Cyberscope

Audit Report

Autonomi

March 2025

Repository	https://github.com/lajosdeme/autonomi-claims
Files	Claims.sol, AutonomiNFT.sol
Commit	1d6e4ff78c307097df38ade0930364246ce6430f
Audited by	© cyberscope

Table of Contents

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/autonomi-claims
Commit	1d6e4ff78c307097df38ade0930364246ce6430f

Audit Updates

Initial Audit	28 Feb 2025

Source Files

Filename	SHA256
Claims.sol	96bce9b8f4f96f62f2bb3aaf580bb8e3b47168dd63aeb0e32be671ad93d20dfa
AutonomiNFT.sol	639775908957eacb9f589990244d66ad67de79a2364ad8f0462a03fb0063f126

Overview

Claims Contract

The `Claims` contract facilitates the controlled release and claiming of ANT tokens for holders of Autonomi NFTs through a structured vesting mechanism. The contract defines two key vesting periods, starting from a specific timestamp (`VESTING_START_TIMESTAMP`). The first period releases 50% of the allocated tokens after 90 days, while the second period releases the rest after 180 days. This ensures a gradual distribution of tokens over time.

Claiming Mechanism

The `claim` function allows Autonomi NFT holders to claim their allocated ANT tokens. Holders can claim up to the unlocked amount according to the vesting schedule, but they cannot claim more than their total allocation. The function checks the current claimable amount based on the vesting schedule, and if the requested amount exceeds the available tokens, the transaction is reverted. Additionally, the claim function increments the total amount claimed for each token, preventing over-claiming and ensuring the security of the process. The `getClaimable` function provides a view of the claimable amount for a specific token ID, allowing users to check how much they are entitled to claim based on the vesting schedule. This function helps ensure transparency and allows users to plan their claims accordingly, based on the time that has passed since the vesting started.

Security Measures

The contract employs the nonReentrant modifier in the claim function to prevent reentrancy attacks. This security feature ensures that external calls cannot re-enter the contract and interfere with the ongoing transaction, safeguarding the integrity of the claiming process. It also uses the SafeERC20 library from OpenZeppelin to ensure that ANT token transfers are safe and that no unintended errors or vulnerabilities occur during the token transfer process.

AutonomiNFT Contract

The `AutonomiNFT` contract is responsible for minting and managing the Autonomi NFTs, which grant holders the ability to claim ANT tokens. The contract allows for minting new NFTs with specified ANT token allocations using the mint function. This function takes an address and an allocation for the token ID and mints a new NFT for the specified address, assigning the ANT allocation to the NFT. The contract ensures that each NFT has a unique allocation, granting holders the right to claim a corresponding amount of ANT tokens.

Minting Mechanism

The `mint` and `mintMultiple` functions allow for the minting of NFTs, where each NFT is assigned a specific ANT token allocation. The `mintMultiple` function accepts arrays of addresses and allocations and ensures that both arrays are of the same length. It iterates through the arrays, minting the NFTs and assigning the corresponding token allocations, allowing for efficient batch processing of multiple NFTs.

Modification Mechanism

The `setAntAllocationForTokenId` function enables the contract owner to modify the ANT token allocation for a specific token ID. This function is useful when adjustments need to be made to the token allocation after an NFT has already been minted. It ensures that the allocation is updated securely, and the contract guarantees that only the owner can modify the allocations. The `setAntAllocationsForTokenIds` function allows the contract owner to update the ANT token allocations for multiple token IDs at once.

Token Metadata and URI

The `tokenURI` function generates a unique URI for each token ID, which points to the metadata of the NFT. This function calls the internal `_baseURI` function to get the base URI for the token and appends the token ID to it to form a complete URI for the token's metadata. This ensures that each NFT has a unique metadata URI that can be accessed by users. The `setTokenURI` function allows the contract owner to update the base URI used for generating token URIs. This flexibility ensures that the metadata for the NFTs can be updated as needed, allowing the owner to modify the metadata structure or content at any time.

Roles

Claims Contract

Users

Users (holders of Autonomi NFTs) can interact with the following functions:

- `function claim(uint256 tokenId, uint256 amount)`

Retrieval Functions

The following functions can be used to retrieve information:

- `function getClaimable(uint256 tokenId)`

AutonomiNFT Contract

Owner

The owner of the contract can interact with the following functions:

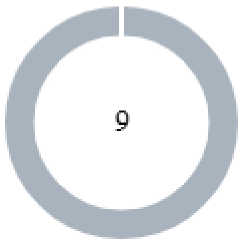
- `function mint(address to, AntAllocation calldata allocation)`
- `function mintMultiple(address[] calldata to, AntAllocation[] calldata allocations)`
- `function setAntAllocationForTokenId(AntAllocation calldata allocation)`
- `function setAntAllocationsForTokenIds(AntAllocation[] calldata allocations)`
- `function setTokenURI(string calldata newTokenURI)`

Retrieval Functions

The following functions can be used to retrieve information:

- `function tokenURI(uint256 tokenId)`
- `function tokenIdToAntAllocation(uint256 tokenId)`

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	9

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

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	MT	Mints Tokens	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	FGO	Function Gas Optimization	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	ZAA	Zero Allocation Allowed	Unresolved
●	MC	Missing Check	Unresolved
●	PACI	Potential Allocation Changes Inconsistencies	Unresolved
●	UTPD	Unverified Third Party Dependencies	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved

MT - Mints Tokens

Criticality	Minor / Informative
Location	AutonomiNFT.sol#L27,32
Status	Unresolved

Description

The contract owner has the authority to mint tokens. The owner may take advantage of it by calling the `mint` or `mintMultiple` functions.

```
function mint(address to, AntAllocation calldata allocation)
external onlyOwner {
    _safeMint(to, allocation.tokenId);
    _setAntAllocationForTokenId(allocation.tokenId,
allocation.antAllocation);
}

function mintMultiple(address[] calldata to, AntAllocation[]
calldata allocations) external onlyOwner {
    require(to.length == allocations.length, "Invalid array
length");
    for (uint256 i = 0; i < allocations.length; i++) {
        _safeMint(to[i], allocations[i].tokenId);
        _setAntAllocationForTokenId(allocations[i].tokenId,
allocations[i].antAllocation);
    }
}
```

Recommendation

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.

Solutions:

- 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.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	AutonomiNFT.sol#L27,32,40,44,61 Claims.sol#L53
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 mint(address to, AntAllocation calldata allocation)
external onlyOwner {}
function mintMultiple(address[] calldata to, AntAllocation[]
calldata allocations) external onlyOwner {}
function setAntAllocationForTokenId(AntAllocation calldata
allocation) external onlyOwner {}
function setAntAllocationsForTokenIds(AntAllocation[] calldata
allocations) external onlyOwner {}
function setTokenURI(string calldata newTokenURI) external
onlyOwner {}
```

Additionally, the `Claims` contract logic works under the assumption that tokens will be held in the contract. However this essentially means that an external party will be responsible for providing the tokens to the contract.

```
ANT_TOKEN.safeTransfer(msg.sender, claimableAmount);
```

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.

FGO - Function Gas Optimization

Criticality	Minor / Informative
Location	AutonomiNFT.sol#L32,44
Status	Unresolved

Description

The `mintMultiple` is using `allocations.length` for the requirement and the `for` logic. Each time `allocations.length` is accessed, Solidity reads from calldata, which incurs a cost.

The case is similar for `setAntAllocationsForTokenIds`.

```
function mintMultiple(address[] calldata to, AntAllocation[]
calldata allocations) external onlyOwner {
    require(to.length == allocations.length, "Invalid array
length");
    for (uint256 i = 0; i < allocations.length; i++) {
        //...
    }
}

function setAntAllocationsForTokenIds(AntAllocation[] calldata
allocations) external onlyOwner {
    for (uint256 i = 0; i < allocations.length; i++) {
        //...
    }
}
```

Recommendation

The team should store `allocations.length` in local variables to reduce the number of times Solidity has to read from calldata, saving gas.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	AutonomiNFT.sol#L61,65
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.

```
function setTokenURI(string calldata newTokenURI) external
onlyOwner {
    _baseTokenURI = newTokenURI;
}
function _setAntAllocationForTokenId(uint256 tokenId, uint256
antAllocation) internal {
    _requireOwned(tokenId);
    tokenIdToAntAllocation[tokenId] = antAllocation;
}
```

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.

ZAA - Zero Allocation Allowed

Criticality	Minor / Informative
Location	AutonomiNFT.sol#L27,32
Status	Unresolved

Description

`mint` and `mintMultiple` functions allow for zero value allocation. Since there is not a reason to mint a token that holds zero allocation, that value should not be allowed.

```
function mint(address to, AntAllocation calldata allocation)
external onlyOwner {}

function mintMultiple(address[] calldata to, AntAllocation[]
calldata allocations) external onlyOwner {}
```

Recommendation

The functions mentioned above could make a check for zero allocation value.

MC - Missing Check

Criticality	Minor / Informative
Location	Claims.sol#L24,37
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.

In the `constructor` of `Claims` contract a check is missing to not allow the address(0) for the `ANT_TOKEN` and `AUTONOMI_NFT`.

```
constructor(IERC20 antToken, IAutonomiNFT autonomiNFT) {
    ANT_TOKEN = antToken;
    AUTONOMI_NFT = autonomiNFT;
}
```

Additionally, in the `claim` function, a check is missing to not allow users to add the value of zero in the `amount`.

```
function claim(uint256 tokenId, uint256 amount) external
nonReentrant {}
```

Recommendation

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

PACI - Potential Allocation Changes Inconsistencies

Criticality	Minor / Informative
Location	AutonomiNFT.sol#L40,44 Claims.sol#L62
Status	Unresolved

Description

The owner of `AutonomiNFT` is able to change the allocation value of tokens by using the `setAntAllocationForTokenId` and `setAntAllocationsForTokenIds`. This can produce inconsistencies when tokens are claimed from the `Claims::claim` function.

For example if the nft holder claims half of the tokens after `VESTING_PERIOD_1` and then the allocation for that token changes to a value lower than half, the function will keep reverting since the `claimableAmount` will be less than zero.

```
function setAntAllocationForTokenId(AntAllocation calldata
allocation) external onlyOwner {}

function setAntAllocationsForTokenIds(AntAllocation[] calldata
allocations) external onlyOwner {}

function _claimable(uint256 tokenId) private view returns
(uint256) {
    uint256 totalAllocation =
AUTONOMI_NFT.tokenIdToAntAllocation(tokenId);

    uint256 alreadyClaimed =
totalAntClaimedForTokenId[tokenId];

    uint256 elapsedTime = block.timestamp -
VESTING_START_TIMESTAMP;
    uint256 claimableAmount = 0;

    if (elapsedTime >= VESTING_PERIOD_2) {
        claimableAmount = totalAllocation - alreadyClaimed;
    } else if (elapsedTime >= VESTING_PERIOD_1) {
        claimableAmount = (totalAllocation * 50) / 100 -
alreadyClaimed;
    }
    return claimableAmount;
}
```

Recommendation

The team should consider adding checks to the `claim` function to ensure that it will revert from situations such as the ones mentioned above and provide the corresponding error message that clearly informs users of the specific reason for the failure.

UTPD - Unverified Third Party Dependencies

Criticality	Minor / Informative
Location	Claims.sol#L53
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.

```
ANT_TOKEN.safeTransfer(msg.sender, claimableAmount);
```

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	Claims.sol#L21,22
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.

```
IERC20 public immutable ANT_TOKEN
IAutonomiNFT public immutable AUTONOMI_NFT
```

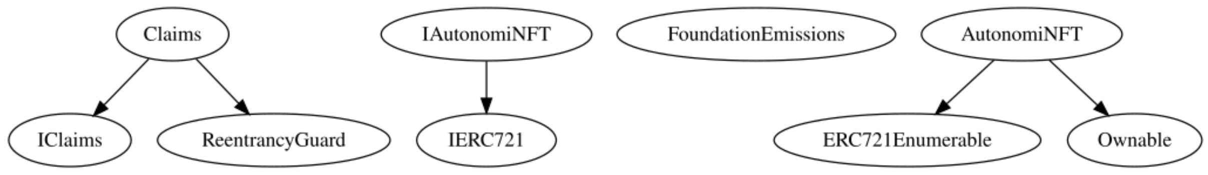
Recommendation

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.

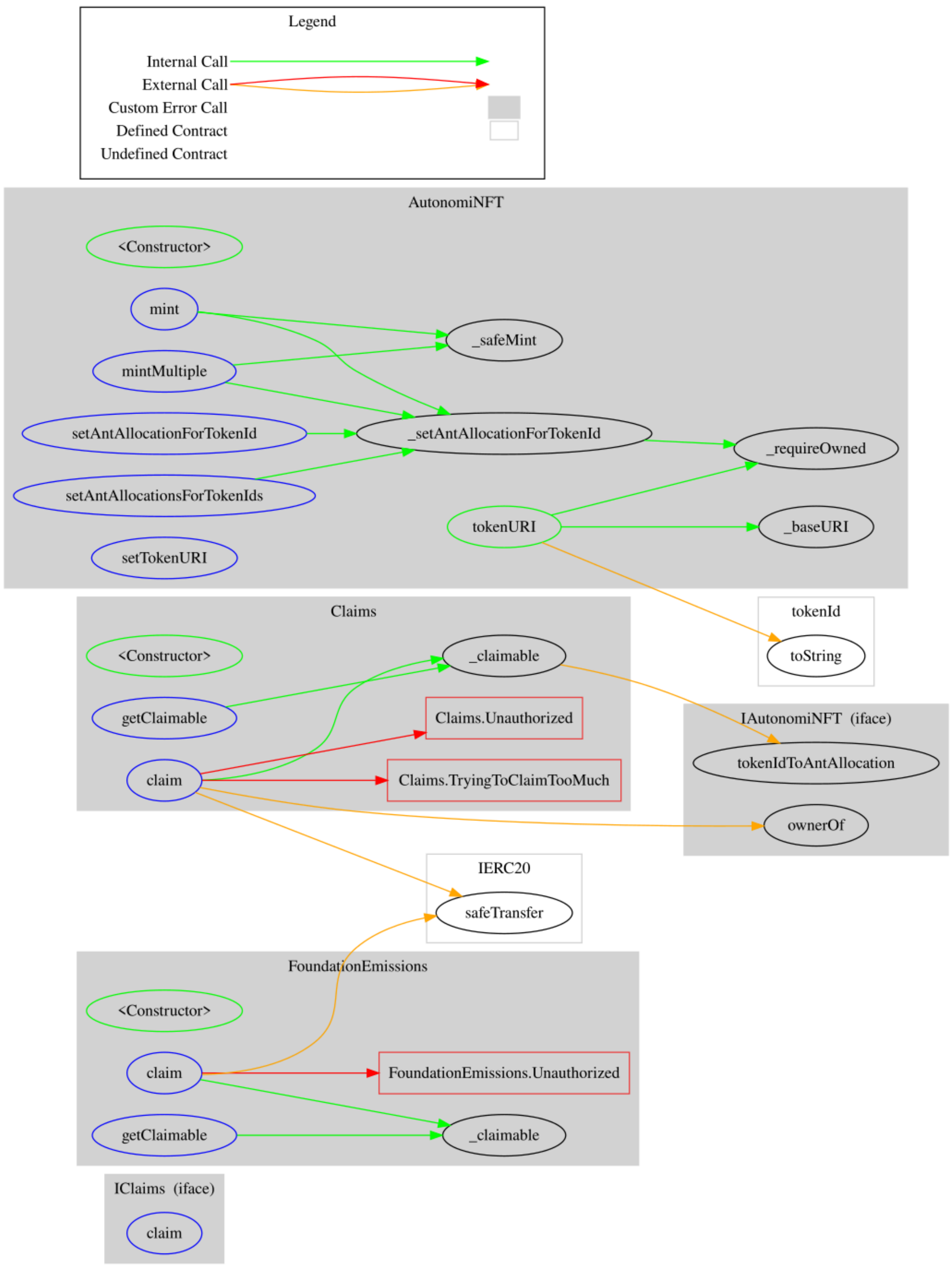
Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Claims	Implementation	IClaims, ReentrancyGuard		
		Public	✓	-
	claim	External	✓	nonReentrant
	getClaimable	External		-
	_claimable	Private		
AutonomiNFT	Implementation	ERC721Enumerable, Ownable		
		Public	✓	ERC721 Ownable
	mint	External	✓	onlyOwner
	mintMultiple	External	✓	onlyOwner
	setAntAllocationForTokenId	External	✓	onlyOwner
	setAntAllocationsForTokenIds	External	✓	onlyOwner
	_baseURI	Internal		
	tokenURI	Public		-
	setTokenURI	External	✓	onlyOwner
	_setAntAllocationForTokenId	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
AutonomiNFT
<Constructor>
mint
mintMultiple
setAntAllocationForTokenId
setAntAllocationsForTokenIds
setTokenURI
_safeMint
_setAntAllocationForTokenId
_requireOwned
tokenURI
_baseURI
tokenId
toString
Claims
<Constructor>
getClaimable
claim
_claimable
Claims.Unauthorized
Claims.TryingToClaimTooMuch
IAutonomiNFT (iface)
tokenIdToAntAllocation
ownerOf
IERC20
safeTransfer
FoundationEmissions
<Constructor>
claim
getClaimable
FoundationEmissions.Unauthorized
_claimable
IClaims (iface)
claim

Summary

Autonomi contract implements a nft and vesting mechanism. This audit investigates security issues, business logic concerns and potential improvements. The Smart Contract analysis reported no compiler error or critical issues.

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