Cyberscope

Audit Report

SEVO-X

March 2025

Network	BSC
Address	0x26f6448253fdf3883f9510c3b119a8bfdac069ff
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
	ANU	Allowance Not Updated	Unresolved
	TPSC	Token Price Stability Concern	Unresolved
	UAC	Unnecessary Allowance Check	Unresolved
	MLR	Misleading Locked Reference	Unresolved
	CCR	Contract Centralization Risk	Unresolved
	FLGO	For Loop Gas Optimization	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	MC	Missing Check	Unresolved
	ODM	Oracle Decimal Mismatch	Unresolved
	POSD	Potential Oracle Stale Data	Unresolved
	TSI	Tokens Sufficiency Insurance	Unresolved
	L13	Divide before Multiply Operation	Unresolved
	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	SEVOX
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0x26f6448253fdf3883f9510c3b119a8bfdac069ff
Address	0x26f6448253fdf3883f9510c3b119a8bfdac069ff
Network	BSC
Symbol	SEVO-X
Decimals	18
Total Supply	1.000.000.000
Badge Eligibility	Yes

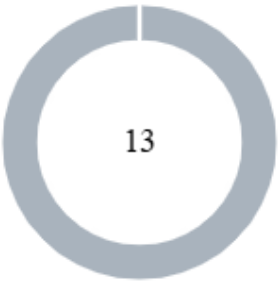
Audit Updates

Initial Audit	05 Mar 2025

Source Files

Filename	SHA256
contracts/SEVOX.sol	4d59669cd7a8c788f1937f7d3e1855fc3ac87daf3ee7c79dc43794cadd632a79

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	13	0	0	0

ANU - Allowance Not Updated

Criticality	Minor / Informative
Location	contracts/SEVOX.sol#L89
Status	Unresolved

Description

The contract uses the `processTokenTransfer` to handle the transferring of tokens but before it checks the `allowance`. Since the intended use is to allow the contract to make a transfer if it is allowed then it should also be updated to reflect the allowance after the transfer. In the context of handling token transfers within a smart contract, it's crucial to ensure that the allowance mechanism is correctly updated after a transfer.

```
function processTokenTransfer(address from, address to, uint
walletTokens, uint lockedTokens) internal  {
    require(allowance(from, address(this)) >= walletTokens,
"Insufficient allowance");
    _transfer(from, to, walletTokens);
    //...
}
```

Recommendation

The `processTokenTransfer` function should be updated to reflect the allowance change after the transfer has been made.

TPSC - Token Price Stability Concern

Criticality	Minor / Informative
Location	contracts/SEVOX.sol#L45
Status	Unresolved

Description

Token's price is fixed at 0.01% of current BNB price (5% of tokens calculated are sold at 0.01% of current BNB price). If the price of BNB drops, the token's price in USD will also decrease, which could potentially make the token seem undervalued. Additionally, if the price of BNB rises, the token may appear overpriced.

```
function calculatePurchaseTokens(uint amount) public view returns(uint
walletTokens, uint lockedTokens) {
    uint bnbPrice = getBNBPrice();
    uint tokens = (amount * bnbPrice) / (PRICE_IN_CENTS * 1e6);
    walletTokens = (tokens * INITIAL_WALLET_PERCENT) / 10000;
    lockedTokens = tokens - walletTokens;
}
```

Recommendation

It is recommended that the team consider the risks of fixing the token price as a percentage of the BNB price. Since BNB is volatile, the token's price will change whenever the BNB price fluctuates. The team could adjust the contract to allow for price changes, ensuring the token's price aligns with the intended value.

UAC - Unnecessary Allowance Check

Criticality	Minor / Informative
Location	contracts/SEVOX.sol#L65,76,92
Status	Unresolved

Description

The `grantTokens` and `claimAirdrop` functions both call the `processTokenTransfer`. Inside that function, there is a `require` statement that checks if the `from` address (`msg.sender` or `creator` respectively) has allowed `this` contract to spend at least `walletTokens` amount. However, the contract does not actually transfer the tokens so this requirement is unnecessary for these two functions.

```
function grantTokens(address to, uint tokens) external {}
function claimAirdrop() external {}

function processTokenTransfer(address from, address to, uint
walletTokens, uint lockedTokens) internal  {
    require(allowance(from, address(this)) >= walletTokens,
"Insufficient allowance");
    //...
}
```

Recommendation

It is recommended to use `require` statements only when they are actually needed to increase gas optimization and readability.

MLR - Misleading Locked Reference

Criticality	Minor / Informative
Location	contracts/SEVOX.sol#L89
Status	Unresolved

Description

There are multiple occasions in the contract that locked tokens are mentioned. The contract even implements functionality that records locked tokens when a user makes a purchase (`purchaseTokens`) or receives them (`grantTokens` , `claimAirdrop`). However, tokens are never actually locked. Instead, they remain on their initial owner.

```
function processTokenTransfer(address from, address to, uint
walletTokens, uint lockedTokens) internal  {
    require(allowance(from, address(this)) >= walletTokens,
"Insufficient allowance");
    _transfer(from, to, walletTokens);
    lockedClaims[to].push(LockedClaim({blockNumber:
block.number, amount: lockedTokens}));
    if (lockedClaims[to].length == 1) {
        lockedClaimUsers.push(to);
    }
}
```

Furthermore, if the intention was to lock the tokens, there should have been some functionality that allows users to unlock them after some criteria are met(e.g.: after `x` years pass). If tokens can be locked with no way of being unlocked it is essentially similar to burning them.

Recommendation

The team should consider updating the contract to accurately reflect its current use.

However, if the intention was actually locking the tokens then the team should implement an actual locking mechanism for the tokens, ensuring that the locked tokens cannot be transferred or used until a certain condition is met.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	contracts/SEVOX.sol#L52,82,104
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 purchaseTokens() public payable {
    //...
    processTokenTransfer(creator, msg.sender, walletTokens,
lockedTokens);
    //...
}
function claimAirdrop() external {
    //...
    processTokenTransfer(creator, msg.sender, walletTokens,
lockedTokens);
    //...
}
function addAirdropUsers(address[] calldata users, uint[]
calldata tokens) external {
    require(msg.sender == creator, "Only creator can add
airdrop users");
    //...
}
```

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.

FLGO - For Loop Gas Optimization

Criticality	Minor / Informative
Location	contracts/SEVOX.sol#L104
Status	Unresolved

Description

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

```
function addAirdropUsers(address[] calldata users, uint[]
calldata tokens) external {
    require(msg.sender == creator, "Only creator can add
airdrop users");
    require(users.length == tokens.length, "Mismatched
arrays");
    for (uint i = 0; i < users.length; i++) {
        airdropUsers[users[i]] += tokens[i];
    }
}
```

Recommendation

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

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/SEVOX.sol#L39
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.

```
creator
```

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	contracts/SEVOX.sol#L104
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 `addAirdropUsers` checks are missing to ensure that the array of `users` does not include `address(0)` and that the array of `tokens` does not include the `uint(0)` .

```
function addAirdropUsers(address[] calldata users, uint[]
calldata tokens) external {
    require(msg.sender == creator, "Only creator can add
airdrop users");
    require(users.length == tokens.length, "Mismatched
arrays");

    for (uint i = 0; i < users.length; i++) {
        airdropUsers[users[i]] += tokens[i];
    }
}
```

Recommendation

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

ODM - Oracle Decimal Mismatch

Criticality	Minor / Informative
Location	contracts/SEVOX.sol#L47
Status	Unresolved

Description

The contract relies on data retrieved from an external Oracle to make critical calculations. However, the contract does not include a verification step to align the decimal precision of the retrieved data with the precision expected by the contract's internal calculations. This mismatch in decimal precision can introduce substantial errors in calculations involving decimal values.

```
function calculatePurchaseTokens(uint amount) public view
returns(uint walletTokens, uint lockedTokens) {
    uint bnbPrice = getBNBPrice();
    uint tokens = (amount * bnbPrice) / (PRICE_IN_CENTS * 1e6);
    walletTokens = (tokens * INITIAL_WALLET_PERCENT) / 10000;
    lockedTokens = tokens - walletTokens;
}
```

Recommendation

The team is advised to retrieve the decimals precision from the Oracle API in order to proceed with the appropriate adjustments to the internal decimals representation.

POSD - Potential Oracle Stale Data

Criticality	Minor / Informative
Location	contracts/SEVOX.sol#L137
Status	Unresolved

Description

The contract relies on retrieving price data from an oracle. However, it lacks proper checks to ensure the data is not stale. The absence of these checks can result in outdated price data being trusted, potentially leading to significant financial inaccuracies.

```
function getBNBPrice() public view returns (uint) {
    (, int256 price, , , ) =
IChainlinkPriceFeed(PRICE_FEED_ADDRESS).latestRoundData();
    require(price > 0, "Invalid BNB price");
    return uint(price);
}
```

Recommendation

To mitigate the risk of using stale data, it is recommended to implement checks on the round and period values returned by the oracle's data retrieval function. The value indicating the most recent round or version of the data should confirm that the data is current. Additionally, the time at which the data was last updated should be checked against the current interval to ensure the data is fresh. For example, consider defining a threshold value, where if the difference between the current period and the data's last update period exceeds this threshold, the data should be considered stale and discarded, raising an appropriate error.

For contracts deployed on Layer-2 solutions, an additional check should be added to verify the sequencer's uptime. This involves integrating a boolean check to confirm the sequencer is operational before utilizing oracle data. This ensures that during sequencer downtimes, any transactions relying on oracle data are reverted, preventing the use of outdated and potentially harmful data.

By incorporating these checks, the smart contract can ensure the reliability and accuracy of the price data it uses, safeguarding against potential financial discrepancies and enhancing overall security.

TSI - Tokens Sufficiency Insurance

Criticality	Minor / Informative
Location	contracts/SEVOX.sol#L52,76,89
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 purchaseTokens() public payable {
    //...
    processTokenTransfer(creator, msg.sender, walletTokens,
lockedTokens);
    //...
}
function claimAirdrop() external {
    //...
    processTokenTransfer(creator, msg.sender, walletTokens,
lockedTokens);
    //...
}
function processTokenTransfer(address from, address to, uint
walletTokens, uint lockedTokens) internal  {
    require(allowance(from, address(this)) >= walletTokens,
"Insufficient allowance");
    _transfer(from, to, walletTokens);
    //...
}
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/SEVOX.sol#L47,48
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint tokens = (amount * bnbPrice) / (PRICE_IN_CENTS * 1e6)
walletTokens = (tokens * INITIAL_WALLET_PERCENT) / 10000
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/SEVOX.sol#L2
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 ^0.8.17;
```

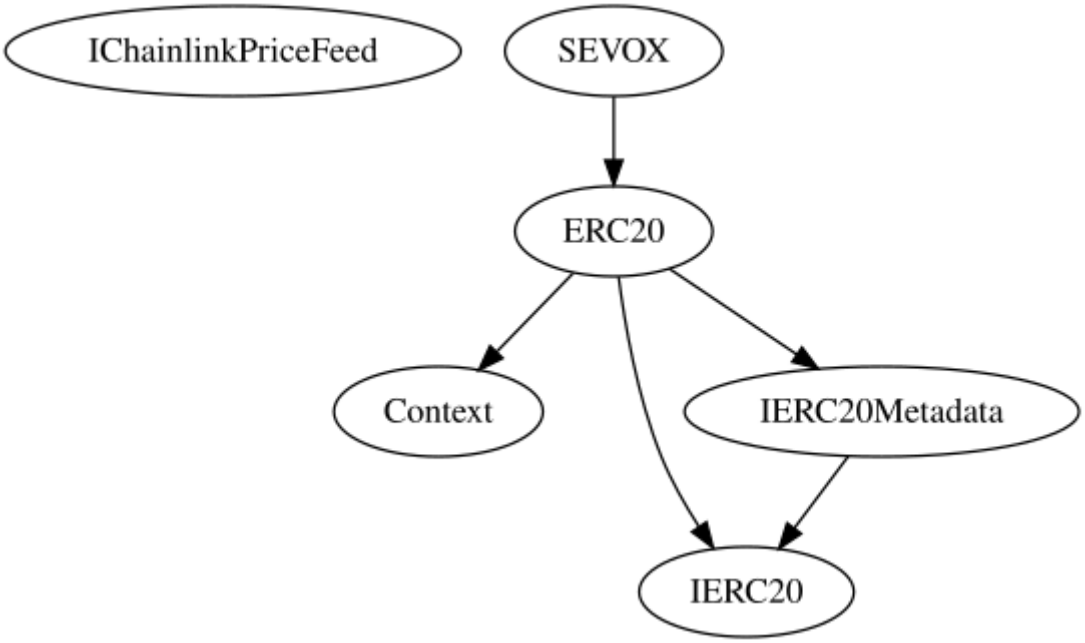
Recommendation

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.

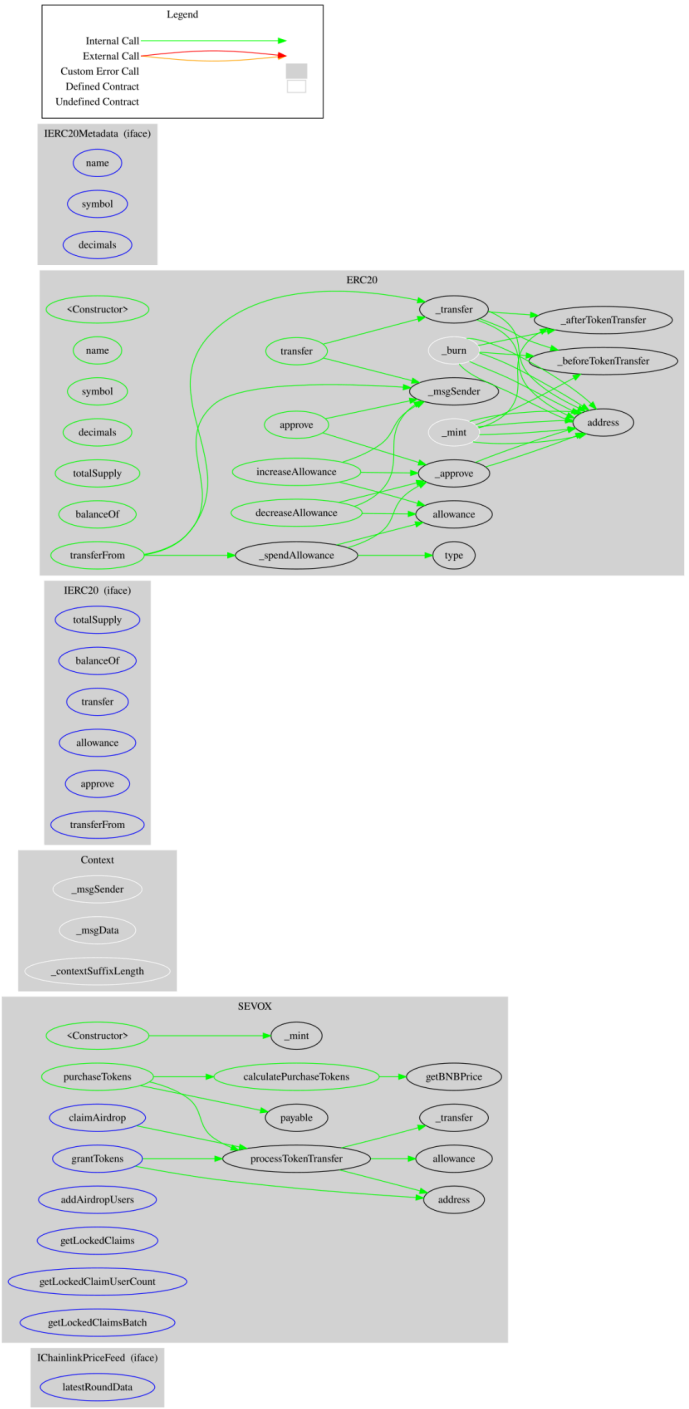
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SEVOX	Implementation	ERC20		
		Public	✓	ERC20
	calculatePurchaseTokens	Public		-
	purchaseTokens	Public	Payable	-
	grantTokens	External	✓	-
	claimAirdrop	External	✓	-
	processTokenTransfer	Internal	✓	
	addAirdropUsers	External	✓	-
	getLockedClaims	External		-
	getLockedClaimUserCount	External		-
	getLockedClaimsBatch	External		-
	getBNBPrice	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Custom Error Call
Defined Contract
Undefined Contract
IERC20Metadata (iface)
name
symbol
decimals
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
approve
increaseAllowance
decreaseAllowance
_spendAllowance
_transfer
_burn
_msgSender
_mint
_approve
allowance
type
_afterTokenTransfer
_beforeTokenTransfer
address
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
_contextSuffixLength
SEVOX
<Constructor>
purchaseTokens
claimAirdrop
grantTokens
addAirdropUsers
getLockedClaims
getLockedClaimUserCount
getLockedClaimsBatch
_mint
calculatePurchaseTokens
payable
processTokenTransfer
getBNBPrice
_transfer
allowance
address
IChainlinkPriceFeed (iface)
latestRoundData

Summary

SEVO-X contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. The Smart Contract analysis reported no compiler error but it did encounter a critical issue. The contract creator can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

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