Cyberscope

Audit Report

CROWDBIT ACADEMY TOKEN

May 2025

Network BSC

Address 0xd231533d291542a75668841efa87683cf8e35fa2

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
	PVC	Price Volatility Concern	Unresolved
	RL	Redundant Libraries	Unresolved
	CC	Commented Code	Unresolved
	HV	Hardcoded Values	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	ME	Misleading Events	Unresolved
	MEM	Missing Error Messages	Unresolved
	MEE	Missing Events Emission	Unresolved
	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
	RED	Redundant Event Declaration	Unresolved
	RRA	Redundant Repeated Approvals	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSRS	Redundant SafeMath Require Statement	Unresolved
	UF	Unused Functionality	Unresolved

●	UV	Unused Variables	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

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	CBA_Token
Compiler Version	v0.8.28+commit.7893614a
Optimization	200 runs
Explorer	https://bscscan.com/address/0xd231533d291542a75668841efa87683cf8e35fa2
Address	0xd231533d291542a75668841efa87683cf8e35fa2
Network	BSC
Symbol	CBA
Decimals	18
Total Supply	1.000.000.000

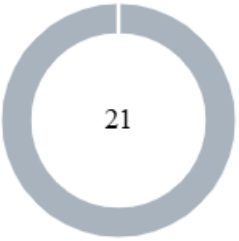
Audit Updates

Initial Audit	04 May 2025

Source Files

Filename	SHA256
CBA_Token.sol	1c96ea5faa2342727f4d565290b505ad925c10f147dbfd9781b3b46e0b91480b

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	21

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	21	0	0	0

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	CBA_Token.sol#L726
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. The contract's `swapTokensAtAmount` is equal to 20% of the total supply.

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.

```
bool canSwap = contractTokenBalance >= swapTokensAtAmount;
```

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.

RL - Redundant Libraries

Criticality	Minor / Informative
Location	CBA_Token.sol#L328,373
Status	Unresolved

Description

The file contains the `SafeMathInt` and `SafeMathUint` libraries however they are never used in the actual implementation therefore they are redundant.

```
library SafeMathInt {/*...*/}
library SafeMathUint {/*...*/}
```

Recommendation

It is recommended to remove redundant libraries from the code to enhance its optimization and readability.

CC - Commented Code

Criticality	Minor / Informative
Location	CBA_Token.sol#L565,627,646,675,696
Status	Unresolved

Description

The contract contains commented code. These comments hinder the understanding of the source code.

```
/*
function updateFeeReciever(address newFeeReciever) external
onlyOwner {
    emit feeRecieverUpdated(newFeeReciever, feeReciever);
    feeReciever = newFeeReciever;
}
*/
```

Recommendation

The team is advised to carefully review the comment and remove commented code to improve code readability.

HV - Hardcoded Values

Criticality	Minor / Informative
Location	CBA_Token.sol#L815,816
Status	Unresolved

Description

The contract contains instances where numeric values are directly hardcoded into the code logic rather than being assigned to constant variables with descriptive names. Hardcoding such values can lead to several issues, including reduced code readability, increased risk of errors during updates or maintenance, and difficulty in consistently managing values throughout the contract. Hardcoded values can obscure the intent behind the numbers, making it challenging for developers to modify or for users to understand the contract effectively.

```
if(contractBalance > swapTokensAtAmount * 20){
    contractBalance = swapTokensAtAmount * 20;
}
```

Recommendation

It is recommended to replace hardcoded numeric values with variables that have meaningful names. This practice improves code readability and maintainability by clearly indicating the purpose of each value, reducing the likelihood of errors during future modifications. Additionally, consider using constant variables which provide a reliable way to centralize and manage values, improving gas optimization throughout the contract.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	CBA_Token.sol#L581,582,585,586,588,589,591,605,607,608
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.

```
maxTransactionAmount
maxWallet
buyBoostTVLFee
buyTotalFees
sellBoostTVLFee
sellTotalFees
feeReciever
tradingActive
launchedAt
limitsInEffect
```

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.

ME - Misleading Events

Criticality	Minor / Informative
Location	CBA_Token.sol#L561
Status	Unresolved

Description

The contract is using misleading comment events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. These events do not accurately reflect the actual implementation.

```
event SwapAndLiquify(
    uint256 tokensSwapped
);
```

Recommendation

The team is advised to carefully review the events in order to reflect their actual purpose. This can be useful for debugging and for providing more information to users that interact with the contract.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	CBA_Token.sol#L831,836
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(msg.sender==feeReciever)
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	CBA_Token.sol#L620,642,830,835
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
function updateSwapTokensAtAmount(uint256 newAmount) external
onlyOwner returns (bool)
function updateSwapEnabled(bool enabled) external onlyOwner()
function rescueETH() external
function rescueERC20(address erc20) external
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	CBA_Token.sol#L784
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.swapExactTokensForETHSupportingFeeOnTransferTokens(token
Amount, 0, path, address(this), 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.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	CBA_Token.sol#L553
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 UpdateUniswapV2Router(address indexed newAddress, address indexed oldAddress);
```

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.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	CBA_Token.sol#L781
Status	Unresolved

Description

The contract is designed to `approve` token transfers during the contract's operation by calling the _approve function before specific operations. This approach results in additional gas costs since the approval process is repeated for every operation execution, leading to inefficiencies and increased transaction expenses.

```
_approve(address(this), address(uniswapV2Router), tokenAmount);
uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(token
Amount, 0, path, address(this), block.timestamp);
```

Recommendation

Since the approved address is a trusted third-party source, it is recommended to optimize the contract by approving the maximum amount of tokens once in the initial set of the variable, rather than before each operation. This change will reduce the overall gas consumption and improve the efficiency of the contract.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	CBA_Token.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 in cases where the explanatory error message is not used.

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

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSRS - Redundant SafeMath Require Statement

Criticality	Minor / Informative
Location	CBA_Token.sol#L244
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns
(uint256) {
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");
      return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

UF - Unused Functionality

Criticality	Minor / Informative
Location	CBA_Token.sol#L793
Status	Unresolved

Description

The contract has the private `addLiquidity` function. Since it is private it cannot be used externally and also it is not used in any of the contract's functions, therefore it is redundant.

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount)
private {
    // approve token transfer to cover all possible scenarios
    _approve(address(this), address(uniswapV2Router),
tokenAmount);

    // add the liquidity
    uniswapV2Router.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        deadAddress,
        block.timestamp
    );
}
```

Recommendation

It is recommended to remove unused functions to increase code readability and optimization

UV - Unused Variables

Criticality	Minor / Informative
Location	CBA_Token.sol#L527,543,756,761,810,821
Status	Unresolved

Description

The contract contains multiple variables that are set once and are never used again in the contract.

```
uint256 public maxTransactionAmount;
uint256 public swapTokensAtAmount;
uint256 public maxWallet;
bool public limitsInEffect = true;
bool public tradingActive = false;
uint256 launchedAt;
```

Additionally, there are variables that are used in other functions but do not provide additional utility.

`tokensForBoostTVL` is updated during the transfer and is reset to zero during `swapBack` but it does not provide anything in the functionality. The case is the same with the local variable `totalTokensToSwap`.

The `buyBoostTVLFee` and `sellBoostTVLFee` are both used in the `_transfer` function to calculate the amount to be added in `tokensForBoostTVL` but since they are permanently equal to `sellTotalFees` and `buyTotalFees` the amount will always be the entire fee.

```
tokensForBoostTVL += fees * sellBoostTVLFee / sellTotalFees;
tokensForBoostTVL += fees * buyBoostTVLFee / buyTotalFees;
uint256 totalTokensToSwap = tokensForBoostTVL;
tokensForBoostTVL = 0;
```

Recommendation

It is recommended to restructure the code so that it contains only variables that are used in the contract.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	CBA_Token.sol#L30,31,48,384,517,559
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 (uint);
function WETH() external pure returns (address);

...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	CBA_Token.sol#L623
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.

```
swapTokensAtAmount = newAmount
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	CBA_Token.sol#L211,793
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 _burn(address account, uint256 amount) internal
virtual {
        require(account != address(0), "ERC20: burn from the
zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount,
"ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

...
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	CBA_Token.sol#L755,756,760,761
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.

```
fees = amount.mul(buyTotalFees).div(100)
tokensForBoostTVL += fees * buyBoostTVLFee / buyTotalFees
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	CBA_Token.sol#L579
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.

```
uint256 totalSupply = 1000000000 * 1e18
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	CBA_Token.sol#L837
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(erc20).transfer(msg.sender,
IERC20(erc20).balanceOf(address(this)))
```

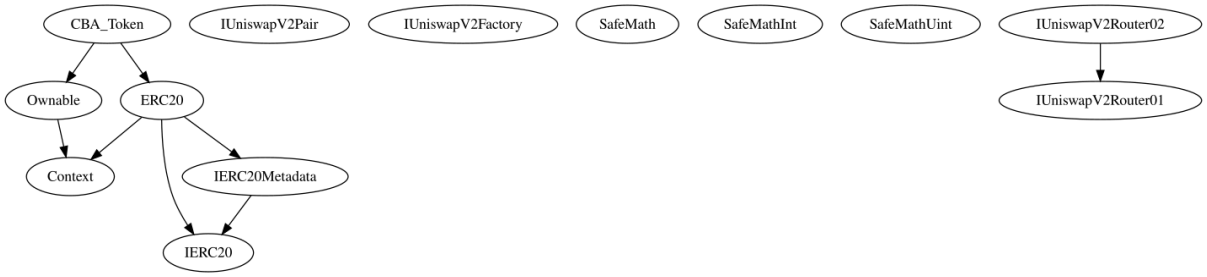
Recommendation

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.

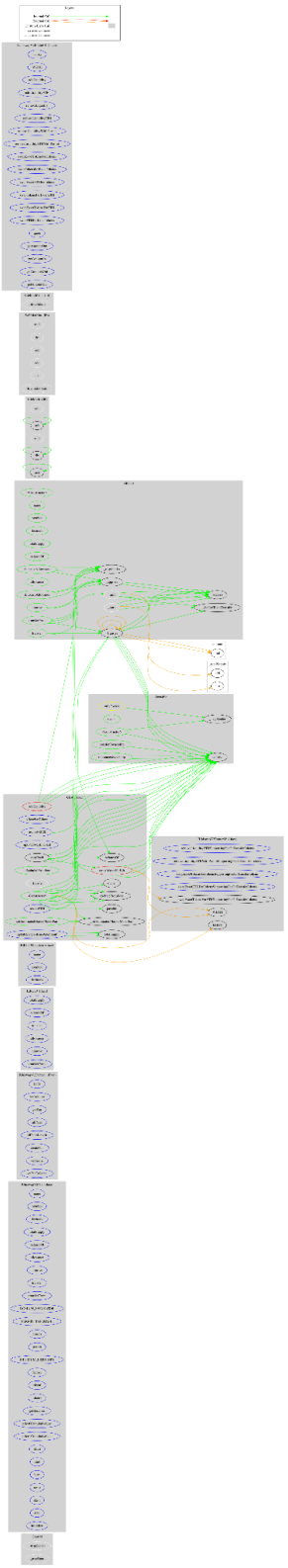
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
CBA_Token	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	updateSwapTokensAtAmount	External	✓	onlyOwner
	updateSwapEnabled	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	isExcludedFromFees	Public		-
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	swapBack	Private	✓	
	rescueETH	External	✓	-
	rescueERC20	External	✓	-

Inheritance Graph



Flow Graph



Summary

CROWDBIT ACADEMY TOKEN contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. CROWDBIT ACADEMY TOKEN is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. There are also 10% fees on buys and sells.

The contract has renounced the ownership so it no longer has an assigned owner and consequently, the owner's privileges and authority are revoked. As a result, the owner is unable to execute any methods that are designated exclusively for owner access. By relinquishing ownership, the contract eliminates the potential risks associated with centralized authority, reducing the possibility of the owner misusing their privileges or becoming a single point of failure. It is important to note that renouncing ownership is an irreversible action, and once executed, it cannot be undone.

The ownership has been renounced on this transaction:

https://bscscan.com/tx/0x9d983cc12a65c77227fac202eb3d67320f6cbf30e22f6971dfeffa488a04af11

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