Cyberscope

Audit Report

Junkie Cats

April 2024

Network	BSC
Address	0x353d21dabdb6dd97958a95df13e0753a24b561b1
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
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RED	Redudant Event Declaration	Unresolved
●	RFS	Redundant Fee Setting	Unresolved
●	RTC	Redundant Token Calculation	Unresolved
●	UTF	Unutilized Transfer Fee	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved

Table of Contents

Review

Contract Name	Junkie
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x353d21dabdb6dd97958a95df13e0753a24b561b1
Address	0x353d21dabdb6dd97958a95df13e0753a24b561b1
Network	BSC
Symbol	Junkie
Decimals	18
Total Supply	420,000,000,000,000
Badge Eligibility	Yes

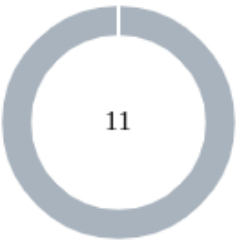
Audit Updates

Initial Audit	06 Apr 2024

Source Files

Filename	SHA256
Junkie.sol	d5db8b991b6056ac02e3043855cb2a6fdf84ef27ef274b372da9721c9dd5d684

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	11

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	11	0	0	0

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	Junkie.sol#L786
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTok
ens(
    tokenAmount,
    0, // accept any amount of ETH
    path,
    address(this), // The contract
    block.timestamp
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Junkie.sol#L731
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
        uint256 contractTokenBalance = balanceOf(address(this));
         bool overMinimumTokenBalance = contractTokenBalance >=
             minimumTokensBeforeSwap;
         ...
         if (
             !inSwapAndLiquify &&
             !fromLP &&
             overMinimumTokenBalance &&
             swapAndLiquifyEnabled
         ) {
             swapAndLiquify();
         }
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	Junkie.sol#L545
Status	Unresolved

Description

The contract uses events that are not emitted within the contract's functions. As a result, these declared events are redundant and serve no purpose within the contract's current implementation.

```
event UpdateTransferFee(uint256);
```

Recommendation

To optimize contract performance and efficiency, it is advisable to regularly review and refactor the codebase, removing the unused event declarations. This proactive approach not only streamlines the contract, reducing deployment and execution costs but also enhances readability and maintainability.

RFS - Redundant Fee Setting

Criticality	Minor / Informative
Location	Junkie.sol#L565,860
Status	Unresolved

Description

The contract contains a fee structure to impose certain percentages on buy and sell transactions, specifically setting both the `buyFee` and `sellFee` to 5%. Additionally, the contract includes a `setFee` function, which is intended to allow the contract owner to adjust these fees. However, the `setFee` function redundantly sets both `buyFee` and `sellFee` to 5%, the same values initially declared at the contract's deployment. This redundancy indicates that the `setFee` function does not effectively alter the fee structure or provide any additional utility in its current form, as it merely reassigns the fees to their original values.

```
uint public buyFee = 5;
uint256 public transferFee = 0;
uint256 public sellFee = 5;
function setFee() external onlyOwner {
    require(sellFee != 0, "Sell Fee is already 0%");
    require(buyFee != 0, "Buy Fee is alreazy 0%");
    sellFee = 5;
    buyFee = 5;
    emit UpdateBuyFee(buyFee);
    emit UpdateSellFee(sellFee);
}
```

Recommendation

It is recommended to reconsider the functionality of the `setFee` function. If the intention behind this function is to provide the contract owner with the ability to adjust fees to new values, then it should be modified to accept parameters that allow setting different values for `buyFee` and `sellFee`. This modification would enable the contract to adapt to changing economic conditions or strategic requirements. On the other hand, if the fee values are intended to remain static, or if there is no foreseeable need to adjust them in the

future, it may be prudent to consider the removal of the `setFee` function altogether. Removing or modifying the `setFee` function to add meaningful functionality would streamline the contract's codebase, reduce potential confusion, and enhance the contract's overall efficiency and clarity.

RTC - Redundant Token Calculation

Criticality	Minor / Informative
Location	Junkie.sol#L731,767
Status	Unresolved

Description

The contract is designed to manage token balances and perform liquidity operations through its `swapAndLiquify` function. The contract declares the `contractTokenBalance` variable, which is intended to store the current token balance of the contract by calling `balanceOf(address(this))`. This variable is strategically checked against a predefined threshold (`minimumTokensBeforeSwap`) to determine if the contract holds enough tokens to initiate the swap and liquidity addition process. However, despite having the `contractTokenBalance` variable readily available and already calculated, the contract proceeds to redundantly recalculate the total token balance within the `swapAndLiquify` function through the statement `totalTokens = balanceOf(address(this))`. This redundancy not only introduces unnecessary computational overhead but also increases the potential for errors and inconsistencies in balance management.

```
        uint256 contractTokenBalance = balanceOf(address(this));
         bool overMinimumTokenBalance = contractTokenBalance >=
             minimumTokensBeforeSwap;
        ...
         if (
             !inSwapAndLiquify &&
             !fromLP &&
             overMinimumTokenBalance &&
             swapAndLiquifyEnabled
         ) {
             swapAndLiquify();
         }
         ...

    function swapAndLiquify() internal lockTheSwap {
         uint256 totalTokens = balanceOf(address(this));
         swapTokensForEth(totalTokens);
         ...
```

Recommendation

It is recommended that the contract utilize the `contractTokenBalance` variable directly within the `swapAndLiquify` function instead of recalculating the token balance. Leveraging the already computed `contractTokenBalance` will optimize the contract's efficiency by eliminating redundant balance queries to the blockchain. This approach ensures that the token balance used across different parts of the contract is consistent, reducing the risk of discrepancies. Furthermore, optimizing the contract in this manner can lead to reduced gas costs for transactions involving the `swapAndLiquify` function, thereby benefiting users with lower transaction fees. Adopting this recommendation will enhance the contract's performance, reliability, and user experience.

UTF - Unutilized Transfer Fee

Criticality	Minor / Informative
Location	Junkie.sol#L754
Status	Unresolved

Description

The contract is designed to incorporate a fee mechanism for transfer transactions, as indicated by the presence of the `transferFee` variable. This variable is intended to specify the percentage fee applied to transactions that are not between excluded addresses, thereby playing a crucial role in the contract's fee management strategy. However, the `transferFee` variable is initialized to zero and remains unchanged throughout the contract's implementation. This initialization effectively neutralizes the intended functionality of applying a fee on transfer transactions, rendering the `transferFee` variable redundant. The presence of code intended to calculate and apply a fee based on the `transferFee` variable suggests that its utilization was anticipated. Yet, without any assignment of a non-zero value to `transferFee`, this part of the contract's logic does not fulfill its purpose, leading to a discrepancy between the contract's design and its operational behavior.

```
else if (NOTtoExcluded && NOTfromExcluded) {
      fee = (transferFee * amount) / 100;
```

Recommendation

It is recommended to reconsider the value of the `transferFee` variable. If the intention behind the fee mechanism is to apply a non-zero fee on certain transfer transactions, the `transferFee` variable should be set to an appropriate value that reflects this policy. This adjustment would activate the fee mechanism as originally intended, aligning the contract's functionality with its design objectives. Alternatively, if there is no intention to levy a fee on transfer transactions, or if the current strategy has evolved away from using such a fee, it may be advisable to remove the `transferFee` variable and associated fee calculation logic to simplify the contract and eliminate any confusion regarding its fee policies. This

approach would clarify the contract's fee structure and ensure that its codebase accurately represents its operational mechanics.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Junkie.sol#L556,565
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 = 420_000_000_000_000 ether
uint256 public transferFee = 0
```

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	Junkie.sol#L265,267,297,337,577,662,841,851,857,901,902
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint256);
function WETH() external pure returns (address);
address private immutable WETH
address _owner
uint256 _minimumTokensBeforeSwap
bool _enabled
address _marketingWallet
address _tokenAddress
uint256 _amount
```

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	Junkie.sol#L821,831
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(
            _isPresaleNoFee[account] != true,
            "The wallet is already excluded!"
        )

require(
            _isPresaleNoFee[account] != false,
            "The wallet is already included!"
        )
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Junkie.sol#L590
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.

```
address currentRouter
```

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	Junkie.sol#L662,720
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.

```
address _owner
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
ReentrancyGuard	Implementation			
		Public	✓	-
	_nonReentrantBefore	Private	✓	
	_nonReentrantAfter	Private	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Ownable	Implementation	Context		
		Public	✓	-

	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-

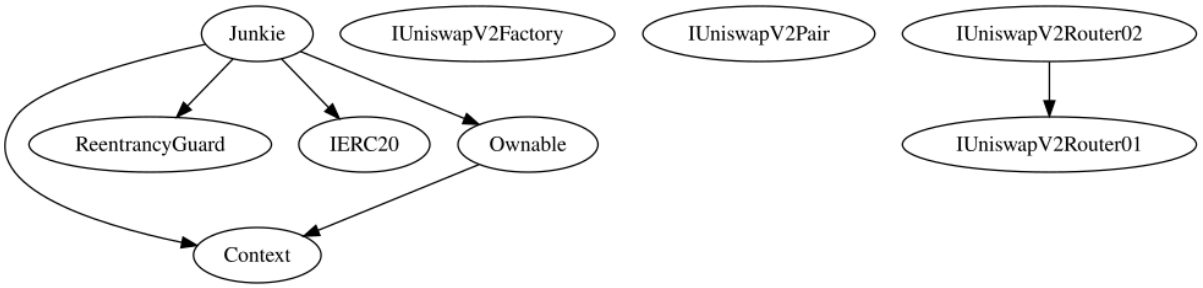
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-
	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-

	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-

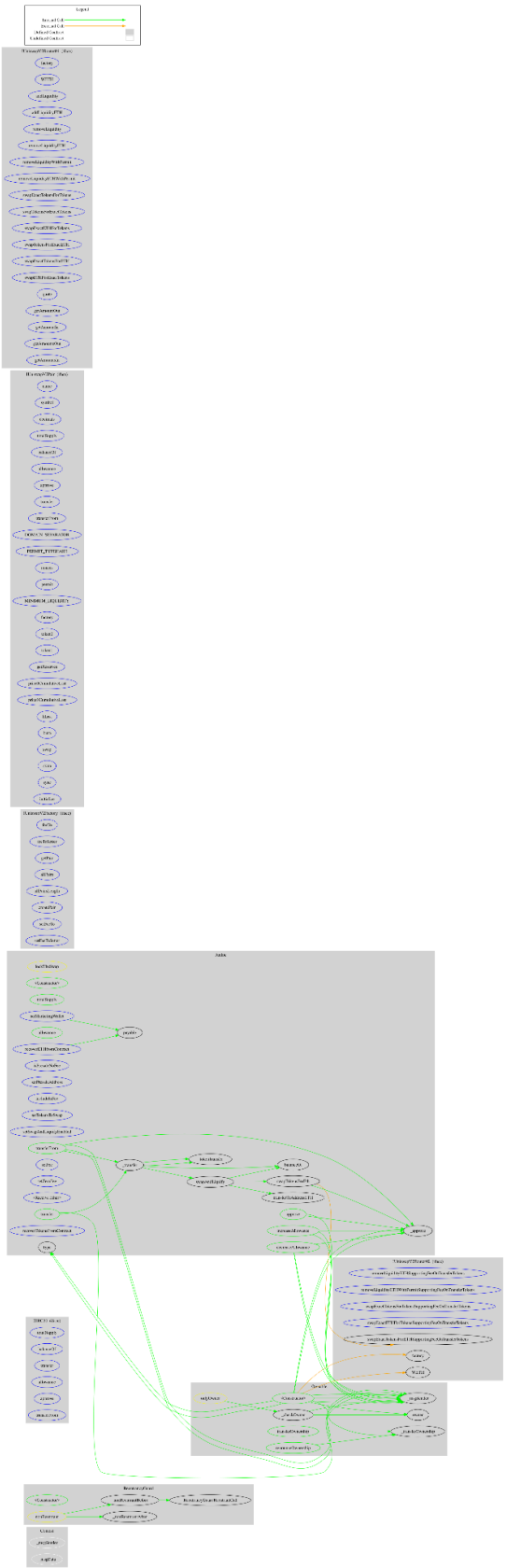
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Junkie	Implementation	Context, IERC20, Ownable, ReentrancyGuard		
		Public	✓	-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	swapAndLiquify	Internal	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	_tokenTransfer	Private	✓	
	isPresaleNoFee	External		-
	setPresaleAddress	External	✓	onlyOwner

	includeInFee	External	✓	onlyOwner
	setTokensToSwap	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	External	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setFee	External	✓	onlyOwner
	setZeroFee	External	✓	onlyOwner
	transferToAddressETH	Private	✓	
		External	Payable	-
	recoverETHfromContract	External	✓	nonReentrant
	recoverTokensFromContract	External	✓	nonReentrant

Inheritance Graph



Flow Graph



Summary

Junkie Cats contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Junkie Cats is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error 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. There is also a limit of max 5% fees.

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