Audit Report

Exohood Labs

February 2024

Network	ETH
Address	0xbd8005612124dc30601e22d8b5d188a89767c640
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	TSD	Total Supply Diversion	Unresolved
●	CO	Code Optimization	Unresolved
●	MEM	Misleading Error Messages	Unresolved
●	MEE	Misleading Event Emission	Unresolved
●	RCC	Redundant Constant Checks	Unresolved
●	RM	Redundant Modifier	Unresolved
●	UIC	Unnecessary Initialization Check	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	MainToken
Compiler Version	v0.4.24+commit.e67f0147
Optimization	200 runs
Explorer	https://etherscan.io/address/0xbd8005612124dc30601e22d8b5d188a89767c640
Address	0xbd8005612124dc30601e22d8b5d188a89767c640
Network	ETH
Symbol	EXO
Decimals	18
Total Supply	2,000,000,000

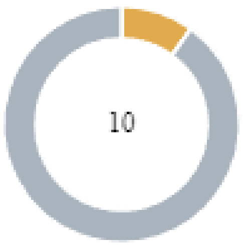
Audit Updates

Initial Audit	30 Jan 2024

Source Files

Filename	SHA256
MainToken.sol	fad01edcf3a17ad033468dd0f343cd6890e2f1b4b5a3987ab63474728bfb1a63

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

TSD - Total Supply Diversion

Criticality	Medium
Location	MainToken.sol#L462
Status	Unresolved

Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

In the `freezeTo` function, tokens are transferred from the sender's balance to a frozen state, effectively reducing the sender's balance. However, this operation overlooks the adjustment of the total supply, leading to a situation where the sum of balances is diverse from the total supply.

```
function freezeTo(address _to, uint _amount, uint64 _until)
public {
    require(_to != address(0));
    require(_amount <= balances[msg.sender]);

    balances[msg.sender] = balances[msg.sender].sub(_amount);

    bytes32 currentKey = toKey(_to, _until);
    freezings[currentKey] = freezings[currentKey].add(_amount);
    freezingBalance[_to] = freezingBalance[_to].add(_amount);

    freeze(_to, _until);
    emit Transfer(msg.sender, _to, _amount);
    emit Freezed(_to, _until, _amount);
}
```

Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply.

CO - Code Optimization

Criticality	Minor / Informative
Location	MainToken.sol#L726
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

Specifically, several inefficiencies and unnecessary operations have been identified around the freeze functionality in the `init` function that impact the contract's overall efficiency and clarity. The function initializes three arrays, `addresses` , `amounts` , and `freezes` ,with a single entry each and proceeds to iterate over the `addresses` array using a `for` loop. Given that the length of this array is statically defined as one, the `for` loop serves no practical purpose and introduces an unnecessary iteration mechanism for a single-operation scenario. Moreover, the check `if (freezes[i] == 0)` within the loop is redundant since the `freezes` array is statically initialized with a zero value, ensuring that the condition is always met, rendering the else branch `mintAndFreeze` unreachable under current logic.

```
address[1] memory addresses =
[address(0x4A53274e7c88a6E6317B9285BDA1F7D5faacB9F9)];
uint[1] memory amounts = [uint(0)];
uint64[1] memory freezes = [uint64(0)];

for (uint i = 0; i < addresses.length; i++) {
    if (freezes[i] == 0) {
        mint(addresses[i], amounts[i]);
    } else {
        mintAndFreeze(addresses[i], amounts[i], freezes[i]);
    }
}
```

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.

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	MainToken.sol#L112,113,179,180,181,305,330,351,356,463,464,483,484, 529,580,608,616,707,712,718
Status	Unresolved

Description

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

```
require(_to != address(0))
require(_value <= balances[msg.sender])
require(_value <= balances[_from])
require(_value <= allowed[_from][msg.sender])
require(msg.sender == owner)
require(_newOwner != address(0))
require(!mintingFinished)
require(_amount <= balances[msg.sender])
require(head != 0)
require(uint64(block.timestamp) > head)
require(_until > block.timestamp)
require(_value <= balances[_who])
require(!paused)
require(paused)

...
```

Recommendation

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

MEE - Misleading Event Emission

Criticality	Minor / Informative
Location	MainToken.sol#L473
Status	Unresolved

Description

Within the `freezeTo` function, the `Transfer` event is emitted following the adjustment of balances due to the freezing of tokens, indicating a transfer of _amount tokens from the sender (msg.sender) to the recipient (_to). However, this emission can be misleading because, in the operational context of this function, tokens are not transferred in the standard sense. Instead, the tokens are frozen, hence creating a discrepancy between the implications of the `Transfer` event and the actual state of the tokens (frozen), which could lead to confusion.

```
function freezeTo(address _to, uint _amount, uint64 _until) public {
    require(_to != address(0));
    require(_amount <= balances[msg.sender]);

    balances[msg.sender] = balances[msg.sender].sub(_amount);

    bytes32 currentKey = toKey(_to, _until);
    freezings[currentKey] = freezings[currentKey].add(_amount);
    freezingBalance[_to] = freezingBalance[_to].add(_amount);

    freeze(_to, _until);
    emit Transfer(msg.sender, _to, _amount);
    emit Freezed(_to, _until, _amount);
}
```

Recommendation

It is recommended to re-evaluate the emission of the `Transfer` event within the context of freezing tokens. If the emission of an event is necessary in order to track transactions, a new event could be introduced to explicitly indicate that tokens have been frozen. This approach would preserve the integrity of transaction tracking and enhance the contract's transparency.

RCC - Redundant Constant Checks

Criticality	Minor / Informative
Location	MainToken.sol#L721,739
Status	Unresolved

Description

The `MainToken` contract incorporates checks for two constants, `PAUSED` and `CONTINUE_MINTING` , within its initialization logic to conditionally trigger actions based on these values. However, these constants are defined with immutable values that negate the necessity for such conditional checks. Specifically, `PAUSED` is permanently set to `false` , and `CONTINUE_MINTING` is `true` , rendering the associated conditionals operationally useless.

```
bool public constant PAUSED = false;
bool public constant CONTINUE_MINTING = true;

if (PAUSED) {
    pause();
}

if (!CONTINUE_MINTING) {
    finishMinting();
}
```

Recommendation

It is recommended to remove conditional statements that evaluate immutable constants, especially when such evaluations cannot result in varied execution paths. This will improve the code's clarity and gas efficiency.

RM - Redundant Modifier

Criticality	Minor / Informative
Location	MainToken.sol#L355
Status	Unresolved

Description

The `MintableToken` contract defines a modifier named `hasMintPermission`, which imposes a requirement that the caller must be the contract's owner. This modifier performs a check to ensure that `msg.sender == owner`, mirroring the functionality provided by the `onlyOwner` modifier inherited from the `Ownable` contract. The existence of the `hasMintPermission` modifier alongside the inherited `onlyOwner` modifier leads to unnecessary duplication of code and functionality and increased gas costs for deployment of the contract.

```
modifier hasMintPermission() {
     require(msg.sender == owner);
     _;
}
```

Recommendation

It is recommended to remove the `hasMintPermission` modifier from the `MintableToken` contract to streamline the contract's code and utilize the `onlyOwner` modifier for enforcing ownership-based access control. This change will reduce the contract's complexity and will result in lower gas costs associated with contract deployment.

UIC - Unnecessary Initialization Check

Criticality	Minor / Informative
Location	MainToken.sol#L717
Status	Unresolved

Description

The `MainToken` smart contract employs an initialization pattern characterized by a private `init` function that is exclusively called within the contract's constructor. This function includes a preliminary check against an `initialized` state variable, immediately setting this variable to true thereafter. This pattern ostensibly aims to guard against multiple initializations. Given the `init` function's private visibility and its singular invocation within the constructor, the initialization check and the subsequent setting of the initialized variable emerge as redundant operations. This redundancy introduces unnecessary complexity and gas inefficiencies in the contract deployment process.

```
function init() private {
    require(!initialized);
    initialized = true;
    ...
```

Recommendation

It is advisable to streamline the initialization process by eliminating the `initialized` state variable and its associated check within the `init` function. Simplifying the initialization sequence in this manner would reduce gas costs during deployment and enhance the contract's clarity.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MainToken.sol#L111,126,172,173,174,200,213,214,234,235,257,257,321,367,368,410,419,423,431,444,462,528,575,648,706,711
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.

```
uint256 _value
address _to
address _owner
address _from
address _spender
uint _addedValue
uint _subtractedValue
address _newOwner
uint256 _amount
address _addr
uint _index
uint _amount
uint64 _until
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	MainToken.sol#L522
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 {
            result := or(result, mul(_addr,
0x10000000000000000))
            result := or(result, and(_release,
0xffffffffffffffff))
        }
    }
```

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	MainToken.sol#L19
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 ^0.4.23;
```

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

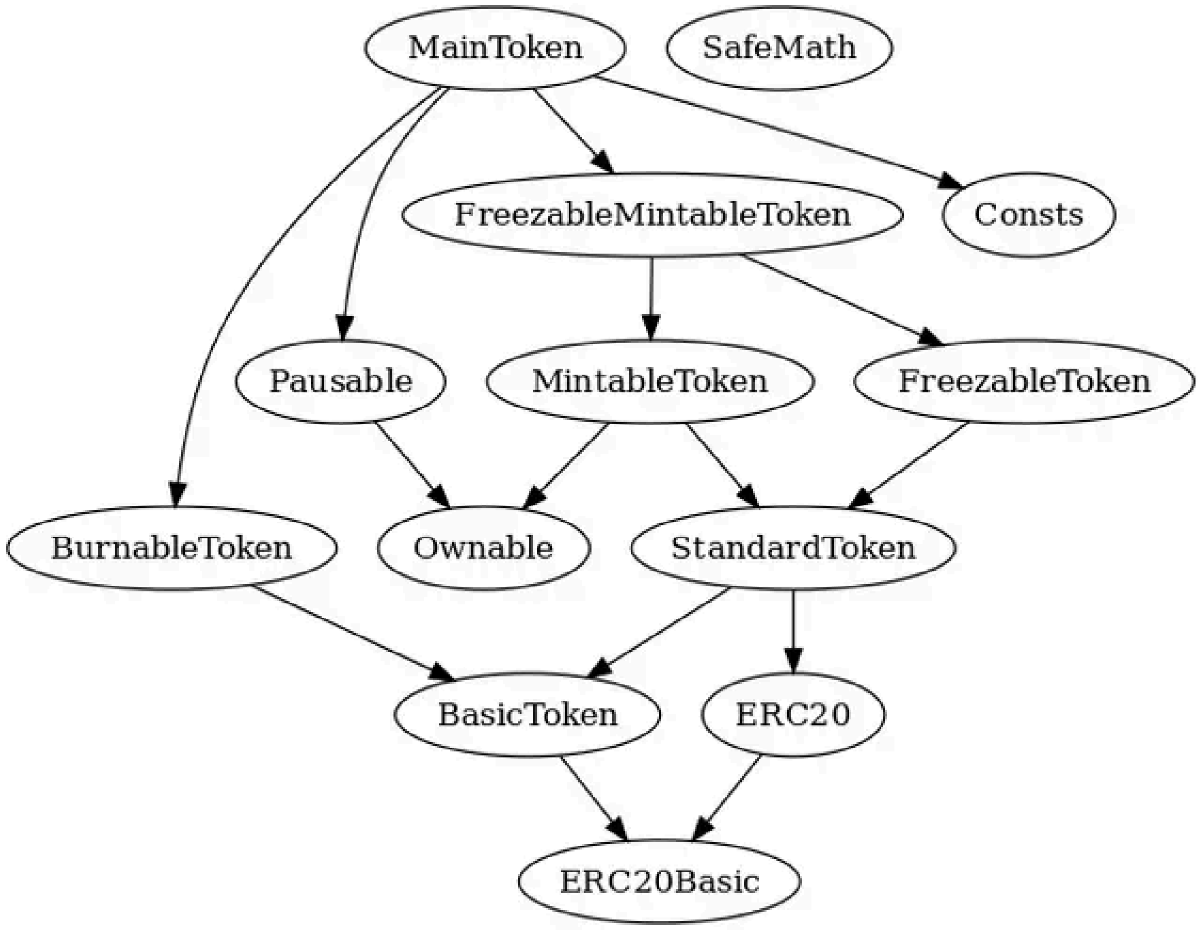
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
ERC20Basic	Implementation			
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
SafeMath	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
BasicToken	Implementation	ERC20Basic		
	totalSupply	Public		-
	transfer	Public	✓	-
	balanceOf	Public		-
ERC20	Implementation	ERC20Basic		
	allowance	Public		-
	transferFrom	Public	✓	-

	approve	Public	✓	-
StandardToken	Implementation	ERC20, BasicToken		
	transferFrom	Public	✓	-
	approve	Public	✓	-
	allowance	Public		-
	increaseApproval	Public	✓	-
	decreaseApproval	Public	✓	-
Ownable	Implementation			
		Public	✓	-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
MintableToken	Implementation	StandardToken, Ownable		
	mint	Public	✓	hasMintPermission canMint
	finishMinting	Public	✓	onlyOwner canMint
FreezableToken	Implementation	StandardToken		
	balanceOf	Public		-
	actualBalanceOf	Public		-

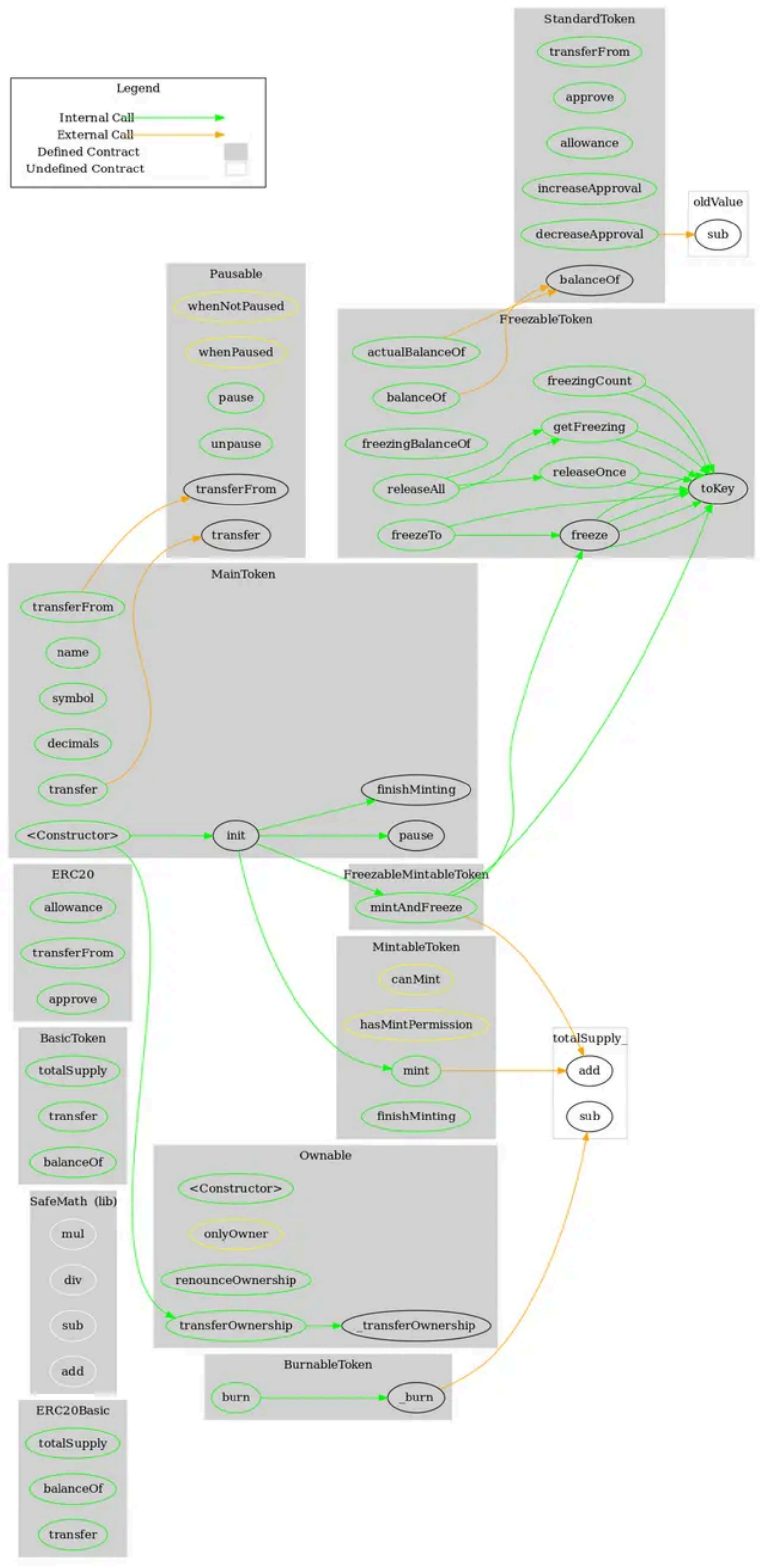
	freezingBalanceOf	Public		-
	freezingCount	Public		-
	getFreezing	Public		-
	freezeTo	Public	✓	-
	releaseOnce	Public	✓	-
	releaseAll	Public	✓	-
	toKey	Internal		
	freeze	Internal	✓	
BurnableToken	Implementation	BasicToken		
	burn	Public	✓	-
	_burn	Internal	✓	
Pausable	Implementation	Ownable		
	pause	Public	✓	onlyOwner whenNotPaused
	unpause	Public	✓	onlyOwner whenPaused
FreezableMintableToken	Implementation	FreezableToken, MintableToken		
	mintAndFreeze	Public	✓	onlyOwner canMint
Consts	Implementation			

MainToken	Implementation	Consts, FreezableMintableToken, BurnableToken, Pausable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	transferFrom	Public	✓	-
	transfer	Public	✓	-
	init	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
StandardToken
transferFrom
approve
allowance
increaseApproval
decreaseApproval
balanceOf
oldValue
sub
Pausable
whenNotPaused
whenPaused
pause
unpause
transferFrom
transfer
FreezableToken
actualBalanceOf
balanceOf
freezingBalanceOf
releaseAll
freezeTo
freezingCount
getFreezing
releaseOnce
freeze
toKey
MainToken
transferFrom
name
symbol
decimals
transfer
<Constructor>
init
finishMinting
pause
ERC20
allowance
transferFrom
approve
FreezableMintableToken
mintAndFreeze
MintableToken
canMint
hasMintPermission
mint
finishMinting
totalSupply_
add
sub
BasicToken
totalSupply
transfer
balanceOf
SafeMath (lib)
mul
div
sub
add
Ownable
<Constructor>
onlyOwner
renounceOwnership
transferOwnership
_transferOwnership
BurnableToken
burn
_burn
ERC20Basic
totalSupply
balanceOf
transfer

Summary

Exohood Labs contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Exohood Labs is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract's ownership has been renounced.

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