Cyberscope

Audit Report

PHEI

June 2024

Network	ETH
Address	0x66A6C46147F974ab8FdB28347Fd1FCd139714d8a
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 3
Audit Updates 3
Source Files 3
Overview 5
Current Contract's State 5
General Purpose 6
Before Start Functionality 6
Funding and Refunding 6
Starting Trading and Refunding 6
After Start Functionality 7
Minting Tokens 7
Claiming Extra ETH 7
Command Implementation 7
Uniswap Integration 7
Findings Breakdown 8
Diagnostics 9
IDI - Immutable Declaration Improvement 10
Description 10
Recommendation 10
ILC - Ineffective Logic Checks 11
Description 11
Recommendation 11
ILM - Inefficient Liquidity Management 12
Description 12
Recommendation 12
ISFC - Insufficient Start Funds Check 14
Description 14
Recommendation 14
RRC - Redundant Require Checks 15
Description 15
Recommendation 15
URM - Unclarified Revert Message 16
Description 16
Recommendation 16
L04 - Conformance to Solidity Naming Conventions 17
Description 17
Recommendation 17
L11 - Unnecessary Boolean equality 18

Description 18
Recommendation 18
L13 - Divide before Multiply Operation 19
Description 19
Recommendation 19
L19 - Stable Compiler Version 20
Description 20
Recommendation 20
Functions Analysis **21**
Inheritance Graph **22**
Flow Graph **23**
Summary **24**
Disclaimer **25**
About Cyberscope **26**

Review

Contract Name	FairLaunchLimitBlockToken
Compiler Version	v0.8.24+commit.e11b9ed9
Optimization	20000 runs
Explorer	https://etherscan.io/address/0x66a6c46147f974ab8fdb28347fd1fcd139714d8a
Address	0x66a6c46147f974ab8fdb28347fd1fcd139714d8a
Network	ETH
Symbol	PHEI
Decimals	18
Total Supply	20,000,000,000

Audit Updates

Initial Audit	13 Jun 2024

Source Files

Filename	SHA256
Meme.sol	b205807e7475f7e0420a93920546b2bfcd2c6c48afe101139055e1252e235098
IMeme.sol	2c72dd6b83eb12520c9f92f18e0fb063602f2cd7aca0fd1872a312034a880a20
IFairLaunch.sol	8a4f83221d50264b74526f30249a396c3190e92bb9e8599d8cddc3b9e94c4fc9

FairLaunchLimitBlock.sol	18fc47c9c11c049ab53707e4505f0262862a17a32a9e807fc8d262a3dd1c3034

Overview

⚠ Current Contract's State ⚠

The `FairLaunchLimitBlockToken` contract is deployed at the address `0x66A6C46147F974ab8FdB28347Fd1FCd139714d8a` on the Ethereum blockchain. However, since the `untilBlockNumber` is set to `20043524`, the `canStart` function evaluates to `true`, meaning the contract is ready to be started. Despite this, the contract currently has no funds, which prevents its core functionalities from being executed. This lack of funds means that refunds cannot be processed, and token mints cannot occur because no addresses have participated in the funding process. Consequently, in its current state, the contract is unable to perform its intended operations, meaning that the contract cannot be used for the purpose it is supposed to.

General Purpose

The contract `FairLaunchLimitBlockToken` 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 `FairLaunchLimitBlockToken` contract is to manage the launch of 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.

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 ETH), 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 ETH) 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 ETH) 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 ETH). 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:

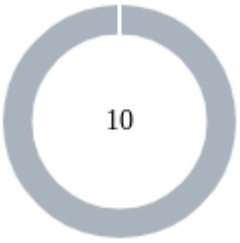
- REFUND_COMMAND (0.0002 ETH): Triggers the refund process for users who wish to withdraw their contributions before the start.
- CLAIM_COMMAND (0.0002 ETH): Allows users to claim extra ETH after the start if the total collected exceeds the soft top cap.
- START_COMMAND (0.0005 ETH): Initiates the start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

Uniswap Integration

The contract integrates with Uniswap to provide liquidity once trading starts. Upon receiving the `START_COMMAND`, it adds the correct amount of collected ETH and tokens to Uniswap, creating a liquidity pool. This ensures that there is immediate liquidity for the

token, facilitating trading on the platform. The contract also handles the burn of liquidity provider (LP) tokens, sending them to a zero address to prevent any further manipulation.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	10

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	10	0	0	0

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	IDI	Immutable Declaration Improvement	Unresolved
●	ILC	Ineffective Logic Checks	Unresolved
●	ILM	Inefficient Liquidity Management	Unresolved
●	ISFC	Insufficient Start Funds Check	Unresolved
●	RRC	Redundant Require Checks	Unresolved
●	URM	Unclarified Revert Message	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L19	Stable Compiler Version	Unresolved

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	FairLaunchLimitBlock.sol#L99,103,104,108,109,111
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.

```
totalDispatch
uniswapRouter
uniswapFactory
untilBlockNumber
softTopCap
refundFeeRate
```

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.

ILC - Ineffective Logic Checks

Criticality	Minor / Informative
Location	FairLaunchLimitBlock.sol#L174,248,255,273
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(softTopCap > 0, "FairMint: soft top cap must be set");
...
require(_mintAmount > 0, "FairMint: mint amount is zero");
assert(_mintAmount <= totalDispatch / 2);
...
uint256 fee = (amount * refundFeeRate) / 10000;
assert(fee < amount);
...
assert(_pair != address(0));
```

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.

ILM - Inefficient Liquidity Management

Criticality	Minor / Informative
Location	FairLaunchLimitBlock.sol#L285
Status	Unresolved

Description

The contract is currently utilizing a separate function, `_dropLP`, to transfer LP tokens to the zero address. This introduces unnecessary complexity and potential inefficiencies. Instead, the contract could directly set the zero address as the recipient of the LP tokens within the `addLiquidity` functionality, simplifying the process and reducing the risk of errors.

```
        (uint256 tokenAmount, uint256 ethAmount, uint256 liquidity) =
router
        // .addLiquidityETH{value: address(this).balance}(
            .addLiquidityETH{value: totalAdd}(
            address(this), // token
            totalDispatch / 2, // token desired
            totalDispatch / 2, // token min
            totalAdd, // eth min
            address(this), // lp to
            block.timestamp + 1 days // deadline
        );
        _dropLP(_pair);
        emit LaunchEvent(address(this), tokenAmount, ethAmount,
liquidity);

        (bool success, ) = msg.sender.call{value: START_COMMAND}("");
        require(success, "FairMint: mint failed");
    }

    function _dropLP(address lp) private {
        IERC20 lpToken = IERC20(lp);
        lpToken.safeTransfer(address(0),
lpToken.balanceOf(address(this)));
    }
```

Recommendation

It is recommended to streamline the liquidity addition process by directly setting the zero address as the recipient of the LP tokens in the addLiquidity functionality, eliminating the need for the separate `_dropLP` function.

ISFC - Insufficient Start Funds Check

Criticality	Minor / Informative
Location	FairLaunchLimitBlock.sol#L132
Status	Unresolved

Description

The contract is designed to allow any user to invoke the `_start` function by sending the `START_COMMAND` value to the contract. However, in its current state, if the contract has zero funds, this could lead to issues where the start process is initiated without any liquidity to add to Uniswap. This can result in an incomplete or faulty launch process, potentially affecting the token's market operations and user trust.

```
if (canStart()) {
    ...
    } else if (msg.value == START_COMMAND) {
        // start trading, add liquidity to uniswap
        _start();
```

Recommendation

It is recommended that the contract include a check to ensure it has native tokens (ETH) greater than zero before allowing the `_start` function to be invoked. This will prevent the start process from being initiated when there are no funds available, ensuring a smooth and functional launch of the token's liquidity provision.

RRC - Redundant Require Checks

Criticality	Minor / Informative
Location	FairLaunchLimitBlock.sol#L173,214,239
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(msg.value == CLAIM_COMMAND, "FairMint: value not
match");
...
require(msg.value == REFUND_COMMAND, "FairMint: value not
match");
...
require(started, "FairMint: not started");
require(msg.value == MINT_COMMAND, "FairMint: value not
match");
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.

URM - Unclarified Revert Message

Criticality	Minor / Informative
Location	FairLaunchLimitBlock.sol#L120
Status	Unresolved

Description

The contract is designed to handle multiple commands after it has started, including minting tokens and claiming extra ETH. However, the revert message `"FairMint: invalid command - mint only"` is misleading as it incorrectly implies that minting is the only allowed option, while in fact, the contract also supports the claim command. This can cause confusion for users and developers interacting with the contract, potentially leading to misunderstandings about its functionality.

```
if (started) {
    // after started
    if (msg.value == MINT_COMMAND) {
        // mint token
        _mintToken();
    } else if (msg.value == CLAIM_COMMAND) {
        _claimExtraETH();
    } else {
        revert("FairMint: invalid command - mint only");
    }
```

Recommendation

It is recommended to update the revert message to accurately reflect the allowed commands. A more accurate message would help clarify that both minting and claiming are valid actions, thereby reducing potential confusion and improving the overall user experience.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	FairLaunchLimitBlock.sol#L12,160
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 WETH() 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/v0.8.17/style-guide.html#naming-convention.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	FairLaunchLimitBlock.sol#L182
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
require(claimed[msg.sender] == false, "FairMint: already
claimed")
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	FairLaunchLimitBlock.sol#L199
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.

```
uint256 _mintAmount = ((totalDispatch / 2) *
fundBalanceOf[account]) / totalEthers
```

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	Meme.sol#L2 IMeme.sol#L2 FairLaunchLimitBlock.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;
```

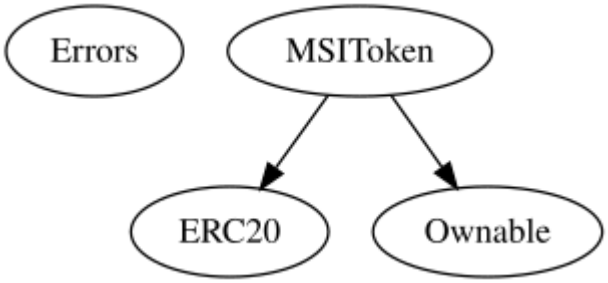
Recommendation

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.

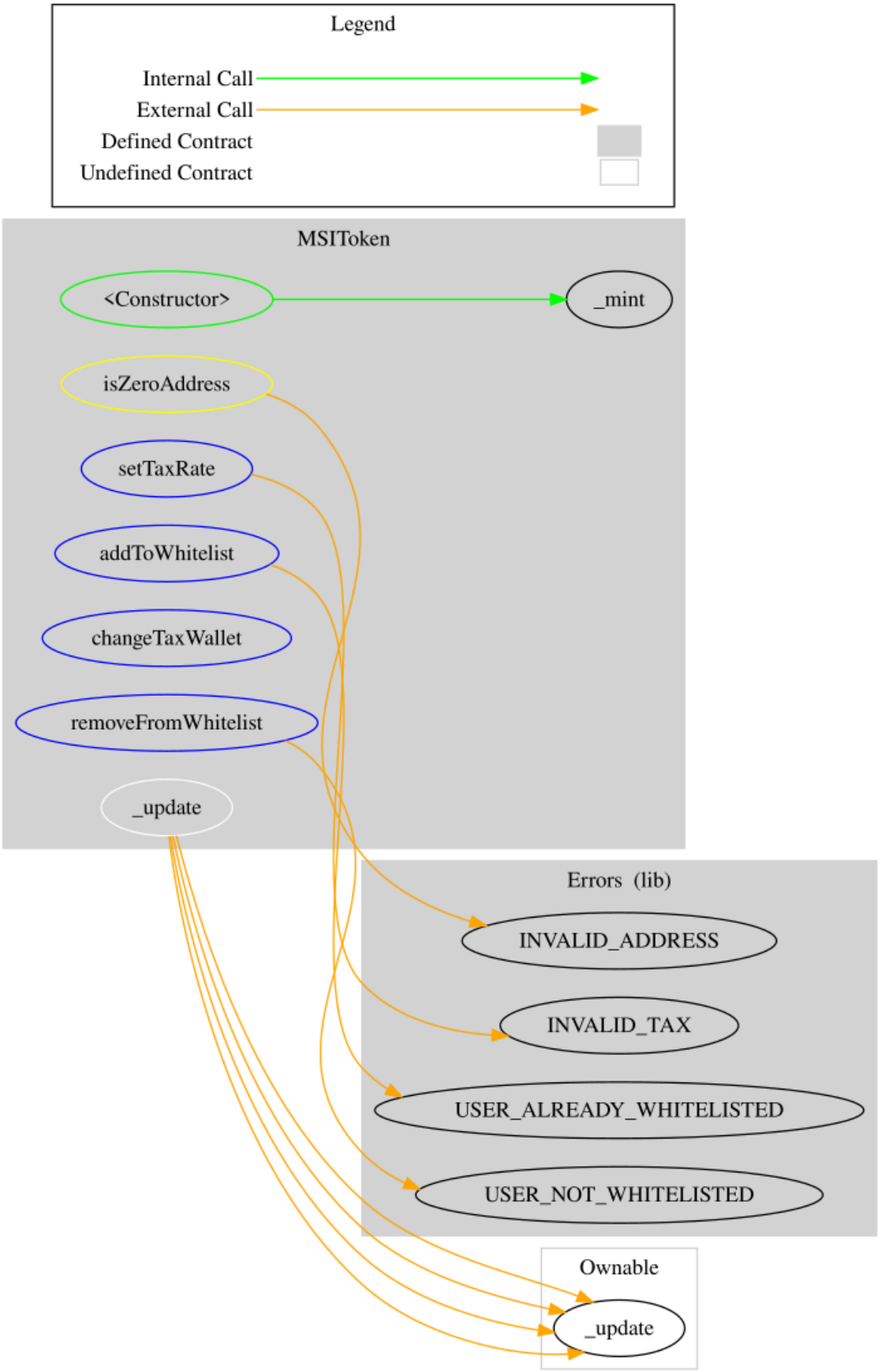
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Errors	Library			
MSIToken	Implementation	ERC20, Ownable		
		Public	✓	isZeroAddress isZeroAddress ERC20 Ownable
	_update	Internal	✓	
	setTaxRate	External	✓	onlyOwner
	changeTaxWallet	External	✓	onlyOwner isZeroAddress
	addToWhitelist	External	✓	onlyOwner isZeroAddress
	removeFromWhitelist	External	✓	onlyOwner isZeroAddress

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
MSIToken
<Constructor>
_mint
isZeroAddress
setTaxRate
addToWhitelist
changeTaxWallet
removeFromWhitelist
_update
Errors (lib)
INVALID_ADDRESS
INVALID_TAX
USER_ALREADY_WHITELISTED
USER_NOT_WHITELISTED
Ownable
_update

Summary

The FairLaunchLimitBlockToken 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

https://www.cyberscope.io