Cyberscope

Audit Report

Tasks

Aug 2023

Github	https://github.com/L3A-Protocol/daodevt-contracts
Commit	8ed9a7f1fea443d57063b3b422d1a074dc51c7b1
Audited by	© cyberscope

Table of Contents

Review

Repository	https://github.com/L3A-Protocol/daodevt-contracts
Commit	8ed9a7f1fea443d57063b3b422d1a074dc51c7b1

Audit Updates

Initial Audit	08 Aug 2023

Source Files

Filename	SHA256
Tasks/TasksUtils.sol	f2252c3ce0ff866e20eee50a3a940d8b036 30334fdb96aff7cb4f57b6fc2a74e
Tasks/TasksEnsure.sol	aa9fe1c94dd30ae0a3f7bbc22b4b3e7d24 dc576cfa7a5ad6781f7bca30d2967b
Tasks/Tasks.sol	6bcd19f4b50665781f73bd26ad59798dc7f cdd071b718ad0426447c0045c69a1
Tasks/ITasks.sol	e3c7275062c6e5c2753bb7d079fa0407d7 0518938d3ffbadef34cb0ef23a666c
Tasks/Escrow.sol	927b5b750fb95700790689bc75d3604701 68c10c5aa741a3b7b728cedac19f09

Overview

L3atom has introduced a web3 bounty protocol that exhibits functional resemblances to platforms such as Upwork. However, what sets it apart is its distinctive attribute of executing all tasks within the blockchain. This pioneering methodology not only ensures a secure and transparent space for user collaboration but also empowers users to construct a credible portfolio by means of verifiable interactions on the blockchain.

Contracts

Tasks

The Tasks contract manages a task-based system. This contract defines various functionalities related to task creation, application, management, and execution.

A breakdown of the key components and functionalities of the contract:

State Variables

`taskCounter` : An incremental ID for tasks.

`openTasks` , `takenTasks` , `successfulTasks` : Total number of open, taken, and successful tasks.

`tasks` : A mapping that stores task information associated with task IDs.

`escrowImplementation` : The address of the base escrow contract used for cloning.

`disabler` : An address that has the authority to disable the contract.

View Functions

`taskCount()` : Returns the total count of tasks.

`taskStatistics()` : Returns Total tasks (open, taken, successful).

`getTask(uint256 _taskId)` : Retrieves task information by task ID.

`getTasks(uint256[] memory _taskIds)` : Retrieves an array of task information based on an array of task IDs.

`getManagingTasks(...)` : Retrieves an array of tasks managed by a specific address, starting from _fromTaskId.

`getExecutingTasks(...)` : Retrieves an array of tasks executed by a specific address, starting from _fromTaskId.

Task Creation and Management Functions

`createTask(...)` : Creates a new task, initializes its parameters, and handles budget allocation.

`applyForTask(...)` : Allows users to apply for a task by providing metadata and rewards.

`acceptApplications(...)` : Allows the manager to accept applications for a task.

`takeTask(...)` : Marks a task as taken by an executor.

`createSubmission(...)` : Allows the executor to create a submission for a taken task.

`reviewSubmission(...)` : Allows the manager to review a submission and determine its acceptance.

`cancelTask(...)` : Allows the manager to cancel a task or request its cancellation.

`acceptRequest(...)` : Allows the executor to accept a request, such as a task cancellation request.

`executeRequest(...)` : Allows the executor to execute a request.

`extendDeadline(...)` : Allows the manager to extend the deadline of a task.

`increaseBudget(...)` : Allows the manager to increase the budget of a task.

`editMetadata(...)` : Allows the manager to edit the metadata of a task.

Disabling and Refunding:

`disable()` : Disables the contract, preventing further interactions.

`refund(uint256 _taskId)` : Refunds the creator of a task if the contract is disabled.

TasksEnsure

The Tasks contract inherits from the TasksEnsure contract which is used to enforce specific preconditions in the business logic.

Precondition Functions are prefixed with _ensure. They all use the revert statement to throw exceptions if a certain condition is not met. Similar approach can be found in well-known implementations like the Venus Protocol.

The precondition methods consist of:

`_ensureTaskIsOpen` : Ensures that a task is in the "Open" state.

`_ensureTaskIsTaken` : Ensures that a task is in the "Taken" state.

`_ensureTaskNotClosed` : Ensures that a task is not in the "Closed" state.

`_ensureSenderIsManager` : Ensures that the sender of the transaction is the manager of the task.

`_ensureSenderIsExecutor` : Ensures that the sender of the transaction is the executor of the task.

`_ensureRewardEndsWithNextToken` : Ensures that the last reward in an array of rewards has the nextToken property set to true.

`_ensureApplicationExists` : Ensures that an application with a given application ID exists within a task.

`_ensureSenderIsApplicant` : Ensures that the sender of the transaction is the applicant of a specific application.

`_ensureApplicationIsAccepted` : Ensures that a specific application has been accepted.

`_ensureSubmissionExists` : Ensures that a submission with a given submission ID exists within a task.

`_ensureSubmissionNotJudged` : Ensures that a submission has not been judged yet.

`_ensureJudgementNotNone` : Ensures that a submission judgment is not set to "None".

`_ensureCancelTaskRequestExists` : Ensures that a cancel task request with a given request ID exists within a task.

`_ensureRequestNotAccepted` : Ensures that a specific request has not been accepted.

`_ensureRequestAccepted` : Ensures that a specific request has been accepted.

`_ensureRequestNotExecuted` : Ensures that a specific request has not been executed.

TasksUtils

Utility functions aim to encapsulate complex or repetitive logic, making the main contract's (Tasks.sol) code more readable and organized.

`_toOffchainTask` : I takes a storage Task instance and converts it into an OffChainTask struct, making it more suitable for off-chain data representation. This function essentially creates a simplified version of the task data for off-chain use.

`_increaseBudgetToReward` : This function takes a task, reward information, and increases the budget in the escrow to match the total reward amount, considering the case when rewards are split across different budget items.

`_setRewardBellowBudget` : This function sets the rewards for an application while ensuring that the total reward doesn't exceed the available budget.

`_payoutTask` : This function pays out the rewards for a completed task to the executor and updates the task state to "Closed".

`_refundCreator` : This function refunds the remaining budget to the creator when a task is canceled.

Escrow

The escrow contract is used to hold the rewards of the task while the task has not been finished. It is a guarantee between the creator and the executor of the task.

Architecture Comments

Payout/Reward Tokens

ERC20 token manipulation

Within the ecosystem, users proactively define reward tokens to facilitate various transactions. However, the existing contract lacks a mechanism to verify the authenticity of these designated tokens. To improve security, it's advisable for the contract to integrate a validation procedure. In a corner-case scenario, the attacker could create an ERC20 token that looks similar to a well-known token aiming to manipulate the transactions.

The contract could contain some predefined tokens for transactions in the ecosystem. Such measures effectively restrict any prospective manipulation of ERC20 contracts and any inconsistencies.

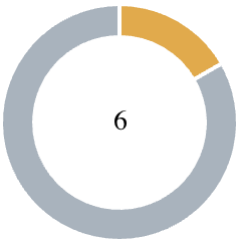
Task Accept request

The current implementation prevents the executor from executing the task once it's accepted. However, it could be modified to enable the executor to proceed with task execution even after its acceptance. This adjustment could be considered, as it wouldn't impact the stability of the ecosystem.

Testing Deploy

Contract Name	Explorer
Tasks	https://mumbai.polygonscan.com/address/0xF63499a77c0d96AE8BEc3367FfCa0fB58AbF8B5a

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	5

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	5	0	0	0

Severity	Code	Description	Status
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	MPS	Metadata Proper Storage	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	PIL	Potential Information Loss	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved

PTAI - Potential Transfer Amount Inconsistency

Criticality	Medium
Location	Tasks/Tasks.sol#L156,546Tasks/TasksUtils.sol#L71
Status	Unresolved

Description

The `transfer()` and `transferFrom()` functions are used to transfer a specified amount of tokens to an address. The fee or tax is an amount that is charged to the sender of an ERC20 token when tokens are transferred to another address. According to the specification, the transferred amount could potentially be less than the expected amount. This may produce inconsistency between the expected and the actual behavior.

The following example depicts the diversion between the expected and actual amount.

Tax	Amount	Expected	Actual
No Tax	100	100	100
10% Tax	100	100	90

```
function createTask(
    string calldata _metadata,
    uint64 _deadline,
    ERC20Transfer[] calldata _budget,
    address _manager,
    PreapprovedApplication[] calldata _preapprove
) external returns (uint256 taskId) {
    ...
    _budget[i].tokenContract.transferFrom(
        _msgSender(),
        address(escrow),
        _budget[i].amount
    );
    ...
}

function increaseBudget(
    uint256 _taskId,
    uint96[] calldata _increase
) external {
    ...
    transfer.tokenContract.transferFrom(
        _msgSender(),
        address(task.escrow),
        _increase[i]
    );
    ...
}

function _increaseBudgetToReward(
    Task storage task,
    uint8 _length,
    mapping(uint8 => Reward) storage _reward
) internal {
    ...
    erc20Transfer.tokenContract.transferFrom(
        _msgSender(),
        address(task.escrow),
        needed - erc20Transfer.amount
    );
   ...
}
```

Recommendation

The team is advised to take into consideration the actual amount that has been transferred instead of the expected.

It is important to note that an ERC20 transfer tax is not a standard feature of the ERC20 specification, and it is not universally implemented by all ERC20 contracts. Therefore, the contract could produce the actual amount by calculating the difference between the transfer call.

```
Actual Transferred Amount = Balance After Transfer - Balance Before Transfer
```

MPS - Metadata Proper Storage

Criticality	Minor / Informative
Status	Unresolved

Description

In scenarios where the contract predominantly employs IPFS hashes, utilizing a string variable to store these hashes can incur high costs. It is advisable to explore the option of storing IPFS hashes in byte32 variables, which could be a more efficient solution.

Recommendation

The team could investigate enumerated approaches where the caller defines the metadata data type, so the contract could validate and store it in a performant way.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	Tasks/Tasks.sol#L156
Status	Unresolved

Description

If the `_manager` is stored as the zero address, then the task will not be able to proceed.

```
function createTask(
    string calldata _metadata,
    uint64 _deadline,
    ERC20Transfer[] calldata _budget,
    address _manager,
    PreapprovedApplication[] calldata _preapprove
) {...}
```

Recommendation

The team is advised to prevent the `_manager` configured with the zero address.

PIL - Potential Information Loss

Criticality	Minor / Informative
Location	Tasks/Tasks.solTasks/TasksUtils.sol
Status	Unresolved

Description

The contract typecasts variables from `uint256` to `uint8` or `uint16`. This is a common practise in order to decrease the gas consuption. If the typecase is not validated correctly, it may lead to the loss of information. This could happen if the narrower data type cannot accommodate the full range of values stored in the original uint256 variable.

For instance, in the corner-case scenario where the reward array length exceeds the size of `256 = 2^8`. The reward entries after the 256th will be lost.

```
task.budgetCount = uint8(_budget.length);
application.rewardCount = uint8(_reward.length);
task.applicationCount = uint16(_preapprove.length);
```

Recommendation

It is recommended to implement proper preconditions before performing any typecasting operations. Before converting a `uint256` variable to a narrower type such as `uint8` or `uint16`, the contract should verify that the value falls within the acceptable range of the target data type. If the value exceeds the valid range, the contract should handle the situation appropriately. Applying these preconditions will prevent information loss and maintain the accuracy and integrity of the contract's operations when typecasting variables.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Tasks/Tasks.sol#L574
Status	Unresolved

Description

The contract updates the variable `task.metadata` even if its current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
function editMetadata(
    uint256 _taskId,
    string calldata _newMetadata
) external {
    _ensureNotDisabled();
    Task storage task = _getTask(_taskId);
    _ensureSenderIsManager(task);

    _ensureTaskIsOpen(task);

    task.metadata = _newMetadata;
    emit MetadataEditted(_taskId, _newMetadata, _msgSender());
}
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Tasks/Tasks.sol#L32
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.

```
escrowImplementation
```

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.

Functions Analysis

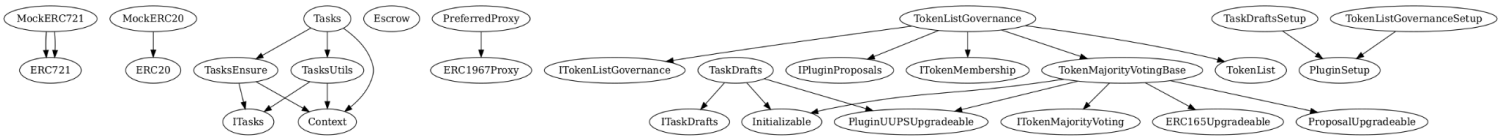
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
TasksUtils	Implementation	ITasks, Context		
	_toOffchainTask	Internal		
	_increaseBudgetToReward	Internal	✓	
	_setRewardBellowBudget	Internal	✓	
	_payoutTask	Internal	✓	
	_refundCreator	Internal	✓	
TasksEnsure	Implementation	ITasks, Context		
	_ensureTaskIsOpen	Internal		
	_ensureTaskIsTaken	Internal		
	_ensureTaskNotClosed	Internal		
	_ensureSenderIsManager	Internal		
	_ensureSenderIsExecutor	Internal		
	_ensureRewardEndsWithNextToken	Internal		
	_ensureApplicationExists	Internal		
	_ensureSenderIsApplicant	Internal		
	_ensureApplicationIsAccepted	Internal		
	_ensureSubmissionExists	Internal		
	_ensureSubmissionNotJudged	Internal		

	_ensureJudgementNotNone	Internal		
	_ensureCancelTaskRequestExists	Internal		
	_ensureRequestNotAccepted	Internal		
	_ensureRequestAccepted	Internal		
	_ensureRequestNotExecuted	Internal		
Tasks	Implementation	Context, TasksEnsure, TasksUtils		
		Public	✓	-
	taskCount	External		-
	taskStatistics	External		-
	getTask	Public		-
	getTasks	Public		-
	getManagingTasks	External		-
	getExecutingTasks	External		-
	createTask	External	✓	-
	applyForTask	External	✓	-
	acceptApplications	External	✓	-
	takeTask	External	✓	-
	createSubmission	External	✓	-
	reviewSubmission	External	✓	-
	cancelTask	External	✓	-
	acceptRequest	External	✓	-
	executeRequest	External	✓	-

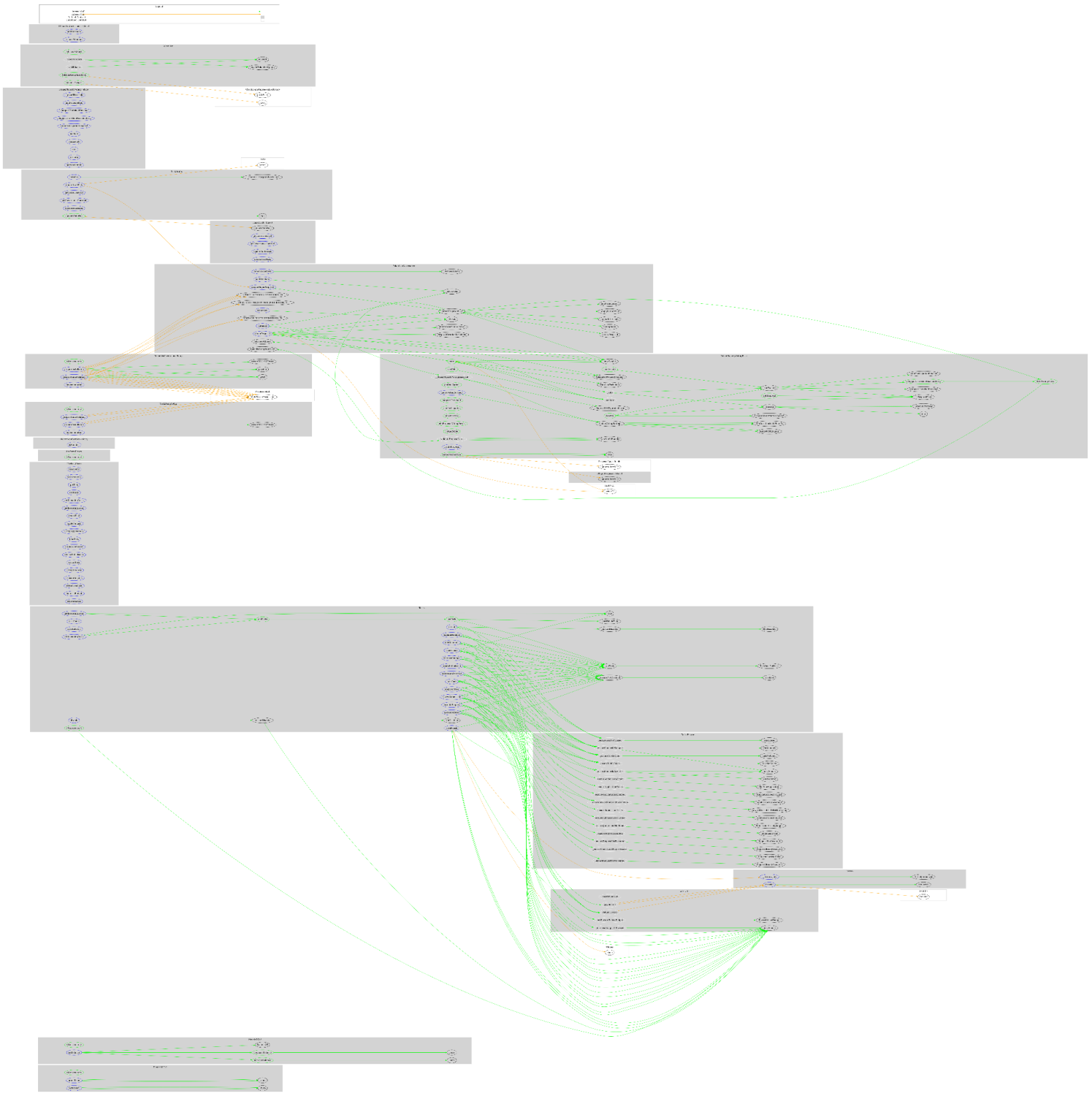
	extendDeadline	External	✓	-
	increaseBudget	External	✓	-
	editMetadata	External	✓	-
	disable	External	✓	-
	refund	External	✓	-
	_getTask	Internal		
	_ensureNotDisabled	Internal		
	_ensureDisabled	Internal		
	_ensureDisabler	Internal		
ITasks	Interface			
	taskCount	External		-
	taskStatistics	External		-
	getTask	External		-
	getTasks	External		-
	getManagingTasks	External		-
	getExecutingTasks	External		-
	createTask	External	✓	-
	applyForTask	External	✓	-
	acceptApplications	External	✓	-
	takeTask	External	✓	-
	createSubmission	External	✓	-
	reviewSubmission	External	✓	-

	cancelTask	External	✓	-
	acceptRequest	External	✓	-
	executeRequest	External	✓	-
	extendDeadline	External	✓	-
	increaseBudget	External	✓	-
	editMetadata	External	✓	-
Escrow	Implementation			
	__Escrow_init	External	✓	-
	transfer	External	✓	-

Inheritance Graph



Flow Graph



Summary

L3atom's web3 is a bounty protocol that executes tasks on the blockchain, ensuring security, transparency, and enabling verifiable user portfolios. 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