Cyberscope

Audit Report

Creath Governance

September 2023

Repository https://github.com/Creath-io/marketplace-v2/tree/29642f339163d25a205ab592cbbf8f65a24702b8/contracts

Commit 29642f339163d25a205ab592cbbf8f65a24702b8

Audited by © cyberscope

Table of Contents

Review

Repository	https://github.com/Creath-io/marketplace-v2/tree/29642f339163d25a205ab592cbbf8f65a24702b8/contracts
Commit	29642f339163d25a205ab592cbbf8f65a24702b8

Audit Updates

Initial Audit	20 Sep 2023 https://github.com/cyberscope-io/audits/blob/main/2-cgt/v1/audit.pdf
Corrected Phase 2	28 Sep 2023

Source Files

Filename	SHA256
CreathTreasury.sol	3e8881c1d043975304a0cfa7acac95ffb899c36b1e1554e7f5e27393d9cb8981
CreathMarketplace.sol	12418a277d3305443d960a7c15fe85b783007bd2026ca8a0031cabd23e2253cc
CreathArtTradable.sol	d0498f27e13ccea2166bb786abde46746ee77d543c43185f05fec6d26a054166
CreathArtFactory.sol	717104fbde3f96dd2d63201ef5cb27c88ef0e622c4698ca401977407dd2ea963
Creath.sol	7e3502ae01d6f1b89d64c0d3dfdf1318590f976f95b2de0fe5149aa7beed4c51

Overview

The Creath platform represents an advanced NFT infrastructure, offering a multifaceted marketplace which make possible the purchase of NFT listings. Alongside this, it features a dedicated factory mechanism for the creation of new NFT contracts tailored to specific criteria. To bolster its financial operations, the ecosystem incorporates a robust treasury module, ensuring the safe accumulation and distribution of funds.

Creath contract

The Creath contract serves as the foundational layer for the Creath Marketplace. This contract, inheriting from the `ERC721URIStorage` and `Ownable` standards, is equipped with functionalities that empower the contract owner with distinct privileges. Specifically, the owner can mint new Non-Fungible Tokens (NFTs) and assign them to a designated `_beneficiary` address. Each minted NFT is associated with a unique token ID and a specified URI that provides detailed information about the token. Additionally, the contract owner retains the authority to burn NFTs based on their specific `_tokenId`. The contract also incorporates mechanisms to check the existence of a token, verify approvals, ensuring seamless interactions within the Creath ecosystem. As a result the contract integrates with the marketplace, allowing for streamlined trading and management of NFTs.

CreathArtFactory Conctract

The CreathArtFactory contract, is facilitating the creation and management of NFT contracts. This contract, inheriting from the `Ownable` standard, grants the owner exclusive privileges to deploy new NFT contracts through the `createNFTContract` function. When invoked, this function establishes a new NFT contract with specified `_name` and `_symbol` parameters, while also associating it with the predefined `marketplace` address. Beyond creation, the `CreathArtFactory` contract offers functionalities to register NFT contracts via the `registerTokenContract` method. This function ensures that only genuine ERC721 compliant contracts are registered, enhancing the security and integrity of the platform. Additionally, the owner can disable any registered NFT contract using the `disableTokenContract` function, providing a

mechanism for quality control and contract lifecycle management. Throughout these operations, relevant events such as `ContractCreated` and `ContractDisabled` are emitted, ensuring transparency and traceability within the ecosystem.

CreathArtTradable Contract

The CreathArtTradable contract, is facilitating the creation and management of individual NFT contracts. This contract, inheriting from both `ERC721URIStorage` and `Ownable` , is equipped with the capability to mint new NFTs to a designated beneficiary address, by the mint function. Each minted NFT is uniquely identified by a token ID, which is systematically incremented to ensure distinctiveness.

While it shares many functionalities with the Creath contract, a distinguishing feature of the `CreathArtTradable` contract is its initialization process. This contract mandates the specification of parameters such as `_name` , `_symbol` , and `_marketplace` during its initialization. Upon initialization, the `CreathArtTradable` contract requires specific parameters, namely `_name` , `_symbol` , and `_marketplace` , to be defined. These parameters set the stage for the NFT's identity. Additioanlly, the contract's owner, granted exclusive privileges, can mint new NFTs and also burn them if necessary, by using the burn function.

CreathMarketplace Contract

The `CreathMarketplace` contract is using the `initialize` function which sets the `platformFee` , `feeReceipient` , and the `paymentToken` parameters of the marketplace.

The marketplace is structured around four core functionalities:

1. ListItem Function The `listItem` function facilitates the process of listing an NFT for sale. It incorporates checks to ensure that only ERC721 compliant NFT contracts are eligible for listing. Before listing, the function verifies that the NFT hasn't been previously listed. Once these checks are passed, the NFT's price and associated artist are set, marking it as available for purchase.
2. CancelListing Function The `cancelListing` function offers the flexibility to cancel the NFTs from the marketplace. Upon invocation, the function erases the

NFT's price and artist details from the contract's records, effectively delisting it from the marketplace.

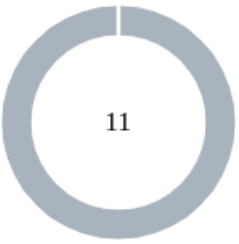
3. UpdateListing Function The `updateListing` function is designed to provide the ability to adjust the price of the already listed NFTs. By setting the new desired price as a parameter, the price of the NFTs can be adjusted .
4. BuyItem Function Potential buyers can use the `buyItem` function to purchase a listed NFT. The function first calculates a `feeAmount` based on the set `platformFee` . This fee is then transferred to the `feeReceipient` address. The remaining amount, after deducting the fee, is sent to the NFT's associated `artist` address. Finally, ownership of the NFT is transferred to the buyer, completing the transaction.

In addition to these core functionalities, the contract is equipped with administrative functions that grant the owner the capability to modify key parameters. Specifically, the owner can update the `platformFee` , `feeReceipient` , and the `paymentToken` , ensuring that the marketplace remains adaptable to evolving requirements and conditions.

CreathTreasury Conctract

The `CreathTreasury` is the contract where tokens are accumulated. Designed with robust security measures, the contract incorporates functionalities that allows the withdrawal of both tokens and native tokens to designated addresses. These withdrawal functions can only be invoked by authorized entities, which the owner have set. The contract is equipped with mechanisms to authorize new addresses and modify existing roles, ensuring that the treasury operations remain flexible and secure.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	OSI	Override Specification Inconsistency	Unresolved
	CCR	Contract Centralization Risk	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L09	Dead Code Elimination	Unresolved
	L13	Divide before Multiply Operation	Unresolved
	L14	Uninitialized Variables in Local Scope	Unresolved
	L15	Local Scope Variable Shadowing	Unresolved
	L16	Validate Variable Setters	Unresolved
	L17	Usage of Solidity Assembly	Unresolved
	L19	Stable Compiler Version	Unresolved

OSI - Override Specification Inconsistency

Criticality	Minor / Informative
Location	Creath.sol#L1723 CreathArtTradable.sol#L1776
Status	Unresolved

Description

The contract is using an `override` within the `isApprovedForAll` function. However, the `override` is only indicating the override for the `IERC721` contract, not the `ERC721` contract. As a result, if the contract is compiled using version `0.8.12` of Solidity or higher, it will encounter compilation errors due to the stricter requirements for specifying overridden contracts.

```
    function isApprovedForAll(address _owner, address operator)
        override(IERC721)
    ....
    }
```

Recommendation

It is recommended to specify the `contracts` that will need to be overridden. If the contract is considered for deployment using a Solidity version of `0.8.12` or above, then the `override(ERC721, IERC721)` syntax should be used instead of the simple `override`, if the intended contracts to be overridden is both the `ERC721` and `IERC721`. This will ensure that the contract adheres to the newer Solidity requirements and can be compiled without issues.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	CreathArtTradable.sol#L1760 CreathMarketplace.sol#L1643
Status	Unresolved

Description

The contract is designed with a high degree of centralization, granting the owner significant authority over its operations. Specifically, the owner has the power to set and modify the `paymentToken` of the contract, determining which token will be used for purchasing NFTs. This level of control allows the owner to potentially switch the payment token at will, which could disrupt users' expectations and operations.

Furthermore, the owner possesses the capability to burn any token from any user. This means that even if an NFT is purchased, the owner can unilaterally decide to burn it, depriving the user of their acquired asset. Such centralized control not only poses risks to the users but also deviates from the decentralized nature of blockchain and smart contracts.

```
    function burn(uint256 _tokenId) external onlyOwner{
        _burn(_tokenId);
    }


    function updatePaymentToken(address newPaymentToken)
external onlyOwner{
        paymentToken = IERC20Upgradeable(newPaymentToken);
        emit PaymentMethodUpdated(newPaymentToken);
    }
```

Recommendation

It is recommended to address these centralization concerns by evaluating the feasibility of integrating critical configurations and functionality directly into the contract's codebase. By doing so, the contract would become more self-reliant, reducing its dependence on external configurations and the associated risks. This would also ensure that the contract operates

in a more predictable and transparent manner, aligning with the principles of decentralization. Additionally, to further safeguard users' assets, the team should consider implementing checks that prevent the owner from burning NFTs that have been purchased by users. This protective measure would enhance trust in the contract and protect users from potential arbitrary actions by the owner.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	CreathArtTradable.sol#L1748 Creath.sol#L1678
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
marketplace
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	CreathTreasury.sol#L303,1405,1454,1469,1480 CreathMarketplace.sol#L219,859,863,907,925,928,943,970,974,1035,1050,1053,1199,1221,1224,1227,1309,1597,1622,1623,1624,1654,1655,1656,1657,1684,1697,1698,1699,1700,1717,1718,1768,1779,1787 CreathArtTradable.sol#L1631,1757,1772,1780,1787 CreathArtFactory.sol#L1494,1691,1706,1714,1780,1791 Creath.sol#L1575,1687,1707,1715,1722
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
address private ADMIN;
(address _marketplace)
(address _token, address _to, uint _amount)
(uint _amount, address _to)
...
```

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	CreathTreasury.sol#L65,90,119,146,156,171,181,220,412,423,438,447,460,473,512,584,591,599,608,632,639,647,658,668,752,765,801,812,854,865,903,916,946,968,993,1000,1031,1357 CreathMarketplace.sol#L301,311,321,331,341,351,478,503,532,559,569,633,801,812,819,859,863,898,925,928,1050,1053,1133,1140,1150,1164,1171,1186,1221,1224,1406,1425,1440,1449,1462,1475,1514 CreathArtTradable.sol#L35,43,62,69,77,86,114,121,129,140,150,234,247,283,294,336,385,398,428,480,487,496,511,518,696,721,731,750,760,777,787,802,812,827,851,863,1631 CreathArtFactory.sol#L35,43,59,66,74,83,108,115,123,134,144,228,241,277,288,330,379,392,422,469,476,485,500,507,573,598,608,627,637,654,664,679,689,704,728,740,1494 Creath.sol#L34,42,57,64,72,81,105,112,120,131,141,225,238,274,285,327,376,389,419,467,474,483,498,505,674,699,709,728,738,755,765,780,790,805,829,841,1411,1575,1642
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 sendValue(address payable recipient, uint256 amount)
internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        (bool success, ) = recipient.call{value: amount}("");
        require(success, "Address: unable to send value,
recipient may have reverted");
    }

function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
        return functionCallWithValue(target, data, 0, "Address:
low-level call failed");
    }

function functionCallWithValue(address target, bytes memory
data, uint256 value) internal returns (bytes memory) {
        return functionCallWithValue(target, data, value,
"Address: low-level call with value failed");
    }

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	CreathTreasury.sol#L714,717,729,733,734,735,736,737,738,744 CreathArtTradable.sol#L196,199,211,215,216,217,218,219,220,226 CreathArtFactory.sol#L190,193,205,209,210,211,212,213,214,220 Creath.sol#L187,190,202,206,207,208,209,210,211,217
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
denominator := div(denominator, twos)
inverse *= 2 - denominator * inverse
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	CreathMarketplace.sol#L1114
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.

```
bytes32 slot
```

Recommendation

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

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	CreathArtTradable.sol#L1744,1745,1787,1808 CreathArtFactory.sol#L1678,1679,1742 Creath.sol#L1722,1743
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
string memory _name
string memory _symbol
address _owner, address operator
address _owner = ERC721.ownerOf(tokenId);
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	CreathTreasury.sol#L1424,1451,1481 CreathMarketplace.sol#L1631,1783CreathArtTradable.sol#L1748 CreathArtFactory.sol#L1682,1771 Creath.sol#L1678
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
ADMIN = _admin;
...
feeReceipient = _feeRecipient
feeReceipient = _platformFeeRecipient
...
marketplace = _marketplace;
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	CreathTreasury.sol#L237,675,974 CreathMarketplace.sol#L283,293,303,313,323,333,343,353,650 CreathArtTradable.sol#L157,461,868,1581 CreathArtFactory.sol#L151,450,745,1444 Creath.sol#L148,448,846,1525
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 {
                let returndata_size := mload(returndata)
                revert(add(32, returndata), returndata_size)
            }

assembly {
...
                prod1 := sub(sub(mm, prod0), lt(mm, prod0))
            }

assembly {
                ptr := add(buffer, add(32, length))
            }
...
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	CreathTreasury.sol#L2 CreathMarketplace.sol#L2 CreathArtTradable.sol#L2 CreathArtFactory.sol#L3 Creath.sol#L2
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

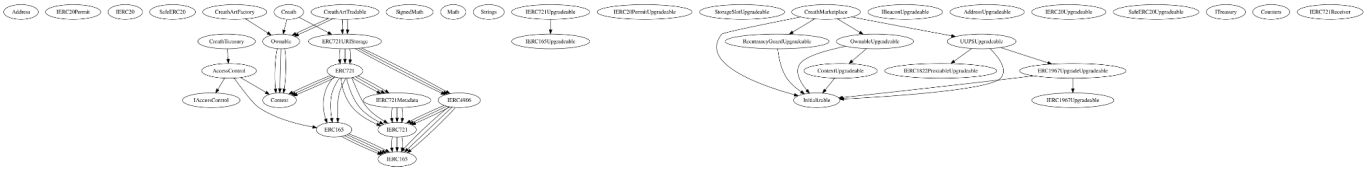
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
CreathTreasury	Implementation	AccessControl		
		Public	✓	-
	isAuthorized	Public		-
	updateAdmin	External	✓	onlyRole
	updateMarketplace	External	✓	onlyRole
	withdrawToken	Public	✓	onlyAuthorized
	withdraw	Public	✓	onlyAuthorized
		External	Payable	-
ITreasury	Interface			
	withdrawToken	External	✓	-
CreathMarketplace	Implementation	Initializable, UUPSUpgradeable, OwnableUpgradeable, ReentrancyGuardUpgradeable		
	initialize	Public	✓	initializer
	_authorizeUpgrade	Internal	✓	onlyOwner

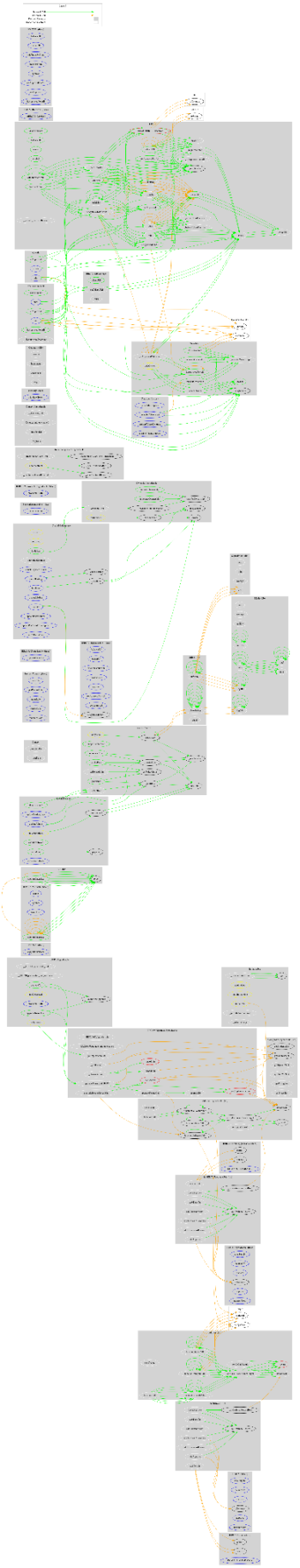
	updatePaymentToken	External	✓	onlyOwner
	listItem	External	✓	onlyOwner notListed
	cancelListing	External	✓	onlyOwner isListed
	updateListing	External	✓	onlyOwner isListed
	buyItem	External	✓	nonReentrant isListed
	_buyItem	Private	✓	
	updatePlatformFee	External	✓	onlyOwner
	updatePlatformFeeRecipient	External	✓	onlyOwner
	getCollectorData	External		-
	_cancelListing	Private	✓	
CreathArtTradable	Implementation	ERC721URIStorage, Ownable		
		Public	✓	ERC721
	mint	External	✓	onlyOwner
	burn	External	✓	onlyOwner
	isApproved	Public		-
	isApprovedForAll	Public		-
	_isApprovedOrOwner	Internal		
CreathArtFactory	Implementation	Ownable		
		Public	✓	-
	updateMarketplace	External	✓	onlyOwner

	createNFTContract	External	✓	onlyOwner
	registerTokenContract	External	✓	onlyOwner
	disableTokenContract	External	✓	onlyOwner
Creath	Implementation	ERC721URIStorage, Ownable		
		Public	✓	ERC721
	mint	External	✓	onlyOwner
	exists	External		-
	isApproved	Public		-
	isApprovedForAll	Public		-
	_isApprovedOrOwner	Internal		

Inheritance Graph



Flow Graph



Summary

The Creath project is a comprehensive NFT ecosystem that encompasses a marketplace for buying the listed NFTs, alongside a factory for generating new NFT contracts based on specific parameters. Additionally, it integrates a treasury system to securely manage and distribute funds, ensuring a seamless and secure transaction experience for users within the platform.

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