Cyberscope

Audit Report

Autonomi

May 2025

Repository	https://github.com/lajosdeme/impossible-futures-contracts
Files	Phase2Voting.sol
Commit	1f9715ef8121f67f196bfe8623f17a78fe03dbe0
Audited by	© cyberscope

Table of Contents

Table of Contents **1**
Risk Classification **3**
Review **4**
Audit Updates 4
Source Files 4
Overview **5**
Locking Tokens for Applications 5
Unlocking Tokens 5
Claiming Rewards 5
Managing the Reward Pool 6
Calculating Shares Over Time 6
Querying User Lock Information 6
Owner Functionalities 7
Findings Breakdown **8**
Diagnostics **9**
PSI - Proportional Shares Inconsistency 10
Description 10
Recommendation 10
RSP - Redundant Struct Property 11
Description 11
Recommendation 11
CCR - Contract Centralization Risk 12
Description 12
Recommendation 12
IDI - Immutable Declaration Improvement 13
Description 13
Recommendation 13
MC - Missing Check 14
Description 14
Recommendation 14
PTAI - Potential Transfer Amount Inconsistency 15
Description 15
Recommendation 16
RDC - Registry Dependency Concern 17
Description 17
Recommendation 17
TSI - Tokens Sufficiency Insurance 18
Description 18
Recommendation 18

UTPD - Unverified Third Party Dependencies 19
Description 19
Recommendation 19
L04 - Conformance to Solidity Naming Conventions 20
Description 20
Recommendation 21
Functions Analysis 22
Inheritance Graph 24
Flow Graph 25
Summary 26
Disclaimer 27
About Cyberscope 28

Risk Classification

The criticality of findings in Cyberscope’s smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Repository	https://github.com/lajosdeme/impossible-futures-contracts
Commit	1f9715ef8121f67f196bfe8623f17a78fe03dbe0

Audit Updates

Initial Audit	13 May 2025

Source Files

Filename	SHA256
Phase2Voting.sol	66d508604ed1c3deed2f97198c2e98b972 b6419e4ae14364ef9b340e504ce44c
interfaces/IPhase2Voting.sol	91952f94c48c45f6995b1b3f3cb714f4f5cf 8da4865f0d5b11196b6e25544c89
interfaces/IAppRegistry.sol	e1282db8e56ee493caedb3d85df415c967 df8abbce33aa031ee4b83453eab178

Overview

The `Phase2Voting` contract is designed to allocate ANT tokens to decentralized applications (apps) through time-decayed token locking. It encourages long-term commitment by requiring users to lock their tokens for over a year, rewarding them based on the performance of the apps they support. Users earn shares by supporting a 4-week campaign. These shares determine the share of a reward pool users will receive if the apps they supported go live. The contract uses an exponential decay function to issue fewer shares over time, motivating early participation.

Locking Tokens for Applications

Users participate by calling the `lockTokens` function with a valid `appId` and an `antAmount`. The app must be registered and in Phase 2 within the `IAppRegistry`. The locking must occur within a 4-week campaign window defined from `START_TIME`. Upon locking, the contract calculates shares based on the current time using a decaying exponential formula. ANT tokens are transferred from the user to the contract, and state variables such as `sharesPerApp`, `antPerApp`, and `userLocks` are updated to track the contribution. An event `LockedTokens` is emitted to record the action.

Unlocking Tokens

After the lock period ends (366 days from the `START_TIME`), users can call `unlockTokens` by specifying the index of their lock. The contract checks the validity of the index and whether the lock was already unlocked. If valid, it marks the lock as unlocked and transfers the originally locked ANT amount back to the user. This ensures users retrieve their tokens after the long-term commitment period. A `UnlockedTokens` event is emitted to log the action.

Claiming Rewards

Users who supported apps that later went live can claim a share of the reward pool using the `claimRewards` function, but only after the unlock time. The function checks that the user has not already claimed and iterates through their locks to determine how many shares were associated with apps that went live. It calculates the user's reward based on their share of the total live-app pool (`totalPoolShares`) and transfers the corresponding ANT amount from the pool to the user. The user is then marked as having claimed, and a `ClaimedRewards` event is emitted.

Managing the Reward Pool

The reward pool, which consists of ANT tokens to be distributed among successful participants, can be increased at any time by calling `increaseTotalRewardPool` . This function allows contributors to add ANT tokens to the pool and emits an `IncreasedTotalRewardPool` event.

The `updateTotalPoolShares` function is restricted to the owner and is used to mark apps that have gone live. Each app is verified through the `IAppRegistry` , and if it is live and not yet counted, its shares are added to `totalPoolShares` . This step finalizes which app shares are eligible for rewards. The function can only be called before the unlock period ends and emits an `UpdatedTotalPoolShares` event.

Calculating Shares Over Time

The number of shares earned per ANT token decreases over time to incentivize early participation. This is managed by the `calculateShares` and `calculateRate` functions. The formula used is:

`shares = antAmount × rate(t)`

`rate(t) = INITIAL_SHARES × exp((t / CAMPAIGN_DURATION)^p × ln(FINAL_SHARES / INITIAL_SHARES))`

Where `p` is the decay exponent, and `t` is the time since the start of the campaign. The result is a non-linear decay in share issuance, meaning users who commit earlier receive more shares for the same ANT amount.

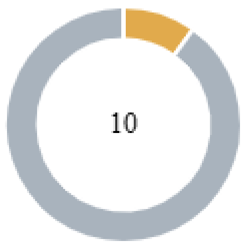
Querying User Lock Information

Users can inspect their lock activity using the `getUserLocks` function, which returns all lock entries for a specific address. Additionally, `getUserLocksLength` returns the total number of lock entries for a user. These functions enable transparency and interface support, allowing users to track their locked tokens and participation history.

Owner Functionalities

The contract includes restricted functionalities for the owner, inherited through the `Ownable` contract. The owner can call `updateTotalPoolShares` to finalize which apps are counted toward reward distribution. The owner has no control over user funds or voting results, but does play a critical role in updating app status tracking and managing reward eligibility, ensuring the process reflects live application data in the registry.

Findings Breakdown



●	Critical	0
●	Medium	1
●	Minor / Informative	9

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	1	0	0	0
●	Minor / Informative	9	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	PSI	Proportional Shares Inconsistency	Unresolved
●	RSP	Redundant Struct Property	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MC	Missing Check	Unresolved
●	PTAI	Potential Transfer Amount Inconsistency	Unresolved
●	RDC	Registry Dependency Concern	Unresolved
●	TSI	Tokens Sufficiency Insurance	Unresolved
●	UTPD	Unverified Third Party Dependencies	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

PSI - Proportional Shares Inconsistency

Criticality	Medium
Location	Phase2Voting.sol#L236,246
Status	Unresolved

Description

Users are able to lock their tokens for a specific app. If `updateTotalPoolShares` has been called before, and `appAddedToTotalPoolShares` of that app is already updated to `true`, then the newly created shares will not be accounted to the `totalPoolShares`. This will allow the user to claim more tokens than the expected amount during the claiming of rewards.

```
for (uint256 i = 0; i < locks.length; i++) {
    if (appAddedToTotalPoolShares[locks[i].selectedApp]) {
        userShareCount += locks[i].shares;
    }
}
...
uint256 rewardAmount = totalRewardPool.mulDiv(userShareCount,
totalPoolShares);
```

Recommendation

The contract should not allow users to lock their tokens for apps that are already calculated to the total pool shares.

RSP - Redundant Struct Property

Criticality	Minor / Informative
Location	Phase2Voting.sol#L149 interfaces/IPhase2Voting.sol#L15
Status	Unresolved

Description

The contract features a struct called `Lock` which has the `locker` variable. `locker` is used to store the caller of the `lockTokens` function. However the `Lock` is also pushed in an array that is mapped to the caller of the function. The contract does not use the `locker` in any of its other functions, therefore it is redundant.

```
userLocks[msg.sender].push(
    Lock({locker: msg.sender, selectedApp: appId, antAmount: antAmount,
shares: shares, unlocked: false})
);
```

```
struct Lock {
    address locker;
    bytes32 selectedApp;
    uint256 antAmount;
    uint256 shares;
    bool unlocked;
}
```

Recommendation

It is recommended to the `locker` variable from the `Lock` struct. This will enhance code readability, and lower execution costs.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	Phase2Voting.sol#L203
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function updateTotalPoolShares(bytes32[] memory appIds) external
onlyOwner
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Phase2Voting.sol#L113
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.

```
p
```

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.

MC - Missing Check

Criticality	Minor / Informative
Location	Phase2Voting.sol#L128,203
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

Specifically, in `lockTokens` the team could ensure that `appId` is a valid app in the registry. Additionally the function does not check if the tokens locked for that `appId` are already accounted for in the `totalPoolShares`.

```
function lockTokens(bytes32 appId, uint256 antAmount) external
```

In `updateTotalPoolShares` a check is missing to ensure that the function is not triggered before the end of the campaign duration.

Additionally, in the same function the team can consider checking if the length of `appIds` array is less than or equal to the size of the state that apps are stored in the registry.

```
function updateTotalPoolShares(bytes32[] memory appIds) external
onlyOwner
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

PTAI - Potential Transfer Amount Inconsistency

Criticality	Minor / Informative
Location	Phase2Voting.sol#L143,190
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

```
ANT.safeTransferFrom(msg.sender, address(this), antAmount);
antPerApp[appId] += antAmount;
userLocks[msg.sender].push(
    Lock({locker: msg.sender, selectedApp: appId, antAmount: antAmount,
shares: shares, unlocked: false})
);
...
ANT.safeTransferFrom(msg.sender, address(this), antAmount);
totalRewardPool += antAmount;
```

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
```

RDC - Registry Dependency Concern

Criticality	Minor / Informative
Location	Phase2Voting.sol#L210
Status	Unresolved

Description

`updateTotalPoolShares` checks each element of the array `appsIds` if they are active. However if authorities of registry can toggle the app's state, then there might be inconsistencies when calculating the `totalPoolShares` .

```
if (!APP_REGISTRY.isLive(appId))
```

Recommendation

It is recommended to carefully consider the interactions with registry to ensure smoothness of operations.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	Phase2Voting.sol#L189
Status	Unresolved

Description

The tokens are not held within the contract itself. Instead, the contract is designed to provide the tokens from an external administrator. While external administration can provide flexibility, it introduces a dependency on the administrator's actions, which can lead to various issues and centralization risks.

```
function increaseTotalRewardPool(uint256 antAmount) external {
    ANT.safeTransferFrom(msg.sender, address(this), antAmount);
    totalRewardPool += antAmount;

    emit IncreasedTotalRewardPool(antAmount);
}
```

Recommendation

It is recommended to consider implementing a more decentralized and automated approach for handling the contract tokens. One possible solution is to hold the tokens within the contract itself. If the contract guarantees the process it can enhance its reliability, security, and participant trust, ultimately leading to a more successful and efficient process.

UTPD - Unverified Third Party Dependencies

Criticality	Minor / Informative
Location	Phase2Voting.sol#L17,19,130,143,178,190,210,252
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

```
IAppRegistry public immutable APP_REGISTRY;
IERC20 public immutable ANT;
```

```
if (!APP_REGISTRY.isRegisteredApp(appId) ||
!APP_REGISTRY.isInPhase2(appId))
...
ANT.safeTransferFrom(msg.sender, address(this), antAmount);
...
ANT.safeTransfer(msg.sender, lock.antAmount);
...
ANT.safeTransferFrom(msg.sender, address(this), antAmount);
...
if (!APP_REGISTRY.isLive(appId))
...
ANT.safeTransfer(msg.sender, rewardAmount);
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Phase2Voting.sol#L17,19,27,29,32,34
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.

```
IAppRegistry public immutable APP_REGISTRY
IERC20 public immutable ANT
uint256 public immutable START_TIME
uint256 public immutable UNLOCK_TIME
uint256 public immutable INITIAL_SHARES
uint256 public immutable FINAL_SHARES
```

Recommendation

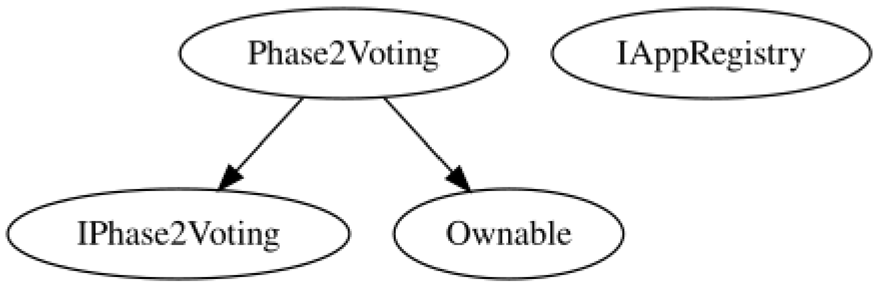
By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions.

Functions Analysis

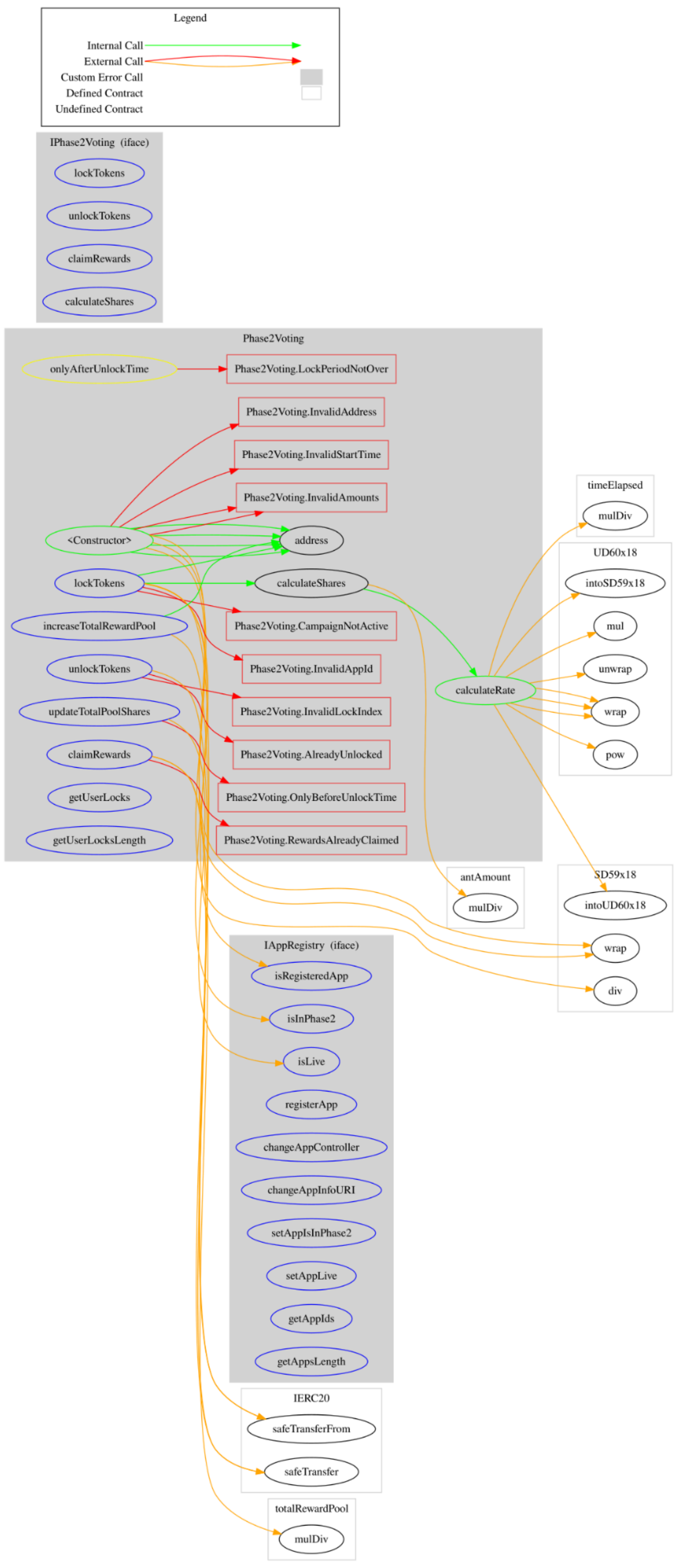
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Phase2Voting	Implementation	IPhase2Voting, Ownable		
		Public	✓	Ownable
	lockTokens	External	✓	-
	unlockTokens	External	✓	onlyAfterUnlockTime
	increaseTotalRewardPool	External	✓	-
	updateTotalPoolShares	External	✓	onlyOwner
	claimRewards	External	✓	onlyAfterUnlockTime
	calculateRate	Public		-
	calculateShares	Public		-
	getUserLocks	External		-
	getUserLocksLength	External		-
IPhase2Voting	Interface			
	lockTokens	External	✓	-
	unlockTokens	External	✓	-
	claimRewards	External	✓	-
	calculateShares	External		-
IAppRegistry	Interface			

	registerApp	External	✓	-
	changeAppController	External	✓	-
	changeAppInfoURI	External	✓	-
	setAppIsInPhase2	External	✓	-
	setAppLive	External	✓	-
	getAppIds	External		-
	getAppsLength	External		-
	isRegisteredApp	External		-
	isInPhase2	External		-
	isLive	External		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
IPhase2Voting (iface)
lockTokens
unlockTokens
claimRewards
calculateShares
Phase2Voting
onlyAfterUnlockTime
Phase2Voting.LockPeriodNotOver
Phase2Voting.InvalidAddress
Phase2Voting.InvalidStartTime
Phase2Voting.InvalidAmounts
<Constructor>
address
lockTokens
calculateShares
increaseTotalRewardPool
Phase2Voting.CampaignNotActive
unlockTokens
Phase2Voting.InvalidAppId
updateTotalPoolShares
Phase2Voting.InvalidLockIndex
claimRewards
Phase2Voting.AlreadyUnlocked
getUserLocks
Phase2Voting.OnlyBeforeUnlockTime
getUserLocksLength
Phase2Voting.RewardsAlreadyClaimed
calculateRate
timeElapsed
mulDiv
UD60x18
intoSD59x18
mul
unwrap
wrap
pow
antAmount
mulDiv
SD59x18
intoUD60x18
wrap
div
IAppRegistry (iface)
isRegisteredApp
isInPhase2
isLive
registerApp
changeAppController
changeAppInfoURI
setAppIsInPhase2
setAppLive
getAppIds
getAppsLength
IERC20
safeTransferFrom
safeTransfer
totalRewardPool
mulDiv

Summary

The Autonomi `Phase2Voting` contract implements a locker and staking mechanism. This audit investigates security issues, business logic concerns, and potential improvements, like optimizing gas usage, to ensure the voting process's robustness, fairness, and cost-effectiveness.

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

cyberscope.io