Cyberscope

Audit Report

Rocket Protocol

Sep 2024

Network	ETH
Address	0x86cf3A865dca1eddE5E1527F9ACA735aaEf85673
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

Contract Name	FairLaunchLimitBlockTokenV3
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	20000 runs
Explorer	https://etherscan.io/address/0x86cf3a865dca1edde5e1527f9aca735aaef85673
Address	0x86cf3a865dca1edde5e1527f9aca735aaef85673
Network	ETH
Symbol	RP
Decimals	18
Total Supply	20,000,000,000

Audit Updates

Initial Audit	12 Jun 2024 https://github.com/cyberscope-io/audits/blob/main/phei/v1/audit.pdf
Corrected Phase 2	27 Aug 2024 https://github.com/cyberscope-io/audits/blob/main/phei/v2/audit.pdf
Corrected Phase 3	06 Sep 2024 https://github.com/cyberscope-io/audits/blob/main/phei/v3/audit.pdf
Corrected Phase 4	17 Sep 2024

Source Files

Filename	SHA256
NoDelegateCall.sol	df71f011abaff271622bb1695fcf0d93cb34d91bfff304b558cd6494ae187190
Meme.sol	540e7b643f661b30afa5552b0cbf8196f07358c49468aa34542155e7e6cad74f
IMeme.sol	fc794154cb742af3cb7397005f188f6e978d66d7e321e725681514480d9574ee
IFairLaunch.sol	8a4f83221d50264b74526f30249a396c3190e92bb9e8599d8cddc3b9e94c4fc9
FairLaunchLimitBlockV3.sol	e617f65f80067e43ff4b9fe0d0008470de4ffb892ebb07360c7160eaea43dcee

Overview

The `FairLaunchLimitBlockV3` contract is designed to facilitate a fair token launch process, incorporating various functionalities for refunding, funding, minting, and starting token trading. It integrates mechanisms to handle user funds, ensure fairness through block-based conditions, and leverage Uniswap for liquidity provision.

The primary purpose of the `FairLaunchLimitBlockV3` contract is to manage the launch of a the `Meme` token with specific constraints and commands. It allows users to fund the contract, request refunds, mint tokens, and initiate trading, all governed by predefined block numbers and funding caps to ensure a fair and orderly process.

Contract's Current State:

The current configuration of the contract sets the `untilBlockNumber` to block 20846863. This configuration means that once block number 20846863 is reached, the `start` function can be invoked.

Before Start Functionality

Funding and Refunding

If the `canStart` condition is `false`, indicating that the designated block number has not yet been reached, users can interact with the contract in several ways. They can send native tokens (ETH) to the contract in specific amounts defined as commands. If users send the `REFUND_COMMAND` value (0.0002), the contract will process a `refund` of their previously contributed funds. Alternatively, if users send any other value, the contract will treat it as a funding contribution, adding the sent amount to their balance and the contract's total fund balance.

Starting Trading and Refunding

If the `canStart` condition is `true`, meaning the designated block number has been reached but trading has not yet started, users can send the `START_COMMAND` value (0.0005) to initiate the token trading process. This action will trigger the contract to add liquidity to Uniswap and set the started state to true. Users can also still request refunds, before the `START_COMMAND` happens, by sending the `REFUND_COMMAND` value, which will process their refund as described earlier.

After Start Functionality

Minting Tokens

Once the contract has started (i.e., trading is enabled), users can send the `MINT_COMMAND` value (0.0001) to mint tokens. The contract verifies that the sender has not already minted tokens and calculates the amount of tokens they are eligible to receive based on their contribution relative to the total funds. The tokens are then transferred to the user, and the origianl `MINT_COMMAND` amount of ETH is returned to the user.

Claiming Extra ETH

After the contract has started, if the total ETH collected exceeds a predefined `softTopCap`, users can claim their share of the excess ETH by sending the `CLAIM_COMMAND` value (0.0002). The contract calculates each user's claimable amount based on their contribution proportion and transfers the corresponding ETH to them. This ensures that any excess funds are fairly distributed among the participants.

Command Implementation

The contract uses specific ETH values as commands to trigger different functionalities:

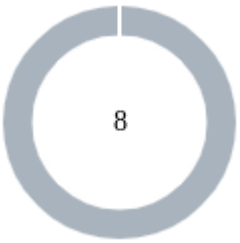
- REFUND_COMMAND (0.0002): Triggers the refund process for users who wish to withdraw their contributions before the start.
- CLAIM_COMMAND (0.0002): Allows users to claim extra funds after the start if the total collected exceeds the soft top cap.
- START_COMMAND (0.0005): Initiates the start of trading and adds liquidity to Uniswap.
- MINT_COMMAND (0.0001): Enables users to mint tokens once trading has started.

Liquidity Pool Initialization

The `_start` function in the contract initializes a Uniswap V3 liquidity pool by minting a position once the presale funds are collected, but the pool itself is created and initialized during the contract's constructor phase using theoretical amounts (`amount0` and `amount1`) to set an initial price. This dual-step approach first locks in the initial price during construction and later mints the actual liquidity position when `_start` is called,

based on the actual ETH and tokens gathered. This prevents premature or malicious pool creation by unauthorized users.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	8	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	DDP	Decimal Division Precision	Unresolved
●	ILC	Ineffective Logic Checks	Unresolved
●	MEM	Missing Error Messages	Unresolved
●	RRC	Redundant Require Checks	Unresolved
●	UBL	Unused BNB Lock	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	FairLaunchLimitBlockV3.sol#L456
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

```
if (fee > 0) {
    (bool success1, ) = refundFeeTo.call{value: totalEthers / 1000}("");
    ...

    (bool success2, ) = projectOwner.call{
        value: (totalEthers * 2) / 1000
    }("");
    ...
}
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

ILC - Ineffective Logic Checks

Criticality	Minor / Informative
Location	FairLaunchLimitBlockV3.sol#L373,349
Status	Unresolved

Description

The contract is implementing checks that do not provide any meaningful logic validation since, based on the contract's setup and variable assignments, they will always evaluate to true. These checks do not contribute to enhancing the security or functionality of the contract, leading to unnecessary complexity and potential inefficiencies.

```
require(_mintAmount > 0, "FairMint: mint amount is zero");
assert(_mintAmount <= totalDispatch / 2);
...
uint256 fee = (amount * refundFeeRate) / 10000;
assert(fee < amount);
```

Recommendation

It is recommended to consider the removal of these redundant checks to simplify the contract. This will improve the contract's readability and maintainability without compromising its security or intended functionality.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	NoDelegateCall.sol#L19
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(address(this) == original)
```

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.

RRC - Redundant Require Checks

Criticality	Minor / Informative
Location	FairLaunchLimitBlockV3.sol#L297,365,366
Status	Unresolved

Description

The contract is performing additional checks that have already been validated earlier in the code. This redundancy can lead to inefficiencies and unnecessary complexity within the contract, potentially increasing gas costs and making the contract harder to maintain.

```
require(started, "FairMint: withdraw extra eth must after start");
...
require(started, "FairMint: not started");
...
require(msg.sender == tx.origin, "FairMint: can not mint to contract.");
```

Recommendation

It is recommended to consider the removal of redundant checks to streamline the code. This will enhance the efficiency of the contract by reducing unnecessary validations, lowering gas costs, and improving code clarity and maintainability.

UBL - Unused BNB Lock

Criticality	Minor / Informative
Location	FairLaunchLimitBlockV3.sol#L432
Status	Unresolved

Description

The contract is invoking the `refundETH()` function after minting liquidity, which successfully returns any excess BNB that was not used in the liquidity provision process back to the contract's balance. However, the contract lacks any mechanisms to either utilize this refunded BNB within its existing functionalities or to withdraw it. As a result, the refunded BNB remains locked within the contract indefinitely, potentially leading to inefficiencies or unintended behavior in terms of resource allocation.

```
(
    uint256 _tokenId,
    uint128 _liquidity,
    uint256 _amount0,
    uint256 _amount1
) = _positionManager.mint{value: totalAdd}(params);
_positionManager.refundETH();
```

Recommendation

It is recommended to consider the intended functionality regarding the refunded BNB. The contract should implement a mechanism to properly handle the refunded BNB, either by integrating it into the contract's operational logic (e.g., adding it to available funds for other uses) or providing a secure method to withdraw or redistribute the funds. This will ensure that the contract’s resources are managed efficiently and prevent unnecessary locking of BNB.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	FairLaunchLimitBlockV3.sol#L40,279
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 WETH9() external pure returns (address);
address _addr
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	FairLaunchLimitBlockV3.sol#L191,192
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

```
locker = _locker
projectOwner = _projectOwner
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	NoDelegateCall.sol#L2 Meme.sol#L2 IMeme.sol#L2 IFairLaunch.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 ^0.8.19;
```

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.

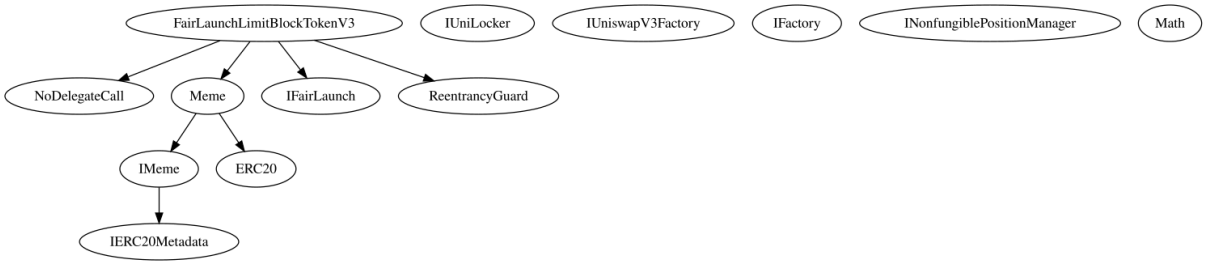
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
NoDelegateCall	Implementation			
		Public	✓	-
	checkNotDelegateCall	Private		
Meme	Implementation	IMeme, ERC20		
		Public	✓	ERC20
	_update	Internal	✓	
IMeme	Interface	IERC20Metadata		
	meta	External		-
IFairLaunch	Interface			
IUniLocker	Interface			
	lock	External	✓	-
IUniswapV3Factory	Interface			
	getPool	External		-

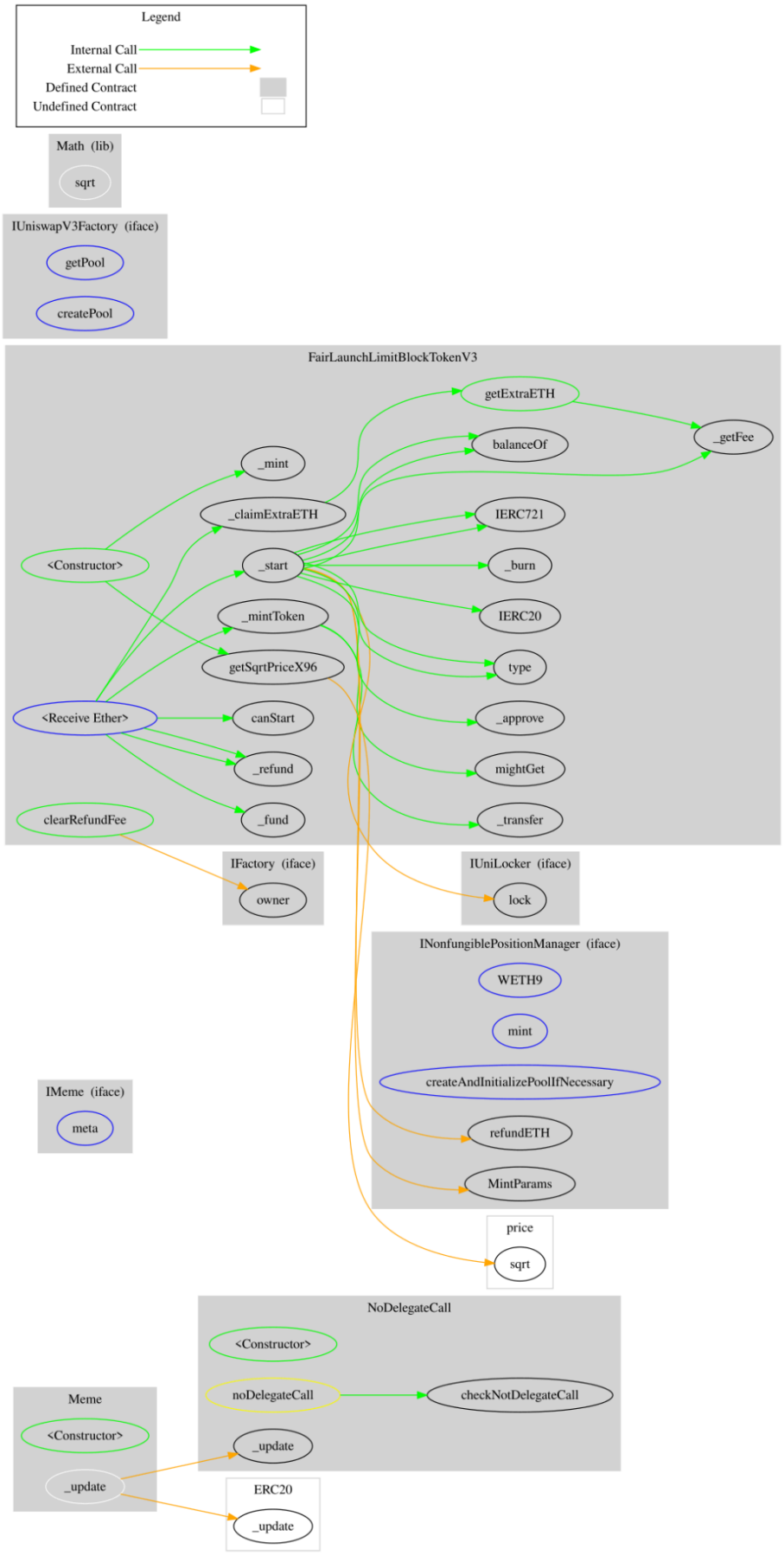
	createPool	External	✓	-
IFactory	Interface			
	owner	External		-
INonfungiblePositionManager	Interface			
	WETH9	External		-
	mint	External	Payable	-
	createAndInitializePoolIfNecessary	External	Payable	-
	refundETH	External	Payable	-
FairLaunchLimitBlockTokenV3	Implementation	IFairLaunch, Meme, ReentrancyGuard, NoDelegateCall		
		Public	✓	Meme
	clearRefundFee	Public	✓	-
		External	Payable	noDelegateCall
	canStart	Public		-
	getExtraETH	Public		-
	_getFee	Private		
	_claimExtraETH	Private	✓	nonReentrant
	mightGet	Public		-
	_fund	Private	✓	nonReentrant

	_refund	Private	✓	nonReentrant
	_mintToken	Private	✓	nonReentrant
	_start	Private	✓	nonReentrant
	_update	Internal	✓	
	getSqrtPriceX96	Internal		

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Math (lib)
sqrt
IUniswapV3Factory (iface)
getPool
createPool
FairLaunchLimitBlockTokenV3
getExtraETH
_getFee
balanceOf
_mint
_claimExtraETH
IERC721
<Constructor>
_start
_burn
_mintToken
IERC20
getSqrtPriceX96
type
<Receive Ether>
canStart
_approve
_refund
mightGet
clearRefundFee
_fund
_transfer
IFactory (iface)
owner
IUniLocker (iface)
lock
INonfungiblePositionManager (iface)
WETH9
mint
createAndInitializePoolIfNecessary
IMeme (iface)
meta
refundETH
MintParams
price
sqrt
NoDelegateCall
<Constructor>
noDelegateCall
checkNotDelegateCall
Meme
<Constructor>
_update
_update
ERC20
_update

Summary

The FairLaunchLimitBlockV3 contract is a comprehensive solution for launching the MEME token contract with built-in functionalities for handling funds, initiating trading, and ensuring fairness through block-based conditions and funding caps, including a refund functionality. This audit investigates security issues, business logic concerns, and potential improvements.

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