Cyberscope

Audit Report

MINER

February 2024

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

Commit 2d5f8b129b7f73a94b5965f5804c972339f0e294

Audited by © cyberscope

Table of Contents

Review

Contract Name	ERC_X
Repository	https://github.com/Miner-Labs/ERC-X
Commit	2d5f8b129b7f73a94b5965f5804c972339f0e294
Testing Deploy	https://mumbai.polygonscan.com/address/0xa988b44d4cd58dc03800f089fdba7c9d24218e5e
Symbol	MINER
Decimals	18
Total Supply	100,000

Audit Updates

Initial Audit	14 Feb 2024
Corrected Phase 2	16 Feb 2024

Source Files

Filename	SHA256
solady/src/utils/LibBitmap.sol	f6fe6b4e4c0bfb1f857a0ccbb985732cc3fd10dba91ab319fb53fb3025046019
solady/src/utils/LibBit.sol	17f56ba291eb0a3c55710e204daa43f2f51a701a9dbdbf9a0a3f9ec081205cef
contracts/ERCX.sol	de7727abb4781e15250600d8b79cd86f321a260e54abe20c016be31731e07e9d
@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	2
Medium	0
Minor / Informative	12

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	12	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	NBV	NFT Burning Vulnerability	Unresolved
●	ST	Stops Transactions	Unresolved
●	CR	Code Repetition	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RCS	Redundant Code Segments	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	TSD	Token Standard Deviation	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L08	Tautology or Contradiction	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

NBV - NFT Burning Vulnerability

Criticality	Critical
Location	contracts/ERCX.sol#L1015
Status	Unresolved

Description

The `_update` function within the contract is designed to facilitate token transfers, incorporating unique procedures for burning and minting Non-Fungible Tokens (NFTs) based on the whitelist status of the sender and recipient. A significant vulnerability is identified in this mechanism, particularly in how NFTs are managed for users whose whitelist status changes. Specifically, the contract attempts to burn NFTs from non-whitelisted senders and mint NFTs for non-whitelisted recipients. An issue arises for users who were previously whitelisted, did not acquire any NFTs during their whitelisted period, and were later removed from the whitelist. In such cases, attempting a transfer could lead to erroneous behavior due to the lack of NFTs to burn, potentially causing the contract to revert or behave unpredictably. Specifically, the `findLastSet` function will return the `type(uint256).max` (max unsigned integer value), no matter the argument passed to it. As a result, every subsequent calculation will be invalid, leading to unpredictable behaviour.

```
bool wlf = whitelist[from];
if (!wlf) {
    uint256 tokens_to_burn = (fromBalance / tokensPerNFT) - ((fromBalance - value) /
tokensPerNFT);
    if(tokens_to_burn > 0)
        _burnBatch(from, tokens_to_burn);
}

if (!whitelist[to]) {
    ...
    } else {
        uint256 tokens_to_mint = ((toBalance + value) / tokensPerNFT) - (toBalance /
tokensPerNFT);
        if(tokens_to_mint > 0)
            _mintWithoutCheck(to, tokens_to_mint);
    }
}
```

Recommendation

The team is advised to revise the contract's logic and ensure that the NFT burn and mint operations take into consideration not only the whitelist status but also on the user's actual NFT balance. The revised implementation should prevent the contract from attempting to burn NFTs from accounts with zero NFT balance, thus avoiding unnecessary transaction reverts and ensuring smoother operation regardless of whitelist status changes. Additionally, the team is recommended to implement additional checks before proceeding with the burn operations. These checks should verify the actual ownership and availability of NFTs in the sender's account prior to burning.

ST - Stops Transactions

Criticality	Critical
Location	contracts/ERCX.sol#L1163
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users excluding the authorized addresses. The owner may take advantage of it by setting the `maxWallet` to zero.

```
require(_balances[to] <= maxWallet, "Transfer exceeds maximum wallet");
```

Additionally, a transaction can be reverted due to the issue described in the NBV section.

Recommendation

The contract could embody a check for not allowing setting the `maxWallet` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

TSD - Token Standard Deviation

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L874,908
Status	Unresolved

Description

The implementation of `_doSafeTransferAcceptanceCheck` and `_doSafeBatchTransferAcceptanceCheck` functions deviates from the intended logic prescribed by the ERC721 and ERC1155 standards for handling token transfers. According to these standards, safe transfer checks should be performed to ensure that the recipient contract is capable of receiving tokens by supporting the relevant interface (IERC721Receiver or IERC1155Receiver). The standard approach involves executing these checks when the recipient is a contract. However, the current implementation triggers the checks based on whether the recipient address is different from `tx.origin`, aiming to differentiate between contracts and EOAs (Externally Owned Accounts) indirectly. This adjustment shifts the focus from verifying the recipient contract's capability to handle tokens to an assumption based on the transaction's origin. This method could restrict legitimate interactions between contracts, potentially limiting the contract's functionality and composability within the broader ecosystem.

```
if (to != tx.origin) {
    if (IERC165(to).supportsInterface(type(IERC1155).interfaceId)) {
        try IERC1155Receiver(to).onERC1155Received(operator, from, id,
amount, data) returns (bytes4 response) {
            if (response != IERC1155Receiver.onERC1155Received.selector) {
...
```

Recommendation

To align the implementation with the ERC721 and ERC1155 standards and ensure the contract's broad compatibility and security, the team is recommended to revise the safe transfer acceptance checks to directly verify the recipient's type (contract or EOA) using a more reliable method. This can be achieved by checking if the code size at the recipient address is greater than zero, which is a standard approach for determining if an address is a contract. This modification ensures that the safe transfer checks are correctly focused on verifying the recipient contract's ability to handle the tokens, adhering to the intent of the ERC721 and ERC1155 standards. It eliminates the reliance on `tx.origin`, which is not a reliable indicator of whether the recipient is a contract or an EOA for the purposes of these checks.

CR - Code Repetition

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L470,706,764
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.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L578
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(type(uint256).max - amount >= startTokenId)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L186,525,1177,1185,1189
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
whitelist[target] = state;
_uri = newuri;
transferDelay = !transferDelay;
dataURI = _dataURI;
baseTokenURI = _tokenURI;
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

RCS - Redundant Code Segments

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L178,412,497,613,662,726,784
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 `erc1155Enabled()` function is designed to indicate whether ERC1155 events are enabled. However, as it is implemented to always return false, any conditional logic dependent on this function's return value will never be executed. This implementation leads to dead code paths in the contract. These sections of code are effectively redundant, contributing to unnecessary code complexity and potential confusion regarding the contract's intended functionality.

```
function erc1155Enabled() internal pure virtual returns (bool) {
    return false;
}

if(erc1155Enabled())
    emit TransferSingle(operator, from, to, id, amount);

...
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L186,1177
Status	Unresolved

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
whitelist[target] = state;
transferDelay = !transferDelay;
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L1135,1140,1141,1142,1143,1144,1184,1188
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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L1181
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
maxWallet = totalSupply * percent / 100
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L1222
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
seed <= 255
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L528,547,630,678
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
    ) internal virtual {
        _mint(to, amount, "");
    }

...
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L1206,1207,1208
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
string memory image
string memory color
string memory description
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/ERCX.sol#L397,470,593,650,706,764
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**
LibBitmap	Library			
	get	Internal		
	set	Internal	✓	
	unset	Internal	✓	
	toggle	Internal	✓	
	setTo	Internal	✓	
	setBatch	Internal	✓	
	unsetBatch	Internal	✓	
	popCount	Internal		
	findLastSet	Internal		
LibBit	Library			
	fls	Internal		
	clz	Internal		
	ffs	Internal		
	popCount	Internal		
	isPo2	Internal		
	reverseBits	Internal		

	reverseBytes	Internal		
	rawAnd	Internal		
	and	Internal		
	rawOr	Internal		
	or	Internal		
	rawToUint	Internal		
	toUint	Internal		
IERCX	Interface			
	isOwnerOf	External		-
ERC721Receiver	Implementation	IERC721Receiver		
	onERC721Received	External	✓	-
ERCX	Implementation	Context, ERC165, IERC1155, IERC1155MetadataURI, IERCX, IERC20Metadata, IERC20Errors, Ownable		
		Public	✓	Ownable
	erc1155Enabled	Internal		
	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	✓	
	_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		-
Strings	Library			
	toString	Internal		
	toStringSigned	Internal		
	toHexString	Internal		
	toHexString	Internal		
	toHexString	Internal		
	equal	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
Address	Library			
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	

	verifyCallResultFromTarget	Internal		
	verifyCallResult	Internal		
	_revert	Private		
SignedMath	Library			
	max	Internal		
	min	Internal		
	average	Internal		
	abs	Internal		
Math	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	max	Internal		
	min	Internal		
	average	Internal		
	ceilDiv	Internal		
	mulDiv	Internal		
	mulDiv	Internal		
	sqrt	Internal		

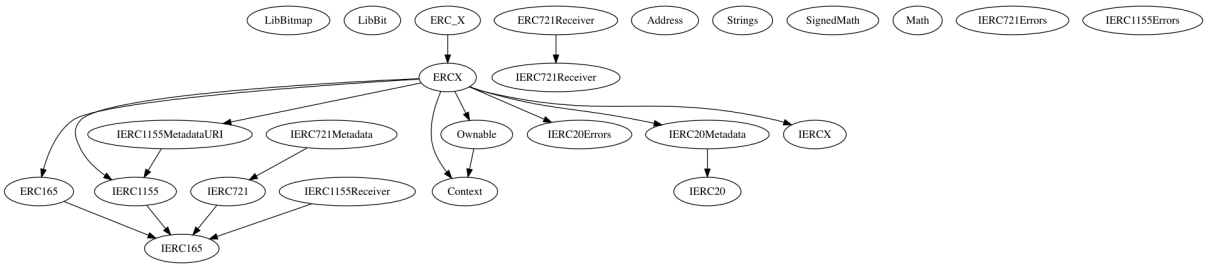
	sqrt	Internal		
	log2	Internal		
	log2	Internal		
	log10	Internal		
	log10	Internal		
	log256	Internal		
	log256	Internal		
	unsignedRoundsUp	Internal		
IERC165	Interface			
	supportsInterface	External		-
ERC165	Implementation	IERC165		
	supportsInterface	Public		-
IERC721Receiver	Interface			
	onERC721Received	External	✓	-
IERC721	Interface	IERC165		
	balanceOf	External		-
	ownerOf	External		-
	safeTransferFrom	External	✓	-
	safeTransferFrom	External	✓	-

	transferFrom	External	✓	-
	approve	External	✓	-
	setApprovalForAll	External	✓	-
	getApproved	External		-
	isApprovedForAll	External		-
IERC721Metadata	Interface	IERC721		
	name	External		-
	symbol	External		-
	tokenURI	External		-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-

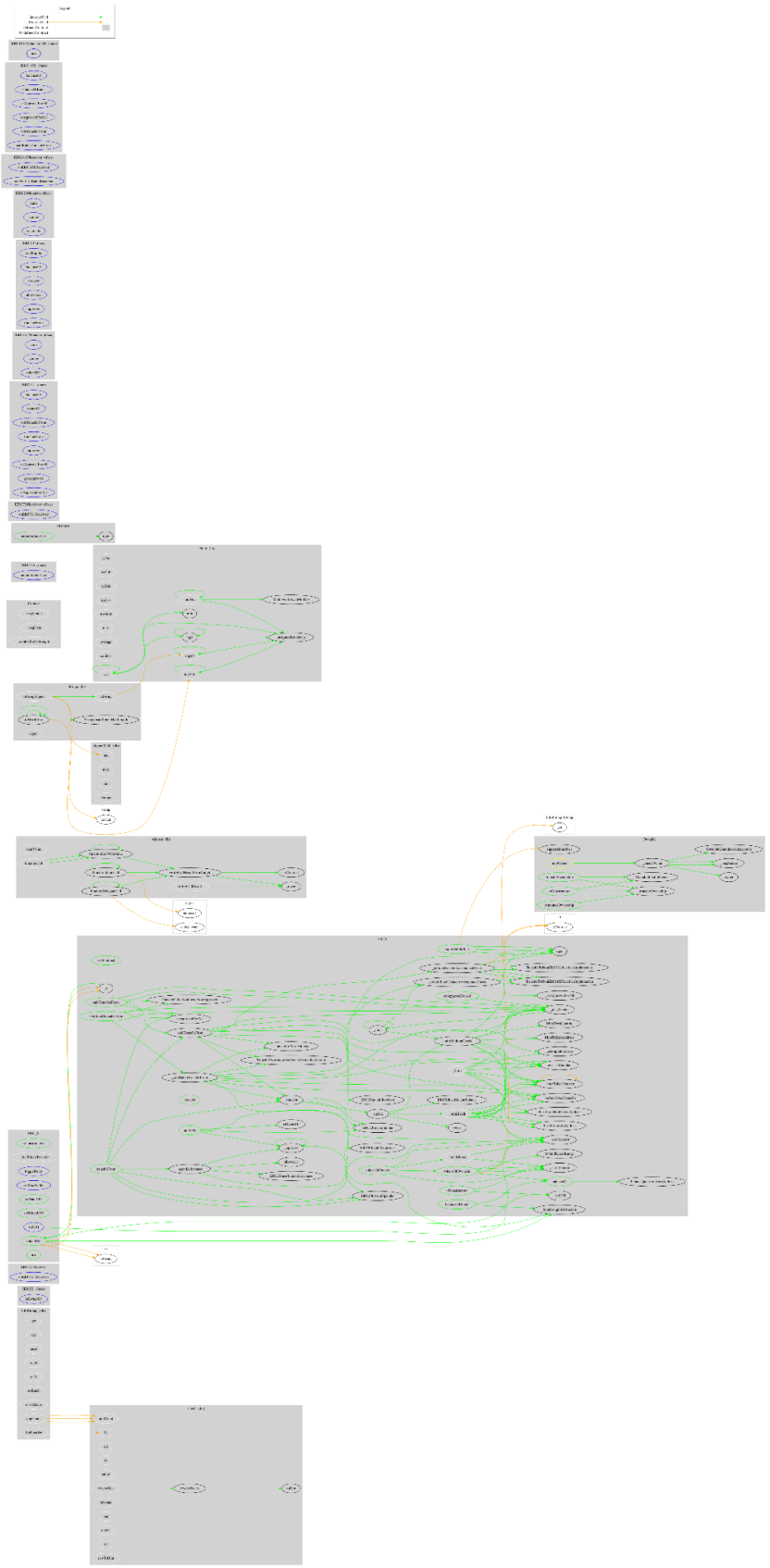
IERC1155Receiver	Interface	IERC165		
	onERC1155Received	External	✓	-
	onERC1155BatchReceived	External	✓	-
IERC1155	Interface	IERC165		
	balanceOf	External		-
	balanceOfBatch	External		-
	setApprovalForAll	External	✓	-
	isApprovedForAll	External		-
	safeTransferFrom	External	✓	-
	safeBatchTransferFrom	External	✓	-
IERC1155MetadataURI	Interface	IERC1155		
	uri	External		-
IERC20Errors	Interface			
IERC721Errors	Interface			
IERC1155Errors	Interface			
Ownable	Implementation	Context		

		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	

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