Cyberscope

Audit Report

MINER

February 2024

Repository https://github.com/Miner-Labs/ERC-X

Commit 0x8ef788f2e780c74661e3f0d364b320d9a305ac25

Audited by © cyberscope

Table of Contents

Table of Contents **1**
Review **2**
Audit Updates 2
Source Files 3
Overview **5**
Findings Breakdown **6**
Diagnostics **7**
CR - Code Repetition 8
Description 8
Recommendation 9
L04 - Conformance to Solidity Naming Conventions 10
Description 10
Recommendation 11
L09 - Dead Code Elimination 12
Description 12
Recommendation 13
L17 - Usage of Solidity Assembly 14
Description 14
Recommendation 14
Functions Analysis **15**
Inheritance Graph **18**
Flow Graph **19**
Summary **20**
Disclaimer **21**
About Cyberscope **22**

Review

Contract Name	ERC_X
Repository	https://github.com/Miner-Labs/ERC-X
Commit	0x8ef788f2e780c74661e3f0d364b320d9a305ac25
Testing Deploy	https://mumbai.polygonscan.com/address/0x8EF788f2E780c74661e3f0d364b320D9a305Ac25
Symbol	MINER
Decimals	18
Total Supply	10,000

Audit Updates

Initial Audit	14 Feb 2024
Corrected Phase 2	16 Feb 2024
Corrected Phase 3	21 Feb 2024

Source Files

Filename	SHA256
solady/src/utils/LibBitmap.sol	f6fe6b4e4c0bfb1f857a0ccbb985732cc3fd10dba91ab319fb53fb3025046019
solady/src/utils/LibBit.sol	17f56ba291eb0a3c55710e204daa43f2f51a701a9dbdbf9a0a3f9ec081205cef
contracts/ERCX.sol	c9bb1cb0df21f1247f757ef8b57ae53f5e9098eb0d5671f30f8bef4ce765734d
@openzeppelin/contracts/utils/Strings.sol	0519199dbc635f98ce2e4537986604ee618bca665c65e9a1738702dfacf72010
@openzeppelin/contracts/utils/Context.sol	847fda5460fee70f56f4200f59b82ae622bb03c79c77e67af010e31b7e2cc5b6
@openzeppelin/contracts/utils/Address.sol	b3710b1712637eb8c0df81912da3450da6ff67b0b3ed18146b033ed15b1aa3b9
@openzeppelin/contracts/utils/math/SignedMath.sol	768c28e3a33c3312e57ae8a1caaec2893bc89ac6e386621de018f85e9a2d6e99
@openzeppelin/contracts/utils/math/Math.sol	a6ee779fc42e6bf01b5e6a963065706e882b016affbedfd8be19a71ea48e6e15
@openzeppelin/contracts/utils/introspection/IERC165.sol	07ae1ac964ab74dedada999e2dfc642031a6495469cffc0bf715daa4f1e4f904
@openzeppelin/contracts/utils/introspection/ERC165.sol	99348354365cbdeb90157e2903334b861a00d69faab7720ae542d911d5c70d87
@openzeppelin/contracts/token/ERC721/IERC721Receiver.sol	ad84633afd30a71a888403875e2ca7db9e97499d258ccdd98f48dd7cea8229f5
@openzeppelin/contracts/token/ERC721/IERC721.sol	8239cba58986af2deb65af8092d5abcd4ffe60a38dee00cb2e8c6465f945c2d0

@openzeppelin/contracts/token/ERC721/extensions/IERC721Metadata.sol	63d104c065c281e8d8e1fabe0eda19ff83d4660ce1378f9dad2471226d2bf34f
@openzeppelin/contracts/token/ERC20/IERC20.sol	6f2faae462e286e24e091d7718575179644dc60e79936ef0c92e2d1ab3ca3cee
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	1d079c20a192a135308e99fa5515c27acfbb071e6cdb0913b13634e630865939
@openzeppelin/contracts/token/ERC1155/IERC1155Receiver.sol	a44a901c06e11f4b5d2d03753d8552bf0ebc13ff5e880922c64704b12182589e
@openzeppelin/contracts/token/ERC1155/IERC1155.sol	eed90b2884d4acacda32332198e84f991a88c86056b175a35cdb0a90a971907b
@openzeppelin/contracts/token/ERC1155/extensions/IERC1155MetadataURI.sol	bae48bba675d9918f8a3a54045f64ad82c46cc417fc116ce40efeeea24d8451c
@openzeppelin/contracts/interfaces/draft-IERC6093.sol	4aea87243e6de38804bf8737bf86f750443d3b5e63dd0fd0b7ad92f77cdbd3e3
@openzeppelin/contracts/access/Ownable.sol	38578bd71c0a909840e67202db527cc6b4e6b437e0f39f0c909da32c1e30cb81

Overview

The ERC_X contract extends a multifaceted contract ERCX, incorporating functionalities from various token standards including ERC-1155, ERC-721, and ERC-20. This contract aims to offer a unified framework that supports the issuance, transfer, and management of both fungible and non-fungible tokens (NFTs) within a single contract architecture. It leverages OpenZeppelin's robust, secure, and widely adopted libraries to ensure compliance with the token standards mentioned above, while also adding unique features and custom error handling to improve the contract's usability and security.

The ERC_X contract introduces several innovative features such as dynamic URI management for token metadata, which allows for on-the-fly adjustments to token URIs, enabling a flexible and updatable approach to token metadata. It implements ownership checks, safe transfer mechanisms, and approval workflows to ensure secure token transactions. The contract also includes a mechanism for restricting the number of tokens a wallet can hold and implements a transfer delay feature to mitigate potential sniping and ensure fair trading practices post-launch. Additionally, the contract features a custom minting and burning logic that integrates with the contract's fungible token economy, allowing for the seamless conversion between fungible tokens and NFTs based on predefined rates, thereby bridging the gap between these two token types. This holistic approach to token management within a single contract framework showcases a forward-thinking design aimed at providing a comprehensive and versatile token issuance and management platform.

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	4

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	4	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	CR	Code Repetition	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L485,713,771
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.

```
assembly {
    fromMasked := and(from, _BITMASK_ADDRESS)
    log4(
        0,
        0,
        _TRANSFER_EVENT_SIGNATURE,
        fromMasked,
        0,
        mload(add(ids, 0x20))
    )

    for {
        let arrayId := 2
    } iszero(eq(arrayId, end)) {
        arrayId := add(arrayId, 1)
    } {
        log4(0, 0, _TRANSFER_EVENT_SIGNATURE, fromMasked, 0, mload(add(ids,
mul(0x20, arrayId))))
    }
}
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L1140,1145,1146,1147,1148,1149,1192,1197
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.

```
contract ERC_X is ERCX {
    using Strings for uint256;
    string public dataURI;
    string public baseTokenURI;

    uint8 private constant _decimals = 18;
...
        }
    }

    function uri(uint256 id) public view override returns (string memory) {
        return tokenURI(id);
    }
}

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L556,638,685
Status	Unresolved

Description

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

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

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

```
function _mint(
        address to,
        uint256 amount,
        bytes memory data
    ) internal virtual {
       (uint256[] memory ids, uint256[] memory amounts) =  _mintWithoutCheck(to,
amount);

        uint256 end = _nextTokenId();
        _doSafeBatchTransferAcceptanceCheck(_msgSender(), address(0), to, ids,
amounts, data);
        if (_nextTokenId() != end) revert Reentrance();
    }

...
```

Recommendation

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

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L410,485,602,658,713,771
Status	Unresolved

Description

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

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

```
assembly {
            // Mask `to` to the lower 160 bits, in case the upper bits somehow
aren't clean.
            toMasked := and(to, _BITMASK_ADDRESS)
            fromMasked := and(from, _BITMASK_ADDRESS)
            // Emit the `Transfer` event.
            log4(
                0, // Start of data (0, since no data).
                0, // End of data (0, since no data).
                _TRANSFER_EVENT_SIGNATURE, // Signature.
                fromMasked, // `from`.
                toMasked, // `to`.
                amount // `tokenId`.
            )
        }

...
```

Recommendation

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

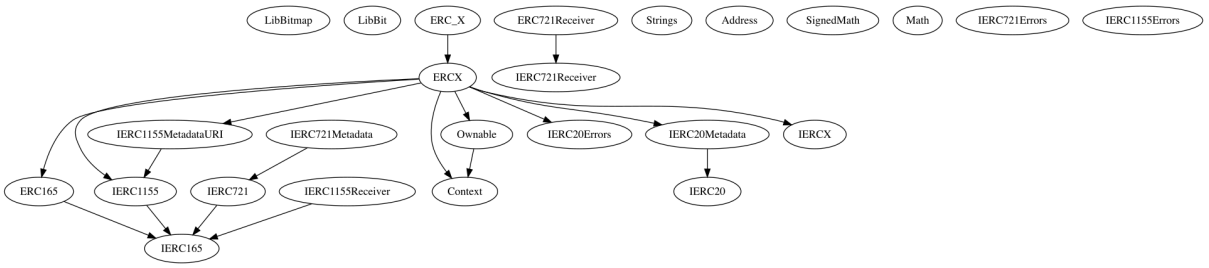
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERCX	Interface			
	isOwnerOf	External		-
ERC721Receiver	Implementation	IERC721Receiver		
	onERC721Received	External	✓	-
ERCX	Implementation	Context, ERC165, IERC1155, IERC1155MetadataURI, IERCX, IERC20Metadata, IERC20Errors, Ownable		
		Public	✓	Ownable
	setWhitelist	Public	✓	onlyOwner
	_startTokenId	Internal		
	_nextTokenId	Internal		
	_totalMinted	Internal		
	isOwnerOf	Public		-
	supportsInterface	Public		-
	uri	Public		-

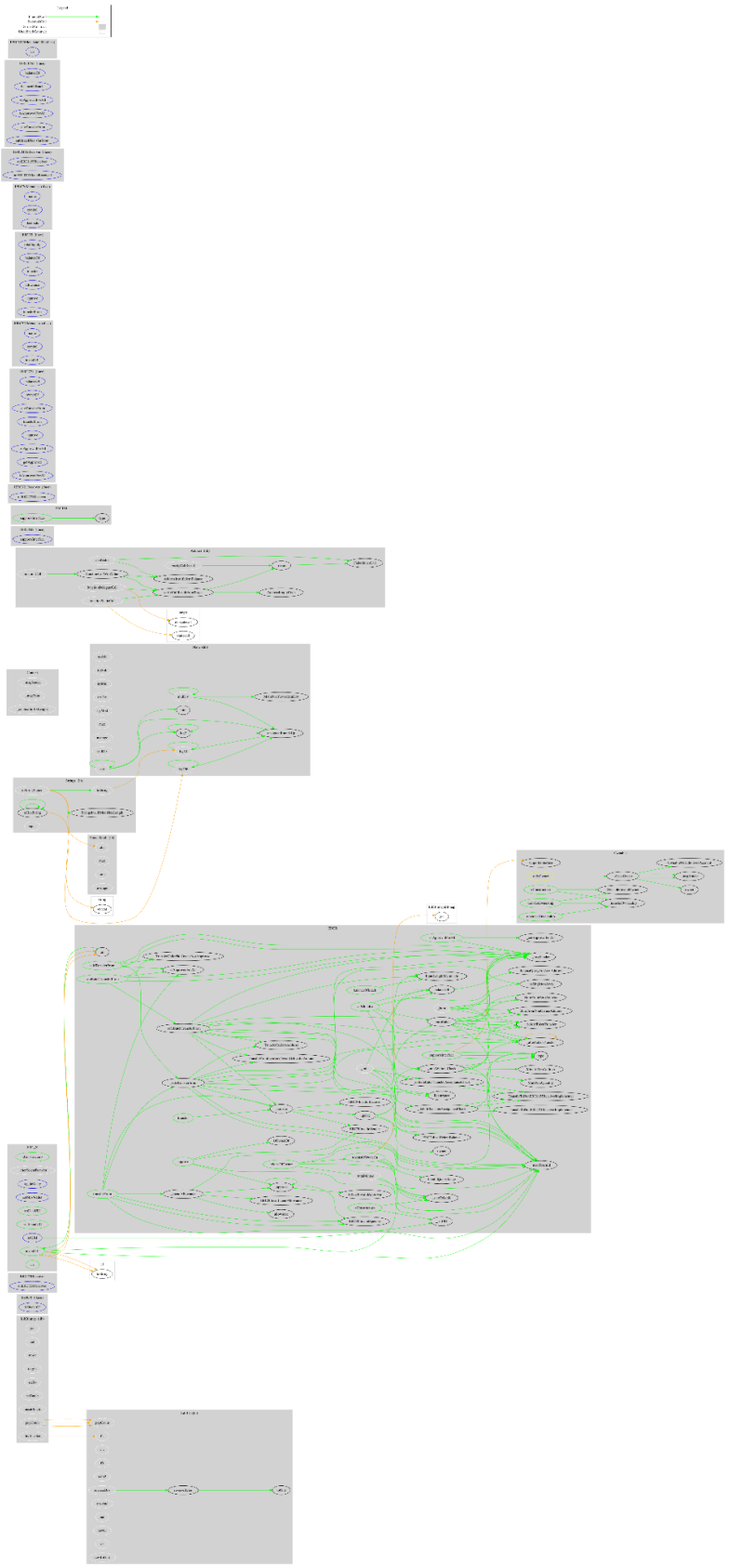
	balanceOf	Public		-
	balanceOf	Public		-
	balanceOf	Public		-
	balanceOfBatch	Public		-
	setApprovalForAll	Public	✓	-
	isApprovedForAll	Public		-
	safeTransferFrom	Public	✓	-
	safeBatchTransferFrom	Public	✓	-
	_safeTransferFrom	Internal	✓	
	_safeBatchTransferFrom	Internal	✓	
	_setURI	Internal	✓	
	_mint	Internal	✓	
	_mintWithoutCheck	Internal	✓	
	_burn	Internal	✓	
	_burnBatch	Internal	✓	
	_burnBatch	Internal	✓	
	_setApprovalForAll	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
	_doSafeTransferAcceptanceCheck	Private	✓	
	_doSafeBatchTransferAcceptanceCheck	Private	✓	
	_asSingletonArray	Private		
	transfer	Public	✓	-

	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Internal	✓	
	_update	Internal	✓	
	_approve	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	tokensOfOwnerIn	Public		-
	tokensOfOwner	Public		-
ERC_X	Implementation	ERCX		
		Public	✓	ERCX
	_afterTokenTransfer	Internal	✓	
	toggleDelay	External	✓	onlyOwner
	setMaxWallet	External	✓	onlyOwner
	setDataURI	Public	✓	onlyOwner
	setTokenURI	Public	✓	onlyOwner
	setURI	External	✓	onlyOwner
	tokenURI	Public		-
	uri	Public		-

Inheritance Graph



Flow Graph



Summary

MINER contract implements a token and NFT mechanism. This audit investigates security issues, business logic concerns, and potential improvements.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

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