Cyberscope

Audit Report

EverEth

October 2023

SHA256 bd8b2f0d2dfffe2e2bd2eabba3e615d640d4524467a101bf8ba5cc37a4192760

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
●	US	Untrusted Source	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0x3b7000e55204f6695f3ec86408b71ec6667a2e56

Audit Updates

Initial Audit	24 Sep 2023 https://github.com/cyberscope-io/audits/blob/main/evereth-2/v1/audit.pdf
Corrected Phase 2	04 Oct 2023

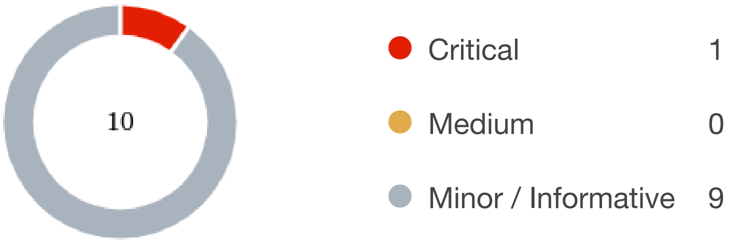
Source Files

Filename	SHA256
contracts/EETH.sol	bd8b2f0d2dfffe2e2bd2eabba3e615d640d4524467a101bf8ba5cc37a4192760

Overview

The EverETH contract implements an ERC-20 token with dividend distribution functionality. The dividends, which consist of native tokens, are distributed to token holders when ETH is sent to the contract, and token holders can claim their dividends by calling the `claim` method. Additionally, the owner of the contract has control over various parameters and can recover funds sent to the contract in error. It is important to note that for the dividend mechanism to function correctly, funds must be deposited into the contract.

Findings Breakdown



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

US - Untrusted Source

Criticality	Critical
Location	contracts/EETH.sol#L1535,1610
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.

As the `dividendTracker` is mutable, it could produce complications in the transfer transaction.

```
 function updateDividendTracker(address newAddress) public onlyOwner {
       EverETHDividendTracker newDividendTracker =
EverETHDividendTracker(payable(newAddress));
        newDividendTracker.excludeFromDividends(
            address(newDividendTracker)
        );
        newDividendTracker.excludeFromDividends(address(this));
        newDividendTracker.excludeFromDividends(owner());
        newDividendTracker.excludeFromDividends(deadWallet);
        dividendTracker = newDividendTracker;
    }
...
function _transfer(
    address from,
    address to,
    uint256 amount
) internal override {
    require(from != address(0), "ERC20: transfer from the zero
address");

    if (amount == 0) {
        super._transfer(from, to, 0);
        return;
    }

    super._transfer(from, to, amount);
        dividendTracker.setBalance(payable(from), balanceOf(from));
        dividendTracker.setBalance(payable(to), balanceOf(to));
}
```

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. It is recommended to use a try-catch statement in external calls.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/EETH.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	contracts/EETH.sol#L1132,1142,2265,2269,2276,2282
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
AddressSlot storage r
BooleanSlot storage r
Map storage map
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/EETH.sol#L363,379,427,431,719,954,958,1019,1245,1262,1266,1348,1385,1389,1404,1488,2074,2171,2178,2190,2204,2540,2565
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.

```
function _msgSender() internal view virtual returns (address) {
    return msg.sender;
}

function _msgData() internal view virtual returns (bytes calldata) {
    return msg.data;
}

function __ERC20_init_unchained(string memory name_, string memory
symbol_) internal onlyInitializing {
    _name = name_;
    _symbol = symbol_;
}

...
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	contracts/EETH.sol#L1232,1238
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
bytes32 internal constant _ADMIN_SLOT =
0xb53127684a568b3173ae13b9f8a6016e243e63b6e8ee1178d6a717850b5d6103;

bytes32 internal constant _BEACON_SLOT =
0xa3f0ad74e5423aebfd80d3ef4346578335a9a72aeaee59ff6cb3582b35133d50;
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/EETH.sol#L363,1393,1954
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 __Context_init() internal onlyInitializing {
    }

function _nonReentrantAfter() private {
        // By storing the original value once again, a refund is
triggered (see
        // https://eips.ethereum.org/EIPS/eip-2200)
        _status = _NOT_ENTERED;
    }
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/EETH.sol#L1218
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
try IERC1822ProxiableUpgradeable(newImplementation).proxiableUUID()
returns (bytes32 slot) {
    require(slot == _IMPLEMENTATION_SLOT, "ERC1967Upgrade: unsupported
proxiableUUID");
} catch {
    revert("ERC1967Upgrade: new implementation is not UUPS");
}
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/EETH.sol#L2094
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
constructor(string memory _name, string memory _symbol)
         ERC20(_name, _symbol)
     {}
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/EETH.sol#L209,1134,1144
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 {
    let returndata_size := mload(returndata)
    revert(add(32, returndata), returndata_size)
    }
...
assembly {
    r.slot := slot
    }
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/EETH.sol#L1526
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(tokenAddress).transfer(owner(), tokenAmount);
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20Upgradeable	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20MetadataUpgradeable	Interface	IERC20Upgradeable		
	name	External		-
	symbol	External		-
	decimals	External		-
AddressUpgradeable	Library			
	isContract	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	

	verifyCallResultFromTarget	Internal		
	_revert	Private		
Initializable	Implementation			
ContextUpgradeable	Implementation	Initializable		
	__Context_init	Internal	✓	onlyInitializing
	_msgSender	Internal		
	_msgData	Internal		
ERC20Upgradeable	Implementation	Initializable, ContextUpgradeable, IERC20Upgradeable, IERC20MetadataUpgradeable		
	__ERC20_init	Internal	✓	onlyInitializing
	__ERC20_init_unchained	Internal	✓	onlyInitializing
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-

	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
SafeMathUpgradeable	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		

	mod	Internal		
OwnableUpgradeable	Implementation	Initializable, ContextUpgradeable		
	__Ownable_init	Internal	✓	onlyInitializing
	__Ownable_init_unchained	Internal	✓	onlyInitializing
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC1822ProxiableUpgradeable	Interface			
	proxiableUUID	External		-
IBeaconUpgradeable	Interface			
	implementation	External		-
IERC1967Upgradeable	Interface			
StorageSlotUpgradeable	Library			
	getAddressSlot	Internal		

	getBooleanSlot	Internal		
ERC1967UpgradeUpgradeable	Implementation	Initializable, IERC1967Upgradeable		
	_getImplementation	Internal		
	_setImplementation	Private	✓	
	_upgradeTo	Internal	✓	
	_upgradeToAndCall	Internal	✓	
	_upgradeToAndCallUUPS	Internal	✓	
UUPSUpgradeable	Implementation	Initializable, IERC1822ProxiableUpgradeable, ERC1967UpgradeUpgradeable		
	__UUPSUpgradeable_init	Internal	✓	onlyInitializing
	proxiableUUID	External		notDelegated
	upgradeTo	Public	✓	onlyProxy
	upgradeToAndCall	Public	Payable	onlyProxy
	_authorizeUpgrade	Internal	✓	
ReentrancyGuardUpgradeable	Implementation	Initializable		
	__ReentrancyGuard_init	Internal	✓	onlyInitializing
	__ReentrancyGuard_init_unchained	Internal	✓	onlyInitializing
	_nonReentrantAfter	Private	✓	

IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
EverETH	Implementation	Initializable, ERC20Upgradeable, OwnableUpgradeable, UUPSUpgradeable, ReentrancyGuardUpgradeable		
	initialize	External	✓	initializer
		External	Payable	-
	_authorizeUpgrade	Internal	✓	onlyOwner
	recoverETH	External	✓	onlyOwner
	recoverERC20	External	✓	onlyOwner
	excludeFromDividends	Public	✓	onlyOwner
	updateDividendTracker	Public	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getTotalDividendsDistributed	External		-
	withdrawableDividendOf	Public		-

	dividendTokenBalanceOf	Public		-
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	claim	External	✓	-
	getNumberOfDividendTokenHolders	External		-
	_transfer	Internal	✓	
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPayingTokenInterface	Interface			
	dividendOf	External		-
	distributeDividends	External	Payable	-
	withdrawDividend	External	✓	-
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	toUint256Safe	Internal		

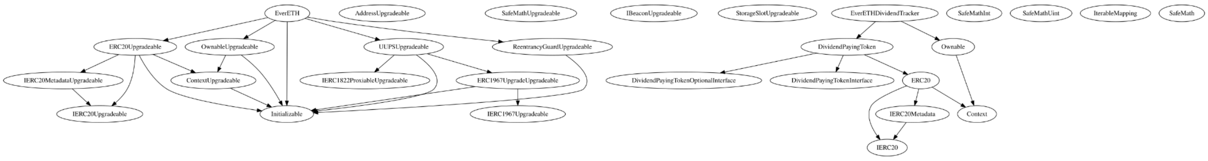
SafeMathUint	Library			
	toInt256Safe	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-

	approve	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
DividendPaying Token	Implementation	ERC20, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
		Public	✓	ERC20
		External	Payable	-
	distributeDividends	Public	Payable	-
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_mint	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	

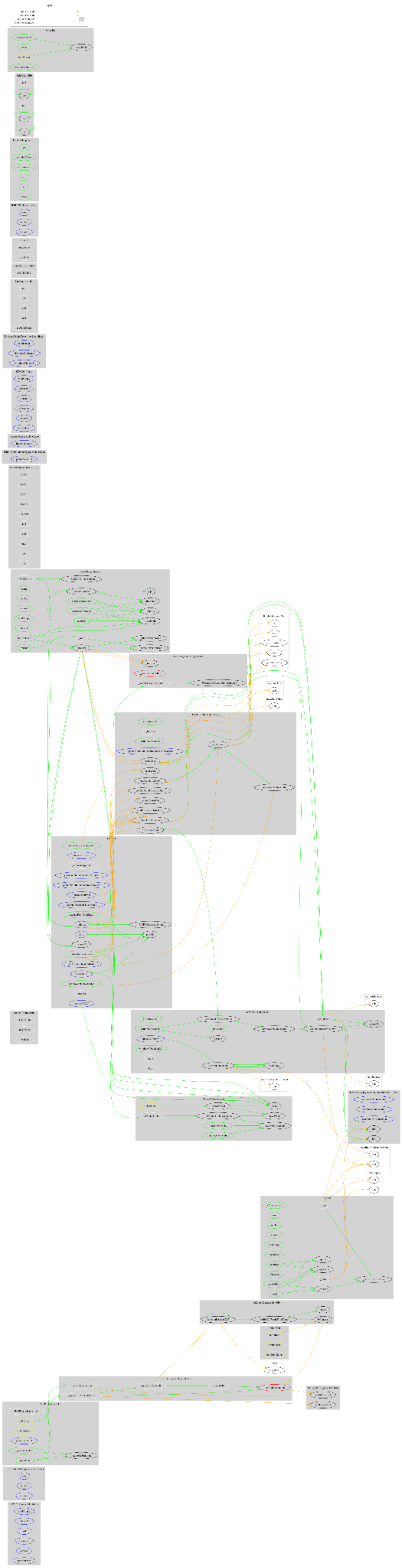
IterableMapping	Library			
	get	Public		-
	getIndexOfKey	Public		-
	getKeyAtIndex	Public		-
	size	Public		-
	set	Public	✓	-
	remove	Public	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	transferOwnership	Public	✓	onlyOwner

EverETHDividendTracker	Implementation	DividendPayingToken, Ownable		
		Public	✓	DividendPayingToken
	_transfer	Internal		
	withdrawDividend	Public		-
	updateMinimumTokenBalanceForDividends	External	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getNumberOfTokenHolders	External		-
	getAccount	Public		-
	getAccountAtIndex	Public		-
	setBalance	External	✓	onlyOwner
	processAccount	Public	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

EverEth contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. EverEth is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors or critical issues. The Contract Owner 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

https://www.cyberscope.io