Cyberscope

Audit Report

Captain Pepe

May 2023

SHA256 15c359146a499bbea06bb488256ba3a93abaaec3b0e2ad998d76ad9f59b2d18b

Audited by © cyberscope

Table of Contents

Review

Contract Name	CaptainPepe
Testing Deploy	https://testnet.bscscan.com/address/0x4be39e75073c38f16ef704f987bbcc3f8df2d123
Symbol	CPEPE
Decimals	9
Total Supply	1.000.000.000

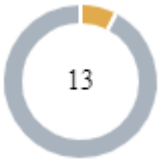
Audit Updates

Initial Audit	12 May 2023 https://github.com/cyberscope-io/audits/blob/main/captain-pepe/v1/audit.pdf
Corrected Phase 2	16 May 2023

Source Files

Filename	SHA256
contracts/contract.sol	15c359146a499bbea06bb488256ba3a93abaaec3b0e2ad998d76ad9f59b2d18b

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	12

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	12	0	0	0

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

ST - Stops Transactions

Criticality	Medium
Location	contracts/contract.sol#L557
Status	Unresolved

Description

The contract owner has the authority to perform transactions when the trading is not open. The owner may take advantage of it by performing transactions when the trading is closed.

```
if (!_isExcludedFromFee[from] && !_isExcludedFromFee[to]) {
  require(tradingEnabled, "Trading not active");
}
```

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. Some suggestions are:

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	MEE	Misleading Event Emission	Unresolved
●	RSC	Redundant SwapAndLiquify Function Call	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

MEE - Misleading Event Emission

Criticality	Minor / Informative
Location	contracts/contract.sol#L611
Status	Unresolved

Description

The contract emits a `Transfer` event with misleading information. The event is emitted when the transfer function call succeeds, but emitted transfer amount does not align with the transferred amount. This may be confusing for users or third-party applications trying to track the movement of tokens on the blockchain.

```
emit Transfer(
    sender,
    address(this),
    s.tLiquidity + s.tMarketing + s.tDev + s.tOps
);
```

Recommendation

It's always a good practice for the contract to provide accurate information about the state of the contract. In this case, the emitted event should accurately reflect the success or failure of the function call. The team is advised to update the contract to emit events that provide accurate information to developers interacting with the contract.

RSC - Redundant SwapAndLiquify Function Call

Criticality	Minor / Informative
Location	contracts/contract.sol#L560
Status	Unresolved

Description

There are code segments that could be optimized. A segment may be optimized so that it becomes a smaller size, consumes less memory, executes more rapidly, or performs fewer operations.

The contract may performs `swapAndLiquify` when the contract balance is zero. Hence it function call is redundant.

```
bool canSwap = balanceOf(address(this)) >= swapTokensAtAmount;
if (
    !swapping &&
    swapEnabled &&
    canSwap &&
    from != pair &&
    !_isExcludedFromFee[from] &&
    !_isExcludedFromFee[to]
) {
    if (to == pair) swapAndLiquify(swapTokensAtAmount,
sellTaxes);
    else swapAndLiquify(swapTokensAtAmount, taxes);
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

The contract could integrate checks to prevent setting the variable `swapTokensAtAmount` to zero. A suggested implementation could check that the

`swapTokensAtAmount` is between an acceptable percentage allocation of the total supply.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	contracts/contract.sol#L651,656,661
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` , `devWallet` , and `opsWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
uint256 marketingAmt = unitBalance * 2 * temp.marketing;
if (marketingAmt > 0) {
    payable(marketingWallet).sendValue(marketingAmt);
}

uint256 devAmt = unitBalance * 2 * temp.dev;
if (devAmt > 0) {
    payable(devWallet).sendValue(devAmt);
}

uint256 opsAmt = unitBalance * 2 * temp.ops;
if (opsAmt > 0) {
    payable(opsWallet).sendValue(opsAmt);
}
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/contract.sol#L206,207
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
router
pair
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	contracts/contract.sol#L139,147
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 private _tTotal = 1e9 * 10**_decimals
address public deadWallet = 0x000000000000000000000000000000000000dEaD
```

Recommendation

Constant state variables can be useful when the contract wants to ensure that the value of a state variable cannot be changed by any function in the contract. This can be useful for storing values that are important to the contract's behavior, such as the contract's address or the maximum number of times a certain function can be called. The team is advised to add the constant keyword to state variables that never change.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/contract.sol#L79,136,144,152,153,177,309,316,715,716,717,718,719,727,728,729,730,731,749,760
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);
uint8 private constant _decimals = 9
uint256 public genesis_block
string private constant _name = "Captain Pepe"
string private constant _symbol = "CPEPE"

...
```

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/contract.sol#L131
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.

```
mapping(address => uint256) private _lastSell
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/contract.sol#L319,746
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
deadline = _deadline
swapTokensAtAmount = amount * 10**_decimals
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/contract.sol#L641,642,649,654,659
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 unitBalance = deltaBalance / (denominator - temp.liquidity)
uint256 opsAmt = unitBalance * 2 * temp.ops
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	contracts/contract.sol#L207
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.

```
pair = _pair
```

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	contracts/contract.sol#L3
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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/contract.sol#L762
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.

```
IBEP20(_tokenAddr).transfer(_to, _amount)
```

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

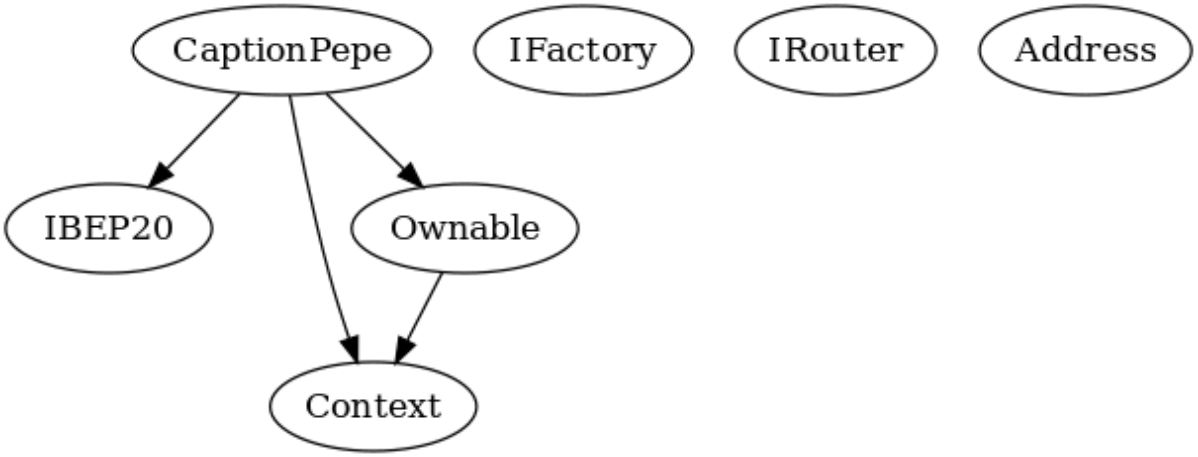
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IBEP20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_setOwner	Private	✓	

IFactory	Interface			
	createPair	External	✓	-
IRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Address	Library			
	sendValue	Internal	✓	
CaptionPepe	Implementation	Context, IBEP20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-

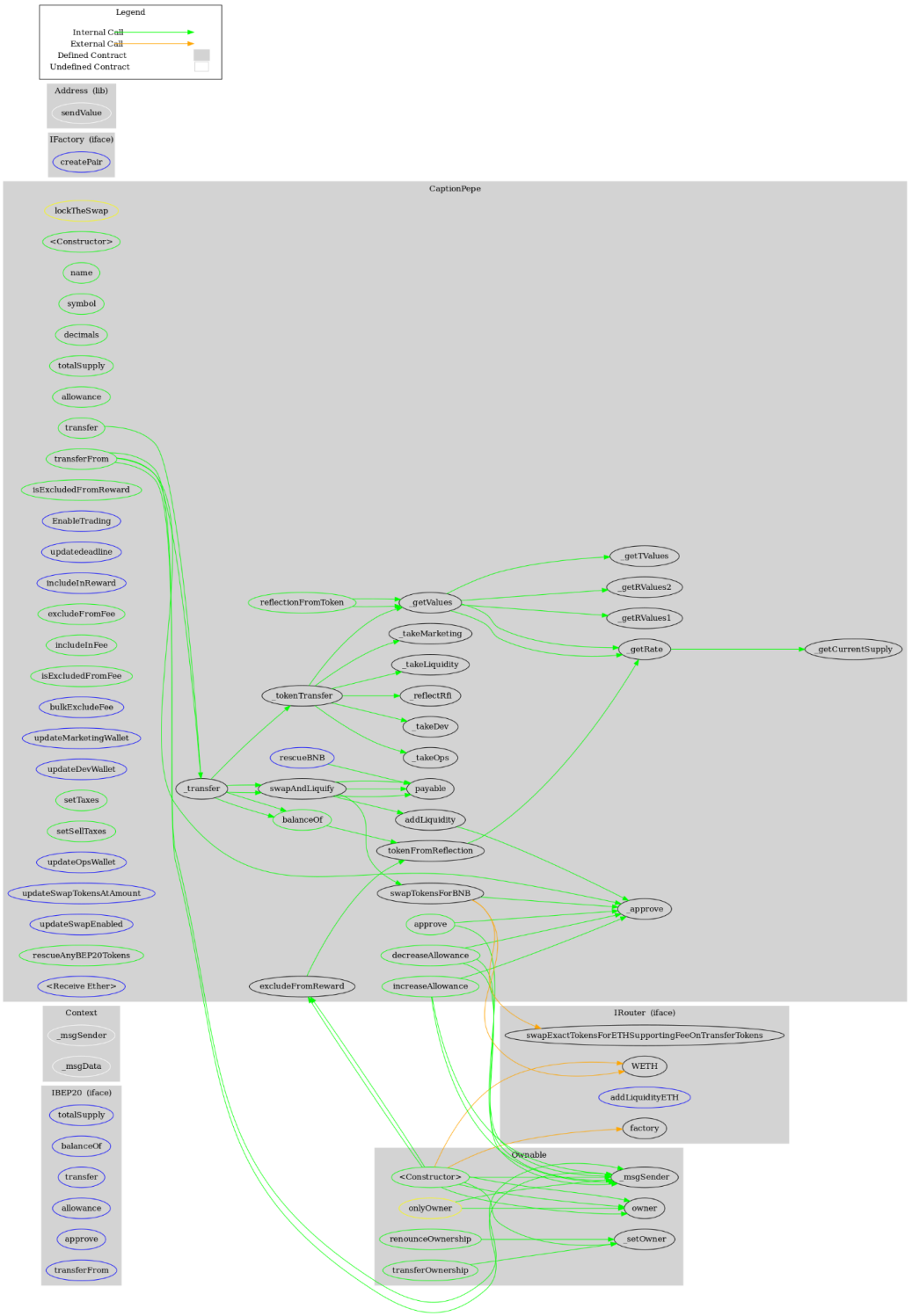
	decreaseAllowance	Public	✓	-
	transfer	Public	✓	-
	isExcludedFromReward	Public		-
	reflectionFromToken	Public		-
	EnableTrading	External	✓	onlyOwner
	updatedeadline	External	✓	onlyOwner
	tokenFromReflection	Public		-
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	isExcludedFromFee	Public		-
	_reflectRfi	Private	✓	
	_takeLiquidity	Private	✓	
	_takeMarketing	Private	✓	
	_takeOps	Private	✓	
	_takeDev	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues1	Private		
	_getRValues2	Private		
	_getRate	Private		
	_getCurrentSupply	Private		

	_approve	Private	✓	
	_transfer	Private	✓	
	_tokenTransfer	Private	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	addLiquidity	Private	✓	
	swapTokensForBNB	Private	✓	
	bulkExcludeFee	External	✓	onlyOwner
	updateMarketingWallet	External	✓	onlyOwner
	updateDevWallet	External	✓	onlyOwner
	setTaxes	Public	✓	onlyOwner
	setSellTaxes	Public	✓	onlyOwner
	updateOpsWallet	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	updateSwapEnabled	External	✓	onlyOwner
	rescueBNB	External	✓	onlyOwner
	rescueAnyBEP20Tokens	Public	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Address (lib)
sendValue
IFactory (iface)
createPair
CaptionPepe
lockTheSwap
<Constructor>
name
symbol
decimals
totalSupply
allowance
transfer
transferFrom
isExcludedFromReward
EnableTrading
updatedeadline
includeInReward
excludeFromFee
includeInFee
isExcludedFromFee
bulkExcludeFee
updateMarketingWallet
updateDevWallet
setTaxes
setSellTaxes
updateOpsWallet
updateSwapTokensAtAmount
updateSwapEnabled
rescueAnyBEP20Tokens
<Receive Ether>
reflectionFromToken
getValues
takeMarketing
takeLiquidity
tokenTransfer
reflectRfi
takeDev
takeOps
rescueBNB
_transfer
swapAndLiquify
payable
balanceOf
addLiquidity
tokenFromReflection
swapTokensForBNB
approve
decreaseAllowance
increaseAllowance
excludeFromReward
getTValues
_getRValues2
_getRValues1
getRate
getCurrentSupply
Context
_msgSender
_msgData
IBEP20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
IRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
addLiquidityETH
factory
Ownable
<Constructor>
onlyOwner
renounceOwnership
transferOwnership
_msgSender
owner
setOwner

Summary

Captain Pepe 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 stopping transactions. A multi-wallet signing pattern will provide security against potential hacks. There is also a limit of max 20% fee. Additionally, the contract utilizes a launch tax. The launch tax can be applied up to the first 5 blocks after the trading opens.

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