Cyberscope

Audit Report

KOR Blockchain

April 2022

SHA256 814bb9a677d7be64303334d023536f6b93c6f669bcc03f8dd77b9d9654f1d5dc

Audited by © cyberscope

Table of Contents

Source Files

Filename	SHA256
@chainlink/contracts/src/v0.8/interfaces/AggregatorV3Interface.sol	749f284bc4454e0fce86f63f94ec8778ee6a1e730d7f983d634cb5cd3e2c2b7b
@openzeppelin/contracts-upgradeable/proxy/utils/Initializable.sol	6e058aaee8c641107b209b62c34d484f2f125a44ecb66f7204a701614dfc1d68
@openzeppelin/contracts-upgradeable/security/ReentrancyGuardUpgradeable.sol	b6adbe9bc075b15cfb4b90f1ae020da4c78e3feada056a4c75b875350285c915
@openzeppelin/contracts-upgradeable/token/ERC721/ERC721Upgradeable.sol	f5a8d45253b77c47fbd0bb3313a72137617385a77383755af58569f78ee7b990
@openzeppelin/contracts-upgradeable/token/ERC721/extensions/ERC721EnumerableUpgradeable.sol	0b1fb4e02feef40d236a1b13acf26ab9b23736950ed3142297b1680840e491bb
@openzeppelin/contracts-upgradeable/token/ERC721/extensions/IERC721EnumerableUpgradeable.sol	bb1d9c3bfcbd9561bfd84793e30332bed0da049c9fb4731f1398eaaac22d13b4

@openzeppelin/contracts-upgradeable/token/ERC721/extensions/IERC721MetadataUpgradeable.sol	78d97961d78c8245d51fef2306c33bbc97edb18cb61159979a75187379e93c86
@openzeppelin/contracts-upgradeable/token/ERC721/IERC721ReceiverUpgradeable.sol	7ba7f2a159c698617f90c59705f1928ba7adb95c52eef0cb4c1ce3f81787bf0c
@openzeppelin/contracts-upgradeable/token/ERC721/IERC721Upgradeable.sol	b7c30218228cde89462d762ff873d1297038bb0f9406da6763bd08e3fccb4f93
@openzeppelin/contracts-upgradeable/utils/AddressUpgradeable.sol	44edc4d7099c781d11421cea2d82a52948e738f5f6191c8ad01dfc0f9858549c
@openzeppelin/contracts-upgradeable/utils/ContextUpgradeable.sol	5fb301961e45cb482fe4e05646d2f529aa449fe0e90c6671475d6a32356fa2d4
@openzeppelin/contracts-upgradeable/utils/CountersUpgradeable.sol	5c1ac829a429b0c2ca9b4c9ed8b78d412320e9175e45f088c4e9056ef95fbf21
@openzeppelin/contracts-upgradeable/utils/introspection/ERC165Upgradeable.sol	fd84e5284eccc479268f0ef36b830019d4f7999ceb7959430d8d8d9e602dd4ef

@openzeppelin/contracts-upgradeable/utils/introspection/IERC165Upgradeable.sol	a39bc026ad6214e9ecd526bd4a1ddf9862d80bd4a9d0d031d9bafa4c3c147c0b
@openzeppelin/contracts-upgradeable/utils/StringsUpgradeable.sol	16a0e36f8dc6a83df3fec4344a11ad166ba99649d1cc52613c7ebe8015bd81a3
@openzeppelin/contracts/token/ERC20/IERC20.sol	c2b06bb4572bb4f84bfc5477dadc0fcc497cb66c3a1bd53480e68bedc2e154a6
contracts/KOR.sol	814bb9a677d7be64303334d023536f6b93c6f669bcc03f8dd77b9d9654f1d5dc

Audit Updates

Initial Audit	5th April 2022
Corrected	

Contract Analysis

Critical Medium Minor

Severity	Description
	Minimum Distribution Reward
	Automatic Tokens Expiration
	Tokens Transform Confirmation
	Rewards expiration
	Prevent Redundant Buys
	Redundant Data Structure
	Fixed Addresses

Minimum Distribution Reward

Criticality	minor
Location	contracts/KOR.sol#L177

Description

The reward amount for each buyer is a calculation that is produced from divisions. Since the division result may be rounded to zero, there are users that may not receive rewards even if they contribute a small amount. The algorithm should guarantee that all the distributors will get a portion from the rewards.

```
uint256 percentOfToken = (totalReward * miners[minerIndex].hashrate) /
totalPower;
uint256 rewardOfToken = (percentOfToken * token.amount * 2) / (3 * 4);
```

Recommendation

The method should check if the awarded amount is enough to distribute the proportional amount of the smaller token holder.

Automatic Tokens Expiration

Criticality	minor
Location	contracts/KOR.sol#L115

Description

The invalidation of the expire token is a fundamental part of the process. There are two issues that may produced:

1. Once the first token expires, then the contract will not be able to distribute rewards, thus the decrease of the mintedMinerCount is redundant.

2. If the expiration issue be fixed, then the decrease of the mintedMinerCount could be embedded in the “isExpired” function, so it will be called even if the contract owner forgot it.

```
function forceExpire(uint256 _tokenId) external onlyOwner {
    require(isExpired(_tokenId), "This token is not expired yet");
    Token memory token = tokenIdToToken[_tokenId];
    uint256 minerIndex = token.index;
    mintedMinerCount[minerIndex] -= token.amount;
}
```

Recommendation

The `isExpired()` could embed the decrease of the mintedMinerCount and execute it if the provided token is expired. That way the process will always be updated and will not depend on the contract owner.

Tokens Transform Confirmation

Criticality	medium
Location	contracts/KOR.sol#L184

Description

The reward distribution mechanism is using the “transfer” function to send the rewards to the users. According to the ERC specification the transfer function returns of the transaction processed successfully. If the transfer returns false, that means that the amount has not transferred. The algorithm is not taking this in account and it will proceed guessing that the amount has been distributed.

```
usdcToken.transfer(ownerOf(i + 1), rewardOfToken);
```

Recommendation

The contract should check if the transfer has been processed successfully.

Rewards expiration

Criticality	medium
Location	contracts/KOR.sol#L171

Description

The rewards distribution is iterating the tokens ascendingly. That means that the first item in the iteration is always the older. Hence, once the first token expires, the award distribution will not be able to operate again even if the rest of the tokens have not expired.

```
for (uint256 i = 0; i < totalSupply(); i++) {
    if (isExpired(i + 1))
        break;
```

Recommendation

The reward distribution algorithm should ignore the expired tokens rather than stopping the entire rewards distribution.

Prevent Redundant Buys

Criticality	minor
Location	contracts/KOR.sol#L139

Description

The contract could prevent the users from buying mines that are worthless like, calling the buyMiner method by providing zero _num. That way the user will buy a miner but will not take any reward.

```
function buyMiner(uint256 index, uint256 _num) external payable nonReentrant {
```

Recommendation

The contract could prevent the users from buying worthless miners.

Redundant Data Structure

Criticality	minor
Location	contracts/KOR.sol#L44

Description

The contract is using two mapping structures. The first one maps an ascending index to the Miner structure. The second, maps the same ascending index to the accumulated sum of miners.

```
mapping(uint256 => Miner) public miners; // miner index to miner
mapping(uint256 => uint256) public mintedMinerCount; // miner index to minted
amount (1 = 1/4)
```

Recommendation

The Miners structure could contain the sum of miners. As a result:

- There will not be two structures mirroring the same mapping.
- Synchronisation issues will be eliminated.
- The business logic will be more clear.

```
struct Miner {
    string minerType;
    uint256 hashrate;
    uint256 numOfMiner;
    uint256 minerCount;
    uint256 price;
}
```

Fixed Addresses

Criticality	minor
Location	contracts/KOR.sol#L57

Description

The contract is using the chainlink's aggregator interface as price oracle. It uses the chainlink's USDC / ETH oracle. This address is created once in the constructor and cannot be changed.

https://data.chain.link/ethereum/mainnet/stablecoins/usdc-eth

```
priceFeed = AggregatorV3Interface(0xdCA36F27cbC4E38aE16C4E9f99D39b42337F6dcf);
```

Recommendation

The contract could provide an option for the contract owner to change the oracle address. This may be helpful in a potential migration or an improved version of the oracle.

Contract Diagnostics

Critical Medium Minor

Severity	Code	Description
	L04	Conformance to Solidity Naming Conventions
	L13	Divide before Multiply Operation
	L14	Uninitialized Variables in Local Scope

L04 - Conformance to Solidity Naming Conventions

Criticality	minor
Location	contracts/KOR.sol#L87,92,97,102,107,115,123,130,139,12 and 15 more

Description

Solidity defines a naming convention that should be followed. Rule exceptions:

- Allow constant variable name/symbol/decimals to be lowercase.
- Allow _ at the beginning of the mixed_case match for private variables and unused parameters.

```
_baseTokenURI
rewardDistributePeriod
expireLimit
_tokenIds
_num
_price
_numOfMiner
_hashrate
_minerType
...
```

Recommendation

Follow the Solidity naming convention.
https://docs.soliditylang.org/en/v0.4.25/style-guide.html#naming-conventions

L13 - Divide before Multiply Operation

Criticality	minor
Location	contracts/KOR.sol#L160

Description

Performing divisions before multiplications may cause lose of prediction.

```
rewardOfToken = (percentOfToken * token.amount * 2) / (3 * 4)
percentOfToken = (totalReward * miners[minerIndex].hashrate) / totalPower
```

Recommendation

The multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	minor
Location	contracts/KOR.sol#L151

Description

The are variables that are defined in the local scope and are not initialized.

```
token
```

Recommendation

All the local scoped variables should be initialized.

Contract Functions

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
AggregatorV3Interface	Interface			
	decimals	External		-
	description	External		-
	version	External		-
	getRoundData	External		-
	latestRoundData	External		-
Initializable	Implementation			
	_isConstructor	Private		
ReentrancyGuardUpgradeable	Implementation	Initializable		
	__ReentrancyGuard_init	Internal	✓	onlyInitializing
	__ReentrancyGuard_init_unchained	Internal	✓	onlyInitializing
ERC721Upgradeable	Implementation	Initializable, ContextUpgradeable, ERC165Upgradeable, IERC721Upgradeable, IERC721MetadataUpgradeable		
	__ERC721_init	Internal	✓	onlyInitializing
	__ERC721_init_unchained	Internal	✓	onlyInitializing
	supportsInterface	Public		-
	balanceOf	Public		-
	ownerOf	Public		-
	name	Public		-

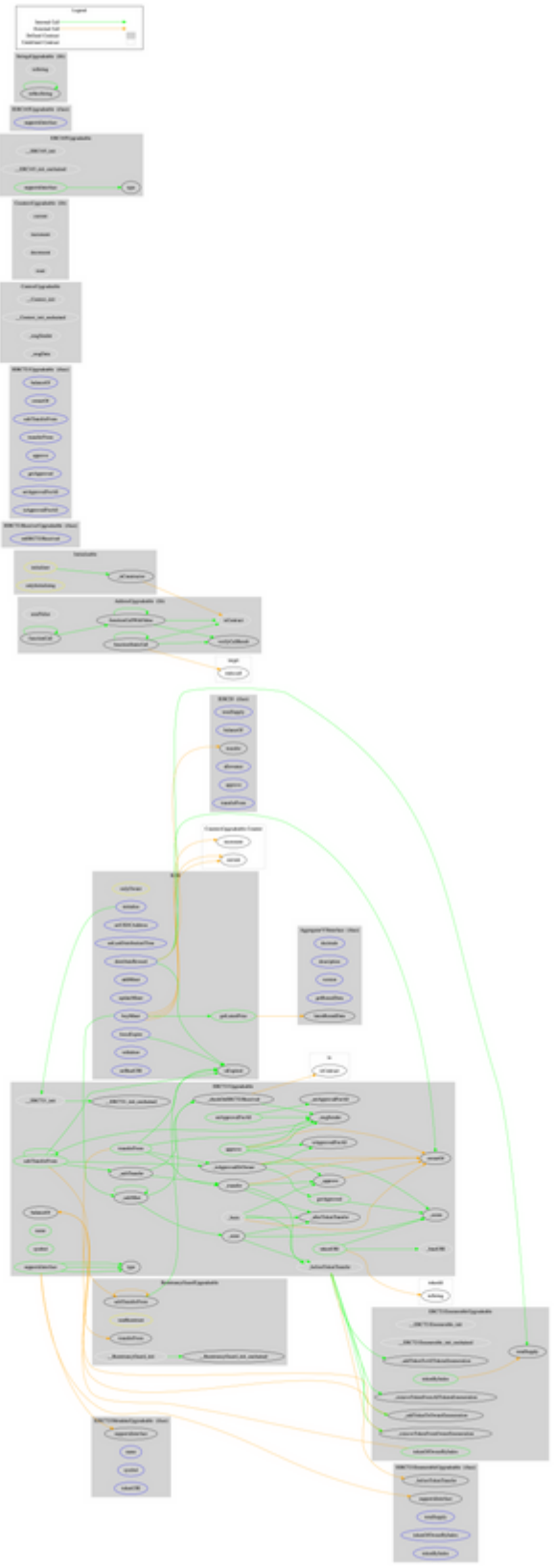
	symbol	Public		-
	tokenURI	Public		-
	_baseURI	Internal		
	approve	Public	✓	-
	getApproved	Public		-
	setApprovalForAll	Public	✓	-
	isApprovedForAll	Public		-
	transferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	_safeTransfer	Internal	✓	
	_exists	Internal		
	_isApprovedOrOwner	Internal		
	_safeMint	Internal	✓	
	_safeMint	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_transfer	Internal	✓	
	_approve	Internal	✓	
	_setApprovalForAll	Internal	✓	
	_checkOnERC721Received	Private	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
ERC721EnumerableUpgradeable	Implementation	Initializable, ERC721Upgradeable, IERC721EnumerableUpgradeable		
	__ERC721Enumerable_init	Internal	✓	onlyInitializing
	__ERC721Enumerable_init_unchained	Internal	✓	onlyInitializing
	supportsInterface	Public		-
	tokenOfOwnerByIndex	Public		-
	totalSupply	Public		-
	tokenByIndex	Public		-
	_beforeTokenTransfer	Internal	✓	

	_addTokenToOwnerEnumeration	Private	✓	
	_addTokenToAllTokensEnumeration	Private	✓	
	_removeTokenFromOwnerEnumeration	Private	✓	
	_removeTokenFromAllTokensEnumeration	Private	✓	
IERC721EnumerableUpgradeable	Interface	IERC721Upgradeable		
	totalSupply	External		-
	tokenOfOwnerByIndex	External		-
	tokenByIndex	External		-
IERC721MetadataUpgradeable	Interface	IERC721Upgradeable		
	name	External		-
	symbol	External		-
	tokenURI	External		-
IERC721ReceiverUpgradeable	Interface			
	onERC721Received	External	✓	-
IERC721Upgradeable	Interface	IERC165Upgradeable		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	transferFrom	External	✓	-
	approve	External	✓	-
	getApproved	External		-
	setApprovalForAll	External	✓	-
	isApprovedForAll	External		-
	safeTransferFrom	External	✓	-

AddressUpgradeable	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	verifyCallResult	Internal		
ContextUpgradeable	Implementation	Initializable		
	__Context_init	Internal	✓	onlyInitializing
	__Context_init_unchained	Internal	✓	onlyInitializing
	_msgSender	Internal		
	_msgData	Internal		
CountersUpgradeable	Library			
	current	Internal		
	increment	Internal	✓	
	decrement	Internal	✓	
	reset	Internal	✓	
ERC165Upgradeable	Implementation	Initializable, IERC165Upgradeable		
	__ERC165_init	Internal	✓	onlyInitializing
	__ERC165_init_unchained	Internal	✓	onlyInitializing
	supportsInterface	Public		-
IERC165Upgradeable	Interface			
	supportsInterface	External		-

StringsUpgradeable	Library			
	toString	Internal		
	toHexString	Internal		
	toHexString	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
KOR	Implementation	ERC721EnumerableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	External	✓	initializer
	getLatestPrice	Public		-
	isExpired	Public		-
	safeTransferFrom	Public	✓	-
	safeTransferFrom	Public	✓	-
	transferFrom	Public	✓	-
	setUSDCAddress	External	✓	onlyOwner
	setLastDistributionTIme	External	✓	onlyOwner
	forceExpire	External	✓	onlyOwner
	addMiner	External	✓	onlyOwner
	updateMiner	External	✓	onlyOwner
	buyMiner	External	Payable	nonReentrant
	distributeReward	External	✓	onlyOwner
	withdraw	External	✓	onlyOwner
	_baseURI	Internal		
	setBaseURI	External	✓	onlyOwner

Contract Flow



Summary

KOR Blockchain is running a mining mechanism. The users are able to buy a miner. As a reward an NFT is minted to the users. Every two weeks the amount is distributed to the users as a profit from the corresponding miner. This audit mentions some security concerns, performance improvements and potential business logic vulnerabilities.

Disclaimer

All the content provided in this document is for general information only and should not be used as financial advice or a reason to buy any investment.

Cyberscope team provides no guarantees against the sale of team tokens or the removal of liquidity by the project audited in this document. Always Do your own research and protect yourselves from being scammed.

The Cyberscope team has audited this project for general information and only expresses their opinion based on similar projects and checks from popular diagnostic tools. Under no circumstances did Cyberscope receive a payment to manipulate those results or change the awarding badge that we will be adding in our website.

Always Do your own research and protect yourselves from scams. This document should not be presented as a reason to buy or not buy any particular token.

The Cyberscope team disclaims any liability for the resulting losses.

About Cyberscope

Coinscope audit and K.Y.C. service has been rebranded to Cyberscope.

Coinscope is the leading early coin listing, voting and auditing authority firm. The audit process is analyzing and monitoring many aspects of the project. That way, it gives the community a good sense of security using an informative report and a generic score.

Cyberscope and Coinscope are aiming to make crypto discoverable and efficient globally. They provides all the essential tools to assist users draw their own conclusions.

The Cyberscope team

https://www.cyberscope.io