Cyberscope

A TAC Security Company

Audit Report

Chrema Coin

July 2025

Network ETH

Address 0x9ac4eE539403e3f101B9Ae3620926F2dEd0D0b99

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
Medium	UTM	Unlock Time Manipulation	Unresolved
Minor / Informative	CCR	Contract Centralization Risk	Unresolved
Minor / Informative	DSV	Deprecated Solidity Version	Unresolved
Minor / Informative	MEM	Missing Error Messages	Unresolved
Minor / Informative	MU	Modifiers Usage	Unresolved
Minor / Informative	NWES	Nonconformity with ERC-20 Standard	Unresolved
Minor / Informative	UAR	Unexcluded Address Restrictions	Unresolved
Minor / Informative	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor / Informative	L09	Dead Code Elimination	Unresolved
Minor / Informative	L19	Stable Compiler Version	Unresolved

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

Contract Name	CHREMACOIN
Compiler Version	v0.4.26+commit.4563c3fc
Optimization	200 runs
Explorer	https://etherscan.io/address/0x9ac4ee539403e3f101b9ae3620926f2ded0d0b99
Address	0x9ac4ee539403e3f101b9ae3620926f2ded0d0b99
Network	ETH
Symbol	CRMC
Decimals	8
Total Supply	50,000,000

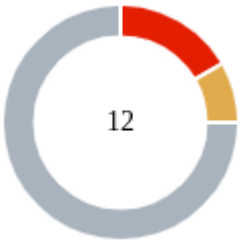
Audit Updates

Initial Audit	03 Jul 2025

Source Files

Filename	SHA256
CHREMACOIN.sol	10a43f6e6d1edf8b74dff26b3f5246460b9b1acfedff2faf2f44925cbb84d632

Findings Breakdown



Critical	2
Medium	1
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	1	0	0	0
Minor / Informative	9	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	CHREMACOIN.sol#L495
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by invoking the `pause` method. As a result, the contract may operate as a honeypot.

```
function transfer( address _to, uint256 _value ) public whenNotPaused returns
(bool) {
require(msg.sender != _to,"Check your address!!");
return super.transfer(_to, _value);
}
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- 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.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

BC - Blacklists Addresses

Criticality	Critical
Location	CHREMACOIN.sol#L484,609
Status	Unresolved

Description

The contract owner and supervisors have the authority to stop addresses from transactions. They may take advantage of it by calling the `setLockedWalletEntity` function.

```
function setLockedWalletEntity(address _address) onlySupervisor public returns
(bool){
require(_address !=address(0) && !lockedWalletEntity[_address]);
lockedWalletEntity[_address] = true;
emit PrintLog(_address, "isLockedWalletEntity",  1);
return true;
}
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- 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.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

UTM - Unlock Time Manipulation

Criticality	Medium
Location	CHREMACOIN.sol#L554
Status	Unresolved

Description

The setLockTime function allows the supervisor to set or modify user unlock times (firstUnlockTime, secondUnlockTime, thirdUnlockTime) in a non-sequential manner. This enables a scenario where the supervisor first sets a very high thirdUnlockTime, and then later increases secondUnlockTime and firstUnlockTime close to that value in reverse order. As a result, unlock times can be indefinitely extended or manipulated after the fact, breaking the intended progressive time-lock mechanism and undermining trust in the locking schedule.

```
function setLockTime(address _to, uint _time, uint256 _lockTime)
onlySupervisor public returns(bool){
require(_to !=address(0) && _time > 0 && _time < 4 && _lockTime > now);
(uint firstUnlockTime,
uint secondUnlockTime,
uint thirdUnlockTime
) = getLockedTimeUserInfo(_to);
...
}
```

Recommendation

The team is recommended to enforce strict checks when setting unlock times to impose limits on the assigned variables. This will maintain the intended progressive time-lock order and prevent indefinite or manipulative extensions.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	CHREMACOIN.sol#L514,559,614,621
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 setLockTime(address _to, uint _time, uint256 _lockTime)
onlySupervisor public returns(bool){...}
function transferToLockedBalance(
address _to,
uint _firstUnlockTime,
uint256 _firstUnlockValue,
uint _secondUnlockTime,
uint256 _secondUnlockValue,
uint _thirdUnlockTime,
uint256 _thirdUnlockValue
) onlySupervisor whenNotPaused public returns (bool) {...}
function setLockedWalletEntity(address _address) onlySupervisor public returns
(bool){...}
function removeLockedWalletEntity(address _address) onlySupervisor public
returns (bool){...}
```

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.

DSV - Deprecated Solidity Version

Criticality	Minor / Informative
Location	CHREMACOIN.sol#L1
Status	Unresolved

Description

The contract uses Solidity version `^0.4.24`, which is deprecated and no longer supported by modern tooling, security analysis frameworks, or best practices. Using outdated compiler versions increases the risk of vulnerabilities due to missing security features, bug fixes, and incompatibility with newer language enhancements.

```
pragma solidity ^0.4.24;
```

Recommendation

It is recommended to consider upgrading to the latest solidity version. This will ensure the contract benefits from the latest security features, language improvements, and broader ecosystem compatibility.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	CHREMACOIN.sol#L43,52,125,126,191,296,320,343,351,460,489,556,581,597,604,611,618,625
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to 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(msg.sender == owner)
require(newOwner != address(0))
require(_to != address(0))
require(_value <= balances[_from])
require(_value <= allowed[_from][msg.sender])
require(_value <= balances[_who])
require(!paused)
require(paused)
require(owner == msg.sender || supervisorEntity[msg.sender])
require(availableValue > 0)
require(_to !=address(0) && _time > 0 && _time < 4 && _lockTime > now)
require(msg.sender == _address || msg.sender == owner ||
supervisorEntity[msg.sender])
require(_address !=address(0) && !supervisorEntity[_address])
require(_address !=address(0) && supervisorEntity[_address])

...
```

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.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	CHREMACOIN.sol#L596,603,609,616
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(_address !=address(0))
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

NWES - Nonconformity with ERC-20 Standard

Criticality	Minor / Informative
Location	CHREMACOIN.sol#L489
Status	Unresolved

Description

The contract does not fully conform to the ERC20 Standard. Specifically, according to the standard, transfers of 0 values must be treated as normal transfers and fire the Transfer event. However, the contract implements a conditional check that prohibits transfers of 0 values. This discrepancy between the contract's implementation and the ERC20 standard may lead to inconsistencies and incompatibilities with other contracts.

```
require(availableValue > 0);
```

Recommendation

The incorrect implementation of the ERC20 standard could potentially lead to problems when interacting with the contract, as other contracts or applications that expect the ERC20 interface may not behave as expected. The team is advised to review and revise the implementation of the transfer mechanism to ensure full compliance with the ERC20 standard. https://eips.ethereum.org/EIPS/eip-20.

UAR - Unexcluded Address Restrictions

Criticality	Minor / Informative
Location	CHREMACOIN.sol#L514
Status	Unresolved

Description

The contract incorporates operational restrictions on transactions, which can hinder seamless interaction with decentralized applications (dApps) such as launchpads, presales, lockers, or staking platforms. In scenarios where an external contract, such as a launchpad factory or a liquidity pool, needs to integrate with the contract, it should be exempt from the limitations to ensure uninterrupted service and functionality. Failure to provide such exemptions can block the successful process and operation of services reliant on this contract.

```
function transferToLockedBalance(
address _to,
uint _firstUnlockTime,
uint256 _firstUnlockValue,
uint _secondUnlockTime,
uint256 _secondUnlockValue,
uint _thirdUnlockTime,
uint256 _thirdUnlockValue
) onlySupervisor whenNotPaused public returns (bool) {...}
```

Recommendation

It is advisable to modify the contract by incorporating functionality that enables the exclusion of designated addresses from transactional restrictions. This enhancement will allow specific addresses, such as those associated with decentralized applications (dApps) and service platforms, to operate without being hindered by the standard constraints imposed on other users. Implementing this feature will ensure smoother integration and functionality with external systems, thereby expanding the contract's versatility and effectiveness in diverse operational environments.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	CHREMACOIN.sol#L120,139,184,185,186,205,218,219,238,239,260,261,291,319,379,380,390,391,392,402,403,413,414,424,425,445,495,501,506,511,512,513,514,515,516,517,555,580,596,603,610,617,624,637,641
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
uint256 _addedValue
uint256 _subtractedValue
uint _addedValue
uint _subtractedValue

...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	CHREMACOIN.sol#L633
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 isSupervisor() view onlyOwner private returns (bool){
        return supervisorEntity[msg.sender];
    }
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	CHREMACOIN.sol#L1
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 ^0.4.24;
```

Recommendation

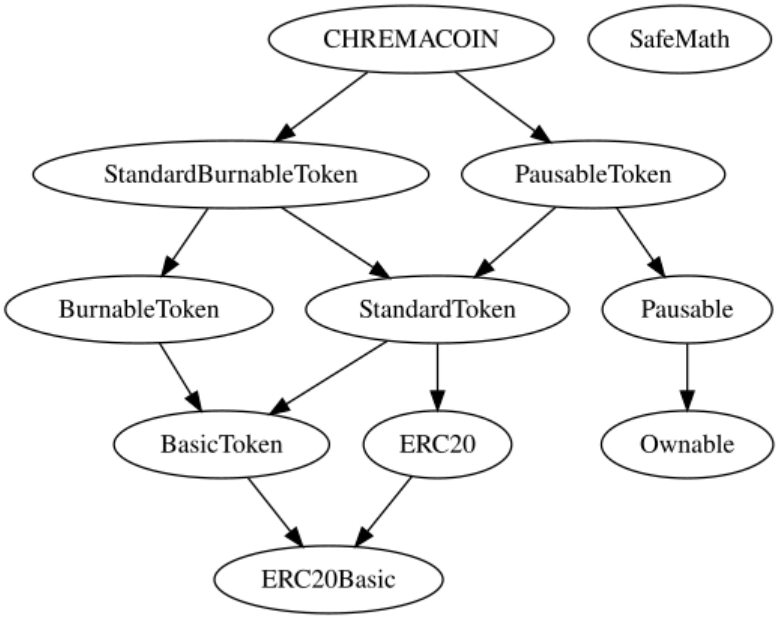
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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

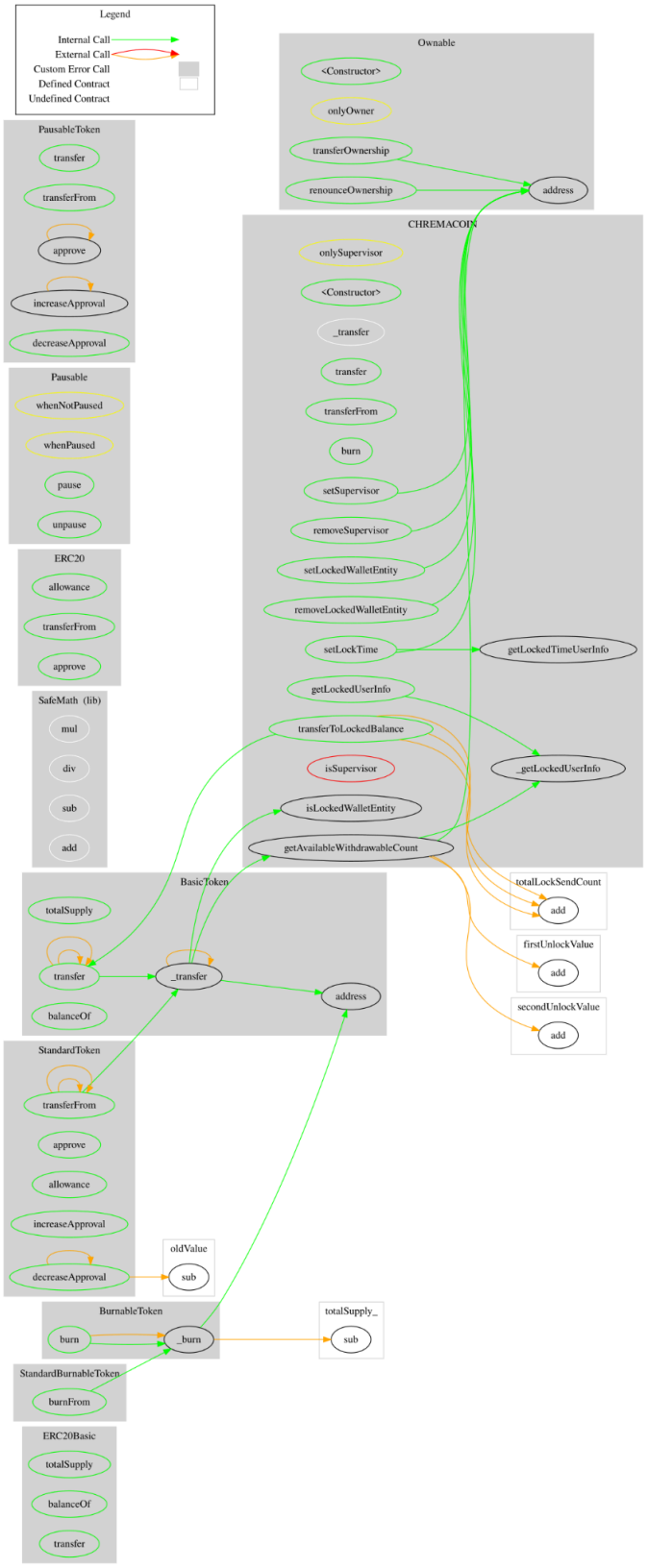
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
CHREMACOIN	Implementation	StandardBurnableToken, PausableToken		
		Public	✓	-
	_transfer	Internal	✓	
	transfer	Public	✓	whenNotPaused
	transferFrom	Public	✓	whenNotPaused
	burn	Public	✓	onlySupervisor
	transferToLockedBalance	Public	✓	onlySupervisor whenNotPaused
	setLockTime	Public	✓	onlySupervisor
	getLockedUserInfo	Public		-
	_getLockedUserInfo	Internal		
	setSupervisor	Public	✓	onlyOwner
	removeSupervisor	Public	✓	onlyOwner
	setLockedWalletEntity	Public	✓	onlySupervisor
	removeLockedWalletEntity	Public	✓	onlySupervisor
	getLockedTimeUserInfo	Private		
	isSupervisor	Private		onlyOwner
	isLockedWalletEntity	Private		

	getAvailableWithdrawableCount	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
Ownable
<Constructor>
onlyOwner
transferOwnership
renounceOwnership
address
PausableToken
transfer
transferFrom
approve
increaseApproval
decreaseApproval
CHREMACOIN
onlySupervisor
<Constructor>
_transfer
transfer
transferFrom
burn
setSupervisor
removeSupervisor
setLockedWalletEntity
removeLockedWalletEntity
setLockTime
getLockedTimeUserInfo
getLockedUserInfo
transferToLockedBalance
isSupervisor
_getLockedUserInfo
isLockedWalletEntity
getAvailableWithdrawableCount
Pausable
whenNotPaused
whenPaused
pause
unpause
ERC20
allowance
transferFrom
approve
SafeMath (lib)
mul
div
sub
add
BasicToken
totalSupply
transfer
_transfer
balanceOf
address
totalLockSendCount
add
firstUnlockValue
add
secondUnlockValue
add
StandardToken
transferFrom
approve
allowance
increaseApproval
decreaseApproval
oldValue
sub
BurnableToken
burn
_burn
totalSupply_
sub
StandardBurnableToken
burnFrom
ERC20Basic
totalSupply
balanceOf
transfer

Summary

Chrema Coin contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions and massively blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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 TAC 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.

A **TAC Security** Company

The Cyberscope team

cyberscope.io