors still raised the issue since the randomness affects the rarity of the ship, which, in turn, affects the daily allocations the ship will bring the owner. The Cryptopia team has assured that the protocol will operate correctly with the current implementation of the random generator. Although the team verified the issue, it was still marked as unresolved, and the contract hasn't passed the test for Entropy Illusion.

Blaize Security team has tested all the smart contracts, including the additional scenarios for minting and upgrading the tokens, allocation distribution, and meta transactions execution. During the testing of allocations and discussion with the Cryptopia team, we have verified that the correct flow of allocation claiming is as follows: the user is able to claim the whole allocation in one transaction after Monday, 1 September 2025, 00:00:00. It should also be noted that the NFT allows OpenSea's authentication to access tokens on behalf of users to allow users to use their OpenSea proxy accounts for gasless transactions.

The overall security of the protocol is high enough. Contracts are well-written and tested by both the Cryptopia and Blaize Security teams. The contracts' settings seem to be set correctly during deployment, though we recommend the team double-check for the following functions before setting them:

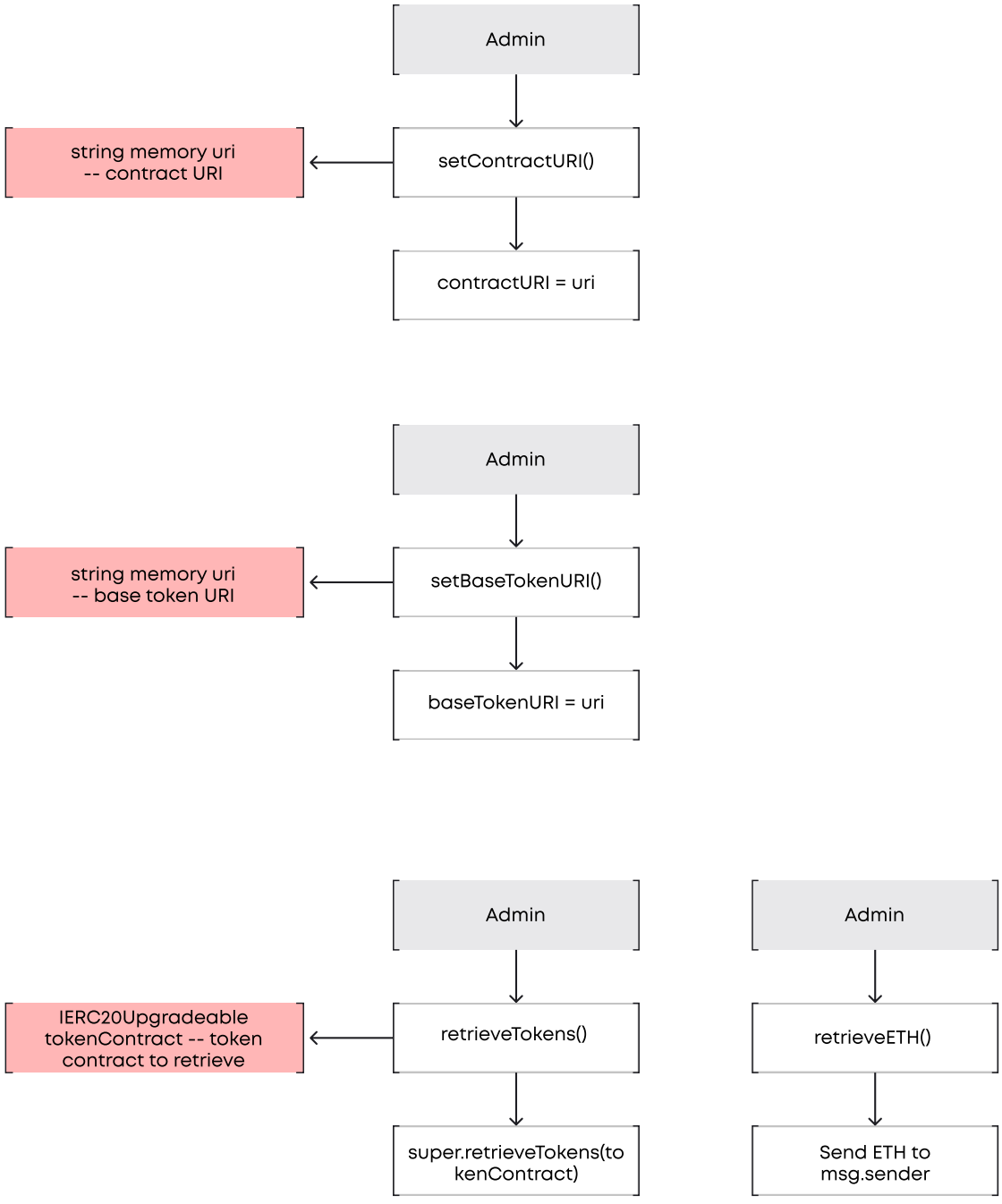
- setLegacyMintedAt()
- setReferrer()
- setBeneficiary()
- setSkins()

	RATING
Security	9.8
Logic optimization	9.7
Code quality	10
Test coverage	9.9
Total	9.8

CRYPTOPIA SCHEME

CryptopiaERC721.sol

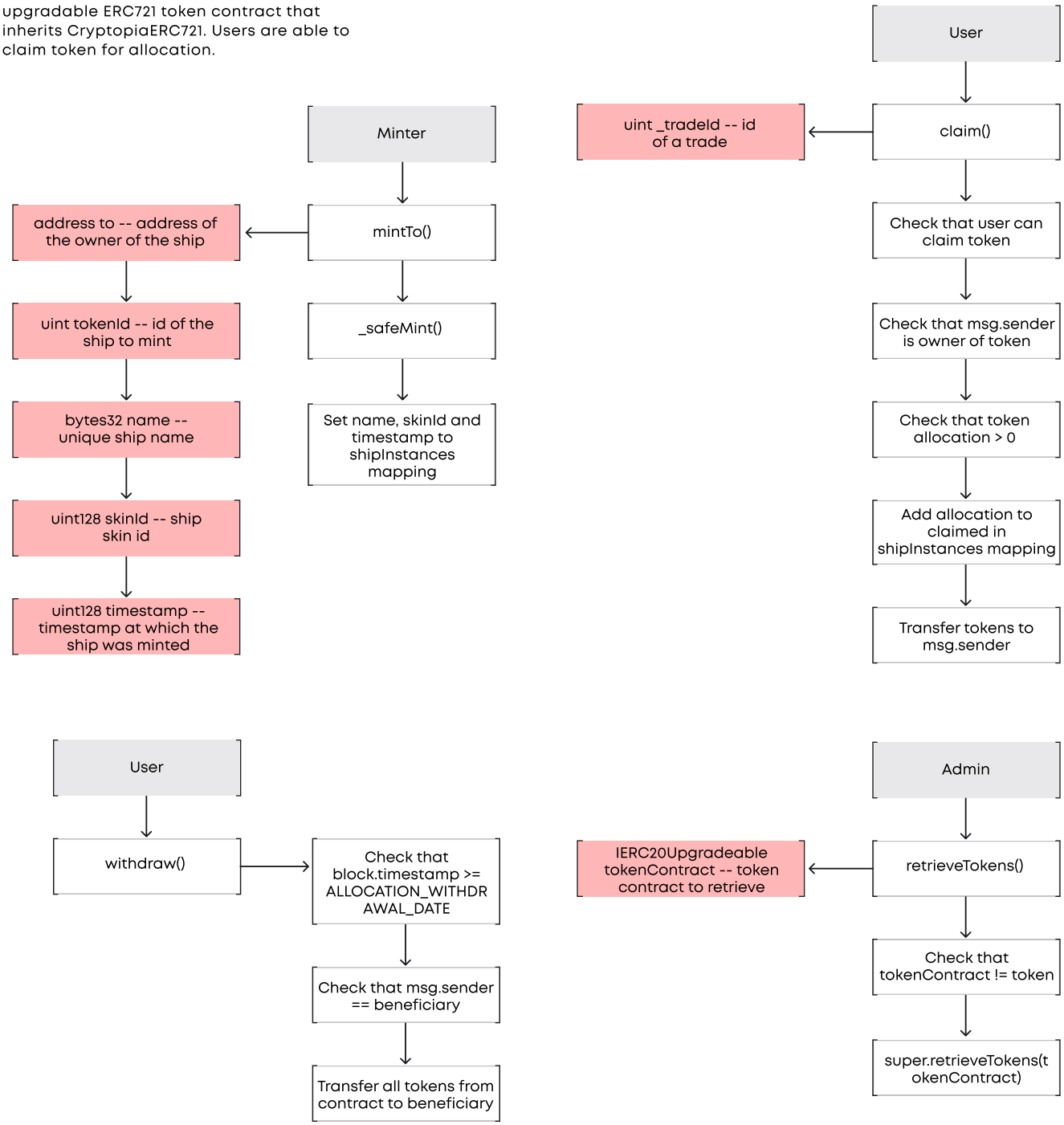
CryptopiaERC721 is a abstract ERC721 token contract that extends Openzeppelin ERC721 contract.



CRYPTOPIA SCHEME

CryptopiaEarlyAccessShipToken.sol

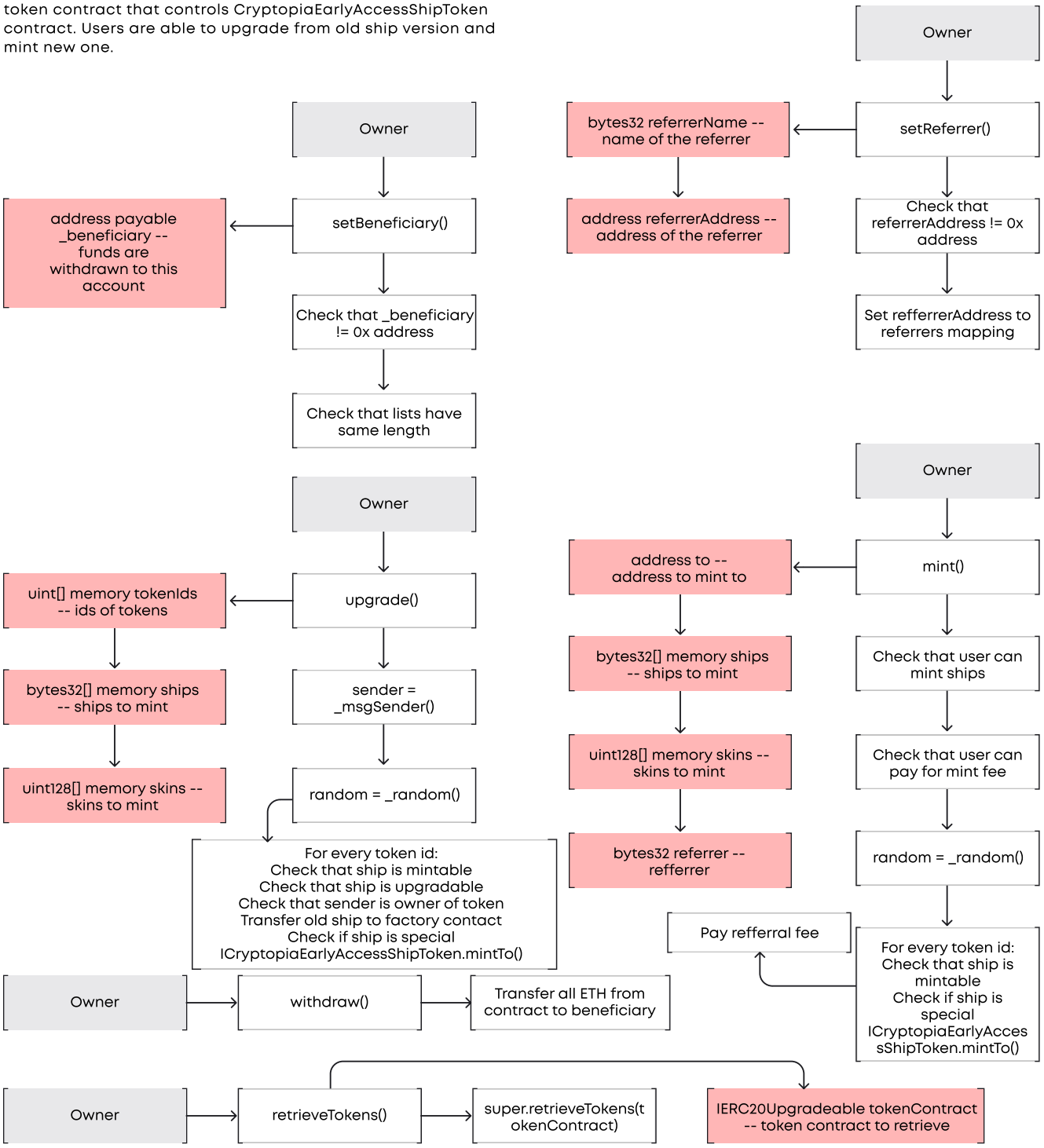
CryptopiaEarlyAccessShipToken is a upgradable ERC721 token contract that inherits CryptopiaERC721. Users are able to claim token for allocation.



CRYPTOPIA SCHEME

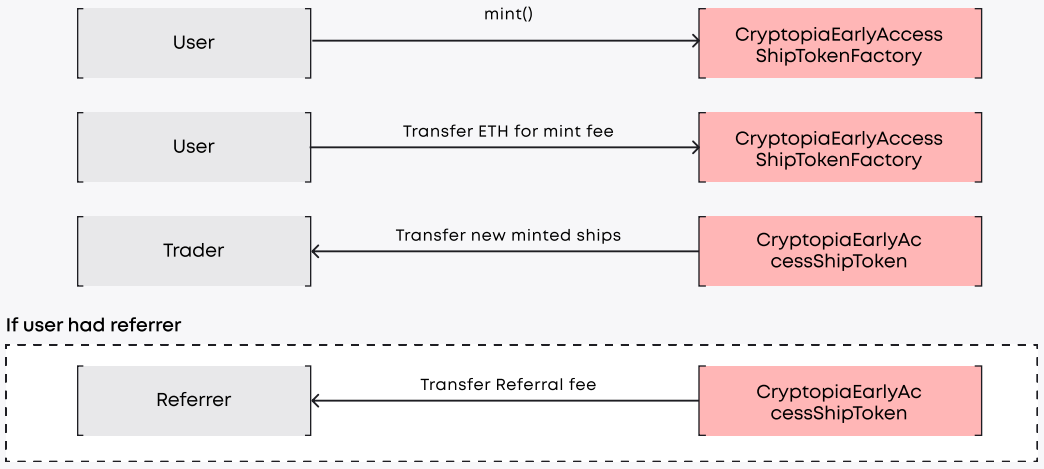
CryptopiaEarlyAccessShipTokenFactory.sol

CryptopiaEarlyAccessShipTokenFactory is a factory ERC721 token contract that controls CryptopiaEarlyAccessShipToken contract. Users are able to upgrade from old ship version and mint new one.

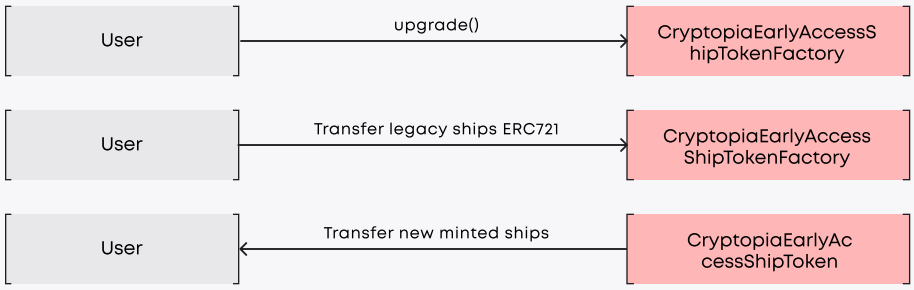


CRYPTOPIA SCHEME

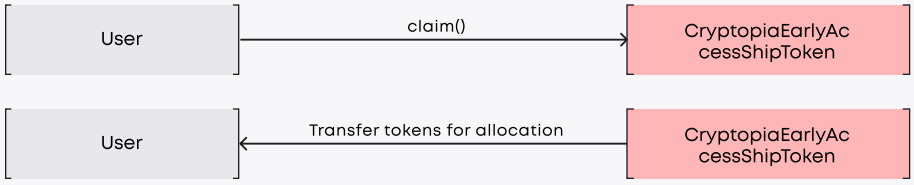
Mint ships



If user has legacy ships -> upgrade to new ships



Claim tokens for ships allocation



ROLES AND RESPONSIBILITIES

CryptopiaEarlyAccessShipTokenFactory

1. Owner (CryptopiaEarlyAccessShipTokenFactory.sol)

Responsibilities:

- Correct deployment of the smart contract.

The contract owner has special privileges, including setting the beneficiary address, setting referrers, and withdrawing funds. Owner can also upgrade legacy ships and manage various aspects of the contract.

Possibilities:

- Can call methods: “withdraw”, “setBeneficiary”, “setReferrer”, “setLegacyMintedAt”.
- Has access to change legacy mintedAt timestamps for specific tokens.
- Can transfer contract balance.

2. Users (CryptopiaEarlyAccessShipTokenFactory.sol)

Responsibilities:

- Correct interaction with contract by methods: mint & upgrade.

The mint() function has permitted mint of the new ship tokens. The basic parameter for ship minting is the ship name (ships) and corresponding skin information (skins) that they would like to mint.
Users can upgrade ship tokens by providing token IDs of the legacy ships they want to upgrade, along with the new ship names (ships) and skin information (skins).
Before the minting, the contract burns old tokens, and only after it mints new ship tokens with the specified properties.
Users provide a referral by bytes32 as a potential rewards receiver.

Possibilities:

- Can call methods: “mint”, “upgrade.
- Can mint new ship token.
- Can upgrade existing tokens in specific manner.

3. Beneficiary (CryptopiaEarlyAccessShipTokenFactory.sol)

Responsibilities:

- Receiver of the contract balance

Possibilities:

- Receive a contract balance after owner call function withdraw

4. Referrers (CryptopiaEarlyAccessShipTokenFactory.sol)

Responsibilities:

- Receiver of the rewards according to contract condition.

Possibilities:

Individuals or addresses acting as referrers can earn referral rewards if users provide their referral code when minting tokens.

CryptopiaEarlyAccessShipToken

1. MINTER_ROLE (CryptopiaEarlyAccessShipToken.sol)

Responsibilities:

Minter controls the minting process for new tokens within that contract.

Possibilities:

- Can call methods: “mintTo”.

Abstract contract **CryptopiaERC721** is ICryptopiaERC721, ERC721EnumerableUpgradeable, ContextMixin, NativeMetaTransaction, AccessControlUpgradeable, TokenRetriever

1. ADMIN_ROLE (CryptopiaERC721.sol)

Responsibilities:

- Administrative actions and management of the contract.
- Grant and revoke roles.
- Set Up a contract URI and base token URI.
- Retrieve tokens or ETH accidentally sent to the contract.

Possibilities:

- Can call methods: “setContractURI”, “setBaseTokenURI”, “retrieveTokens”.

The holder of the ADMIN_ROLE can perform various administrative functions to manage the contract's metadata and access control. This role is responsible for maintaining the overall functionality of the contract.

2. DEFAULT_ADMIN_ROLE (CryptopiaERC721.sol)

Responsibilities:

- General administrative actions and emergency recovery.
- Similar to ADMIN_ROLE but with broader access.

Possibilities:

The holder of the DEFAULT_ADMIN_ROLE has similar administrative capabilities to the ADMIN_ROLE. This role is often used for general contract administration and can also be used for emergency scenarios.

Abstract contract ContextMixin

1. Sender (ContextMixin.sol)

Responsibilities:

- to return address of caller

Possibilities:

- returns payable address

EIP712Base is Initializable
Contains no specific roles and responsibilities

NativeMetaTransaction is EIP712Base

1. Caller (NativeMetaTransaction.sol)
Roles are not explicitly defined.
Responsibilities:
- A caller can call public method “executeMetaTransaction”

“executeMetaTransaction”: This function is used to execute a meta-transaction. It takes in the user's address, the function signature to be executed, and the signature components (R, S, and V) of the signer's.

TokenRetriever is ITokenRetriever
Contains no specific roles and responsibilities

DEPLOYMENT

Contracts list for deploy

WhitelistContract a.k.a "Whitelist";
ShipTokenContract a.k.a "CryptopiaEarlyAccessShipToken";
ShipTokenFactoryContract a.k.a. "CryptopiaEarlyAccessShipTokenFactory";

WhitelistContract

The deployment script seems correct and follows the standard procedure for deploying smart contracts using the Truffle deployment plugin. Here's the breakdown of the script:

1. It imports the necessary modules and gets the named accounts for `deployer` and `configs`.

2. It retrieves the address of the previously deployed `opensea` contract.

3. Parameter config.opensea is an address settled as “0xa5409ec958c83c3f309868babaca7c86dcb077c1”

At etherscan this address is WyvernProxyRegistry: https://etherscan.io/address/0xa5409ec958c83c3f309868babaca7c86dcb077c1#code

ShipTokenContract

The deployment script seems correct and follows the standard procedure for deploying smart contracts using the Truffle deployment plugin. Here's the breakdown of the script:

1. It imports the necessary modules and gets the named accounts for `deployer` and `configs`.

2. Parameter of:
   - config.token = 0xabdd22dfe5db2be20262050523470b650e91f246

At ether scan address is CryptopiaToken https://etherscan.io/address/0xabdd22dfe5db2be20262050523470b650e91f246#code
   - config.beneficiary = 0xdaA304c435b40468da4BA5B8EC89365C66Cc4501
   - whitelistInstance.address - contract previously deployed
   - config.contractURI = 'https://api.ethereum.cryptopia.com/ERC721/CryptopiaEarlyAccessShipToken'
   - config.baseTokenURI = 'https://api.ethereum.cryptopia.com/ERC721/CryptopiaEarlyAccessShipToken/'

ShipTokenFactoryContract:
The deployment script seems correct and follows the standard procedure for deploying smart contracts using the Truffle deployment plugin. Here's the breakdown of the script:

1. It imports the necessary modules and gets the named accounts for `deployer` and `configs`.

2. Parameter of:
- shipTokenInstance.address = contract deployer previously
- config.legacyShipToken = 0xd14Cb56763EF92a17d1d26303aCB35683aC467d8

At etherScan CryptopiaEarlyAccessToken https://etherscan.io/address/0xd14Cb56763EF92a17d1d26303aCB35683aC467d8#code
- config.beneficiary = 0xdaA304c435b40468da4BA5B8EC89365C66Cc4501

At etherscan address https://etherscan.io/address/0xdaA304c435b40468da4BA5B8EC89365C66Cc4501

ASSETS AND SETTINGS

Assets (Cryptopia)

Contract CryptopiaEarlyAccessShipToken stores:

NFT:

- As ERC721 token stores metadata of each NFT. Metadata contains two types Ship & ShipInstance.
- Only address with MINTER_ROLE can mint new NFT

token:

- As ERC20 token contract allows NFT owner can claim this token as allocation
- Address settled as beneficiary can withdraw full contract balance in tokens.

Contract CryptopiaEarlyAccessShipTokenFactory stores:

Native currency:

- as payment for mint NFTs
- address settled as beneficiary can fully withdraw contract balance

NFT:

- as ERC721 token, contract stores metadata about each NFT
- Only NFT owner can upgrade token or make any other action with own NFT

Settings (CryptopiaEarlyAccessShipToken):

1. **token**: This setting determines the token that will be used for allocation of a ship ERC721. It's important to set this to the correct token address. If set incorrectly, it could potentially harm protocol.
2. **beneficiary**: This setting determines the address that will be able to withdraw the allocation token from the contract. It is important to secure and control this address, as it has access to the full balance of the token.

Settings (CryptopiaEarlyAccessShipTokenFactory):
1. **shipToken**: This setting determines the CryptopiaEarlyAccessShipToken contract that will be used to mint new ERC721 tokens. Incorrect address set into this settings, will break the contract logic.

2. **legacyShipToken**: This setting determines the address of an old ship token contract. If set address incorrectly, users would not be able to upgrade to new ship contract.

3. **beneficiary**: This setting determines the address that will be able to withdraw the ETH from the contract. It is important to secure and control this address, as it has access to the full balance of the ETH on the contract.

Meta Transaction description
Meta transaction is a transaction, for which user will not pay gas to send.
It works in the following way:
- A user on frontend invokes the upgrade() function from CryptopiaEarlyAccessShipTokenFactory
- Pop-up comes from wallet to sign message that will be used to send transaction.
- After the user signs the message, the backend executes a transaction with a separate wallet address as the user would invoke it.
- User receives new NFTs without paying any gas.

COMPLETE ANALYSIS

HIGH-1	✔ Resolved

Obsolete eth transfer method

CryptopiaEarlyAccessShip\CryptopiaEarlyAccessShipTokenFactory.sol, mint(), withdraw()
Function utilizes .transfer() method for ETH transfer. The function sends all ETH to the owner, which may be set to the multisig account. In this case, transfer() may revert since it forwards not enough gas. Therefore funds may be stuck on the contract.
Since .transfer() and .send() methods became obsolete after the Istanbul Ethereum update, it is recommended to use .call() for funds transfer with the mandatory check of the .call result.

Recommendation:

Use .call() for ETH transfer.

Post audit.

.call() is used now.

HIGH-2	✔ Resolved

Anyone can withdraw tokens from the contract.

TokenRetriever.sol: retrieveTokens()Function has no access control. Thus, anyone can invoke it and withdraw all tokens from the contract. As this contract is only used in other contracts and not by itself, it should be marked as abstract as a function overridden in contracts that inherit this function.

Recommendation:

Add access control or role check to the function.

Post audit.

CryptopiaERC721 inherits retrieveTokens() function where access control is implemented.

MEDIUM-1	✔ Resolved

SafeTrasfer should be used

TokenRetriever.sol, retrieveToken()
functionCryptopiaEarlyAccessShipToken.sol, claim(), withdraw()
functions
The function uses .transfer() method for ERC20 tokens. It works for most tokens, though it will not work for tokens with incorrect ERC20 interface - in case there is no transfer method return value. Though the issue is marked as Medium, since USDT and several bridged tokens have incorrect ERC20 interface, so even stablecoins may be stuck in the contract.

Note: the same applies to CryptopiaEarlyAccessShipToken.sol, claim(). Though this place requires verification if the token used and may not require safetransfer

Recommendation:
Use SafeERC20 library OR list tokens which will be used in the function. It is better to use SafeTransfer in case if the function has general meaning and should work with any ERC20

Post audit.
Cryptopia team added SafeERC20 to TokenRetriever contract.
Cryptopia team verified that CryptopiaEarlyAccessShipToken contract will use token created by Cryptopia so it will be implemented correctly.

MEDIUM-2	✔ Resolved

Timestamp in the future will temporarily break the view logic

CryptopiaEarlyAccessTokenShip.sol, mintTo(), _getTotalAllocation()
mintTo() function allows the setting of a custom timestamp for the ship instance. In case of the timestamp set in the future, function _getTotalAllocation() will revert. Thus, all functionality bound to this function and such ship instance will revert.

Recommendation:
Add a condition to return 0 for the total allocation in case of the ship instance timestamp in the future.

Post audit.
It is now validated in _getTotalAllocation() that timestamp of ship instance is not greater than the block.timestamp. Cryptopia team has also clarified that legacy data will be minted using the script setLegacyMintData.ts and there won't be any timestamp set in the future.

MEDIUM-3	✔ Resolved

_safeMint() should be used

CryptopiaEarlyAccessShipToken.sol, mintTo() function
Function uses _mint() method for ERC721. It is recommended to use _safeMint() as it ensures that the recipient is either an EOA or implements IERC721Receiver.

Recommendation:
Change _mint() to _safeMint().

Post audit.
safeMint is now used.

LOW-1	✔ Resolved

SafeMath library should be omitted

NativeMetaTransactions.sol
Since solc 0.8.x has built-in overflow and underflow restrictions, SafeMath library should be omitted, as now it adds extra gas spendings for the contract.

Recommendation:

remove SafeMath library.

Post-audit.
SafeMath was removed.

LOW-2	✔ Resolved

Missing explicit visibility

NativeMetaTransactions.sol, nonce mapping
CryptopiaEarlyAccessShipTokenFactory.sol: lines 34-36
CryptopiaEarlyAccessShipToken.sol: lines 47-48
Missing explicit visibility violates the standard security checklist for Solidity contracts. Missing visibility makes contract code misleading for a user and may signalize missing edge cases for values reading.

Recommendation:

Provide explicit visibility.

Post-audit.
Added visibility to given variables.

LOW-3	✔ Verified

Existence of skin is not checked.

CryptopiaEarlyAccessShipToken.sol: mintTo().
The parameter `skinId` is not checked to be set in mapping `skins`. Thus, the minter can pass a non-existing skin. Since tokens are minted through the factory and the user's parameters for minting are passed, cases, where non-existing skin is passed, may occur.

Recommendation:

Validate `skinId` to be set **OR** verify that such ability to set non-existing skin is allowed (For example, in case if skin should be set later).

Post-audit.
According to the the Cryptopia team, this is implemented by design.

LOW-4	✔ Verified

Admin is able to update skins at any time.

CryptopiaEarlyAccessShipToken.sol: setSkins().
Admin is able to update skins' names for already set skins. Skins are part of shils' metadata, which is usually immutable on most implementations of NFTs. Thus, allowing the admin to update it at any time may be unsafe for users who have paid funds for the NFT.

Recommendation:

Remove the ability of updating already existing skins **OR** verify the necessity of update.

Post-audit.
According to the Cryptopia team, this is implemented by design. The team has also assured that once this function is unnecessary, admin role will be revoked so that skins remain unchanged.

LOW-5	✔ Verified

Minting of the tokens via Factory may be blocked.

CryptopiaEarlyAccessShipTokenFactory.sol: mint(), line 228.
The ID of the NFT to mint is determined, starting with 0 and incrementing by 1. Thus, if an NFT with a certain ID is already minted, minting via Factory will be blocked as it is impossible to choose an available ID. The issue can occur as CryptopiaEarlyAccessShipToken.sol allows multiple users with minter roles. Thus, if other minter mint NFT, Factory can no longer mint tokens properly. The issue is marked as low since only the admin can grant minter roles to users.

Recommendation:

Restrict the minter role, so that only the factory can have it.

Post-audit.

The Cryptopia team has verified that the minter role will be granted only via script 2_deploy_contracts.ts where minter role is granted only to the factory. However, it is still possible to grant minter role later to other address. Thus, we recommend Cryptopia team to be careful with the role and ensure that the functionality of factory is not prevented by other minters.

LOWEST-1	✔ Verified

Any method call allowed

Metatransactions.sol
CryptopiaERC721.sol
CryptopiaEarlyAccessTokenShip.sol

Metatransactions contract allows calls for any method of the contract (on behalf of the contract itself). In general, it's normal to have such functionality, though it may open a risky vulnerability since there are no restrictions for the caller and called methods. Therefore, the auditors' team highly recommends re-checking the admin-controlled methods of target contracts so that the admin role (or any other significant role) will not be assigned to the contract itself. Also, it is recommended to restrict allowed functional signatures for public user methods only. The issue is marked as info, since despite the severity of possible exploitation, auditors, though the round of testing, verified contracts methods. However, since contracts are upgradeable, it creates risk during further development.

Recommendation:

Verify access control to have admin roles not distributed for the contract. Provide monitoring of the MetaTransactions during the active phase of the protocol. Also, it is recommended to provide sanitizing measures on the frontend part of the dApp to restrict possible functions to be called.

Post-audit.

The Cryptopia team has assured that the token won't be upgradable after the running of the script migration 3_transfer_ownership.ts. In the following script, the ownership of Proxy Admin is transferred to the address(1) which is the equivalent of renouncing the role. Nevertheless, the token remains upgradable until the script is run. Thereby we recommend the Cryptopia team to be careful with any possible upgrades of the token.

LOWEST-2	✔ Verified

Ship's exisiting is checked based on the arbitrary stat

CryptopiaEarlyAccessTokenShip.sol, _exists()

The function now checks if the ship exists based on the base_speed property. However, there is no guarantee that this property cannot be 0. Thus, there is a possibility of collisions. The issue is marked as info since it refers to business logic and should be verified by the Cryptopia team.

Recommendation:

Verify that base_speed will never be set to 0 for existing ships and add the appropriate restriction into the ship setting functionality.

Post-audit.
According to the Cryptopia team, this is implemented by design. The validation will work fine if the base_speed of existing ship is greater than 0.

LOWEST-3	✔ Verified

No verification for the allocation token to be minted for the contract

CryptopiaEarlyAccessTokenShip.sol

There is a functionality for the allocation token to be transferred to the claimer or beneficiary. However, there is no functionality for the token distribution to the contract in the first place.

Recommendation:

Provide the description of the allocation token distribution to the contract.

Post-audit. The Cryptopia team has verified that the necessary amount of token will be transferred manually to the contract's balance. However we still recommend to add safety checks either in the contract's balance or in the additional script to ensure that tokens are transferred in a correct amount.

LOWEST-4	Acknowledged

Custom errors should be used.

NativeMetaTransaction.sol, CryptopiaERC721.sol, CryptopiaEarlyAccessShipTokenFactory.sol, TokenRetriever.sol
Starting from the 0.8.4 version of Solidity, it is recommended to use custom errors instead of storing error message strings in storage and use “require” statements. Using custom errors is more efficient regarding gas spending and increases code readability.

Recommendation:

Use custom errors.

Post-audit.
The team has acknowledged the issue. However due to the timelines, custom errors won’t be implemented in the current code.

LOWEST-5	✔ Verified

Parameters lack validation.

CryptopiaEarlyAccessShipTokenFactory.sol: initialze(), setBeneficiary(), setReferrer().
Address parameters miss validation for not being zero address. The issue is important for the `_beneficiary`parameter since funds are sent to this address.
CryptopiaEarlyAccessShipToken.sol: initialize().
It is recommended to validate if a token parameter is a smart contract and a beneficiary is not a zero address.
CryptopiaERC721.sol: __CryptopiaERC721_init().
Validate that parameter _authenticator implements the interface of IAuthenticator.

Recommendation:

Add validations.

Post-audit:

Cryptopia team verified parameters manually, via config and via the migration script

LOWEST-6	Acknowledged

Lack of events.

CryptopiaEarlyAccessShipTokenFactory.sol: setBeneficiary(), setReferrer(), setLegacyMintedAt(), upgrade(), mint(), withdraw().
CryptopiaEarlyAccessShipToken.sol: setSkins(), claim(), withdraw().
CryptopiaERC721.sol: setContractURI(), setBaseTokenURI().
In order to keep track of historical changes in storage variables, it is recommended to emit events on every change in the functions that modify the storage or interact with funds.

Recommendation:

Emit events in the functions which modify state.

Post-audit.
The team has acknowledged the issue. However due to the timelines, events won't be implemented in the current code.

LOWEST-7	✔ Verified

Token is upgradable.

CryptopiaEarlyAccessShipToken.sol
The implementation of NFT is upgradable, which contradicts the safety pattern of ERC721. Since the ERC721 usually represents valuable assets, it is implemented as a non-upgradable token. Moreover, some marketplaces may not accept a token with upgradable implementation due to safety issues since the implementation owner can update its logic regarding ownership of NFTs and NFT metadata.

Recommendation:

Verify that implementation of CryptopiaEarlyAccessShipToken should be upgradable.

Post-audit. The token's upgradability will be permanently disabled after running the script 3_transfer_ownership.ts

LOWEST-8	Acknowledged

Insecure randomness.

CryptopiaEarlyAccessShipTokenFactory.sol: _random().
The random value is generated based on the previous block's hash, the address of msg.sender, and the current random seed. Such randomness can be easily manipulated as transactions can be included in a certain block where the previous block's hash will give the desired value. This value is then used to determine the rarity of the ship. Thus, manipulating the randomness may provide the users with a certain advantage. The issue is marked as info due to the statement in the commentary section where it is said that such randomness is enough of the protocol.

Recommendation:

We recommend two approaches on how to achieve a random value on-chain.

1. Chainlink VRF oracles or any alternative, including a custom backend for generating random values on-chain.
2. Usage of a future randao value. In this solution, mint can be split into two transactions. The first transaction determines the future block number (for example, block.number + 128), and the second one will perform the mint using the block.prevrandao for a truly random value after a determined earlier block. **Note**! This solution works only for Ethereum forks of Ethereum, which migrated to PoS. In other chains, you may use a hash of a future block specified block (block.number + 128). (It is more difficult to manipulate future blockchash than blockhash of previous block). Also, **note** that the function blockchash`has access only to the last 256 blocks. Thus, the user should be on time to be able to mint NFT using the hash of the specified block.

Post-audit.

The team has acknowledged the issue and verified that such type of randomness generation is enough for the protocol. Though auditors' team still leave the concern about it as it is the violation of a standard checklist.

contracts\source\tokens\ERC721\CryptopiaEarlyAccessShip
CryptopiaEarlyAccessShipTokenFactory.sol
CryptopiaEarlyAccessShipToken.sol
contracts\source\tokens
ERC721\CryptopiaERC721.sol

✔ Re-entrancy	Pass
✔ Access Management Hierarchy	Pass
✔ Arithmetic Over/Under Flows	Pass
✔ Delegatecall Unexpected Ether	Pass
✔ Default Public Visibility	Pass
✔ Hidden Malicious Code	Pass
✔ Entropy Illusion (Lack of Randomness)	Fail
✔ External Contract Referencing	Pass
✔ Short Address/Parameter Attack	Pass
✔ Unchecked CALL Return Values	Pass
✔ Race Conditions/Front Running	Pass
✔ General Denial Of Service (DOS)	Pass
✔ Uninitialized Storage Pointers	Pass
✔ Floating Points and Precision	Pass
✔ Tx.Origin Authentication	Pass
✔ Signatures Replay	Pass
✔ Pool Asset Security (backdoors in the underlying ERC-20)	Pass

contracts\source\common\meta_transactions
ContentMixin.sol
EIP712Base.sol
NativeMetaTransaction.sol
contracts\source\tokens\ERC20\retriever
TokenRetriever.sol

✔ Re-entrancy	Pass
✔ Access Management Hierarchy	Pass
✔ Arithmetic Over/Under Flows	Pass
✔ Delegatecall Unexpected Ether	Pass
✔ Default Public Visibility	Pass
✔ Hidden Malicious Code	Pass
✔ Entropy Illusion (Lack of Randomness)	Pass
✔ External Contract Referencing	Pass
✔ Short Address/Parameter Attack	Pass
✔ Unchecked CALL Return Values	Pass
✔ Race Conditions/Front Running	Pass
✔ General Denial Of Service (DOS)	Pass
✔ Uninitialized Storage Pointers	Pass
✔ Floating Points and Precision	Pass
✔ Tx.Origin Authentication	Pass
✔ Signatures Replay	Pass
✔ Pool Asset Security (backdoors in the underlying ERC-20)	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES, PREPARED BY BLAIZE SECURITY TEAM

Cryptopia: CryptopiaERC721

Getters
- ✓ Should get token uri by token id (87ms)
- ✓ Should return true if owner or approved (78ms)

Set contract uri
- ✓ Should set new contract uri
- ✓ Should revert with 'AccessControl: account is missing role' (107ms)

Set contract uri
- ✓ Should set new contract uri
- ✓ Should revert with 'AccessControl: account is missing role' (82ms)

Set base contract uri
- ✓ Should set new base contract uri
- ✓ Should revert with 'AccessControl: account is missing role' (82ms)

#supportsInterface
- ✓ Should return true

Cryptopia: CryptopiaEarlyAccessShipToken

withdraw
- ✓ Change receiver address balance (49ms)
- ✓ Should revert with 'Not beneficiary' (50ms)
- ✓ Should revert with 'Unable to withdraw'

claim
- ✓ Change receiver address balance (96ms)
- ✓ Should revert with 'No allocation' (114ms)
- ✓ Should revert with 'Not owner of toke' (91ms)
- ✓ Should revert with 'Unable to claim' (74ms)

mintTo
- ✓ Change receiver address balance (62ms)
- ✓ Should revert with 'AccessControl: account is missing role' (108ms)

Getters
- ✓ Should check ship instances (82ms)
- ✓ Should getShipCount

Ship Getters
- ✓ Should getShips (41ms)

Setters
✓ Should set ship skins
✓ Should revert with 'AccessControl: account is missing role' (103ms)
Get ship by name
✓ Should return data by ship name
#retrieveTokens
✓ Should retrieve contract balance (50ms)
✓ Should revert with 'Use withdraw'
✓ Should revert with 'AccessControl: account is missing role' (100ms)
Get ship by name
✓ Should return data by ship name

Cryptopia: Factory

Mint
✓ Change receiver address balance (75ms)
✓ Change receiver address balance with referrer bytes(0) (64ms)
✓ Revert with 'Referrer not found' (65ms)
✓ revert with 'Unable to pay'
Set beneficiary
✓ Sets new beneficiary
✓ Should revert with 'Beneficiary cannot be zero address'
✓ Should revert with 'Ownable: caller is not the owner'
Set referrer
✓ Sets new referrer
✓ Should revert with 'Referrer cannot be zero address'
✓ Should revert with 'Ownable: caller is not the owner'
Set legacy minted at
✓ Sets legacy mint
✓ Should revert with 'Ownable: caller is not the owner'
upgrade
✓ Upgrade tokens (189ms)
✓ Reverts with 'Sender not owner' (89ms)
✓ Reverts with 'Ship not upgradable'
Withdraw
✓ Change receiver address balance (89ms)
✓ Should revert with 'Ownable: caller is not the owner'
Get current token id
✓ Should get current token id

#retrieveTokens
- ✓ Should retrieve contract balance (44ms)
- ✓ Should revert with 'Ownable: caller is not the owner'

#onERC721Received
- ✓ Should return true

#initialized
- ✓ Should revert with 'Initializable: contract is already initialized'
- ✓ Should revert with 'Beneficiary cannot be zero address' (336ms)

Cryptopia: CryptopiaERC721

#retrieveETH()
- ✓ Should retrieve ETH
- ✓ Should revert with 'AccessControl: account is missing role' (112ms)

#isApprovedForAll
- ✓ Should return true

Cryptopia: CryptopiaEarlyAccessShipToken

#initialized
- ✓ Should revert with 'Initializable: contract is already initialized'

Cryptopia: Scenarios
- ✓ Claim daily allocation (428ms)
- ✓ Upgrade tokens by factory: scenarios (240ms)

Cryptopia -- meta transactions

Meta transaction
- ✓ Pay for user transaction (196ms)
- ✓ Try to withdraw token from contract as regular user
- ✓ Try to mint token as regular user
- ✓ Try to set skin as regular user
- ✓ Try to approve and transfer NFT from another user
- ✓ Try to invoke meta transaction with zero address
- ✓ Make bad request

66 passing.

TEST COVERAGE RESULTS

FILE	% STMTS	% BRANCH	% FUNCS
CryptopiaEarlyAccessShipTokenFactory.sol	100	84	100
CryptopiaEarlyAccessShipToken.sol	100	90	100
CryptopiaERC721.sol	100	81.5	100
NativeMetaTransaction.sol	100	100	100
EIP712Base.sol	100	50	100
ContentMixin.sol	100	100	100
TokenRetriever.sol	100	100	100
All files	**100**	**86.5**	**100**

DISCLAIMER

The information presented in this report is an intellectual property of the customer, including all the presented documentation, code databases, labels, titles, ways of usage, as well as the information about potential vulnerabilities and methods of their exploitation. This audit report does not give any warranties on the absolute security of the code. Blaize.Security is not responsible for how you use this product and does not constitute any investment advice.

Blaize.Security does not provide any warranty that the working product will be compatible with any software, system, protocol or service and operate without interruption. We do not claim the investigated product is able to meet your or anyone else's requirements and be fully secure, complete, accurate, and free of any errors and code inconsistency.

We are not responsible for all subsequent changes, deletions, and relocations of the code within the contracts that are the subjects of this report.

You should perceive Blaize.Security as a tool, which helps to investigate and detect the weaknesses and vulnerable parts that may accelerate the technology improvements and faster error elimination.