Cyberscope

Audit Report

UNITY

May 2025

Network	BSC
Address	0x6a07d5c7464b59a5c2a54eaf8870d72d2c3c9629
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
●	LVU	Local Variable Usage	Unresolved
●	RRA	Redundant Repeated Approvals	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	UZA	Unnecessary Zero Addition	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	RCS	Redundant Conditional Statements	Unresolved
●	RSD	Redundant Swap Duplication	Unresolved
●	UDO	Unnecessary Decimals Override	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	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	Unity
Compiler Version	v0.8.25+commit.b61c2a91
Optimization	200 runs
Explorer	https://bscscan.com/address/0x6a07d5c7464b59a5c2a54eaf8870d72d2c3c9629
Address	0x6a07d5c7464b59a5c2a54eaf8870d72d2c3c9629
Network	BSC
Symbol	UNT
Decimals	18
Total Supply	500.000.000
Badge Eligibility	Yes

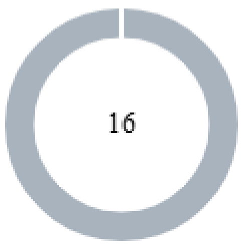
Audit Updates

Initial Audit	25 Apr 2025 https://github.com/cyberscope-io/audits/blob/main/2-unt/v1/audit.pdf
Corrected Phase 2	02 May 2025

Source Files

Filename	SHA256
Token.sol	754cd926d01f171eeac401ad40151087c7358d0349ed99cd7d7b99fef694810c

Findings Breakdown



🔴	Critical	0
🟠	Medium	0
●	Minor / Informative	16

Severity		Unresolved	Acknowledged	Resolved	Other
🔴	Critical	0	0	0	0
🟠	Medium	0	0	0	0
●	Minor / Informative	16	0	0	0

LVU - Local Variable Usage

Criticality	Minor / Informative
Location	Token.sol#L91,113
Status	Unresolved

Description

In the `constructor`, `supplyRecipient` is assigned a specific address, but instead of using the variable, the hardcoded address is directly passed to `_transferOwnership`.

```
constructor()
    ERC20(unicode"Unity", unicode"UNT")
    Ownable(msg.sender)
{
    address supplyRecipient =
0x6CF7fbBF8F2b73035E17A13c9e86cc0e2dD9F136;
    //...

_transferOwnership(0x6CF7fbBF8F2b73035E17A13c9e86cc0e2dD9F136);
}
```

Recommendation

It is recommended to reduce the times a hardcoded value is used and use local variables instead.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	Token.sol#L145
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.

```
function _swapTokensForCoin(uint256 tokenAmount) private {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = routerV2.WETH();

    _approve(address(this), address(routerV2), tokenAmount);


routerV2.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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	Token.sol#L152
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.

```
routerV2.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.

UZA - Unnecessary Zero Addition

Criticality	Minor / Informative
Location	Token.sol#L168,321
Status	Unresolved

Description

In the contract there are occasions that hardcoded zeros are added to calculations that return addition results. Zero additions are redundant as they do not change the result.

```
return 0 + _projectsPending + _marketingPending +
_rewardsPending;
uint256 token2Swap = 0 + _projectsPending + _marketingPending;
```

Recommendation

It is recommended to remove zero additions as they are redundant and decrease the efficiency and readability of the code.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	Token.sol#L301,303,305,336
Status	Unresolved

Description

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

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

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

```
_marketingPending += fees * marketingFees[txType] /
totalFees[txType];
_projectsPending += fees * projectsFees[txType] /
totalFees[txType];
_rewardsPending += fees * rewardsFees[txType] /
totalFees[txType];
//...
uint256 marketingPortion = coinsReceived * _marketingPending /
token2Swap;
```

Recommendation

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

RCS - Redundant Conditional Statements

Criticality	Minor / Informative
Location	Token.sol#L320
Status	Unresolved

Description

The contract contains redundant conditional statements that can be simplified to improve code efficiency and performance. Conditional statements that merely return the result of an expression are unnecessary and lead to larger code size, increased memory usage, and slower execution times. By directly returning the result of the expression, the code can be made more concise and efficient, reducing gas costs and improving runtime performance. Such redundancies are common when simple comparisons or checks are performed within conditional statements, leading to redundant operations.

Specifically, there is an `if(false || ...)` statement which does not provide any functionality.

```
if (false || _projectsPending > 0 || _marketingPending > 0)
```

Recommendation

It is recommended to refactor conditional statements that return results by eliminating unnecessary code structures and directly returning the outcome of the expression. This practice minimizes the number of operations required, reduces the code footprint, and optimizes memory and gas usage. Simplifying such statements makes the code more readable and improves its overall performance.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	Token.sol#L324,348
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

Specifically, in `_update` method a swap is triggered in `_swapTokensForCoin` and in `_sendDividends`.

```
function _update(address from, address to, uint256 amount)
    internal
    override
{
    //..
    if (false || _projectsPending > 0 || _marketingPending > 0)
{
            //...
            _swapTokensForCoin(token2Swap);
            //...
    }
    //..
    if (_rewardsPending > 0 &&
getNumberOfDividendTokenHolders() > 0) {
        _sendDividends(_rewardsPending);
        //...
    }
    //..
}
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

UDO - Unnecessary Decimals Override

Criticality	Minor / Informative
Location	Token.sol#L123
Status	Unresolved

Description

The contract is currently implementing an override of the decimals function, which simply returns the value 18. This override is redundant since the extending token contract already specifies 18 decimals as its standard. In the context of ERC-20 tokens, 18 decimals is a common default, and overriding this function to return the same value adds unnecessary complexity to the contract. This redundancy does not contribute to the functionality of the contract and could potentially lead to confusion about the necessity of this override.

```
function decimals() public pure override returns (uint8) {
    return 18;
}
```

Recommendation

Since the inherited ERC-20 contract already defines the decimals number, maintaining an overriding function that merely repeats this value does not contribute to the contract's effectiveness. As a result, it is recommended to remove the redundant `decimals` function from the contract. Removing this function will simplify the contract, making it more straightforward to maintain without impacting its operational capabilities.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	CoinDividendTracker.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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	Token.sol#L56,119,155,171,180,191,200,230,257 SafeERC20Remastered.sol#L29 IUniswapV2Router01.sol#L5 IUniswapV2Pair.sol#L18,19,36 CoinDividendTracker.sol#L201,422
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.

```
pping (address => bool) public AMMs;

dress _router)
nt16 _swapThresholdRatio)
dress _newAddress)
nt16 _buyFee,
...
dress AMM,

function safeTransfer_noRevert(IERC20 token, address to,
uint256 value) internal returns (bool) {
        return _callOptionalReturnBool(token,
abi.encodeCall(token.transfer, (to, value)));
    }
function WETH() external pure returns (address);
function DOMAIN_SEPARATOR() external view returns (bytes32);

...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	Context.sol#L25
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 _contextSuffixLength() internal view virtual returns
(uint256) {
        return 0;
    }
```

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	Token.sol#L298,301,303,305
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.

```
es = amount * totalFees[txType] / 10000;

ewardsPending += fees * rewardsFees[txType] /
totalFees[txType];
```

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	CoinDividendTracker.sol#L213
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.

```
string memory _name
string memory _symbol
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	Token.sol#L90 Address.sol#L151
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
sembly { if iszero(extcodesize(caller())) { revert(0, 0) } }


assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	Token.sol#L16 SafeERC20Remastered.sol#L4 Ownable2Step.sol#L4 Ownable.sol#L4 IUniswapV2Router02.sol#L1 IUniswapV2Router01.sol#L1 IUniswapV2Pair.sol#L1 IUniswapV2Factory.sol#L1 Initializable.sol#L3 IERC20Metadata.sol#L4 IERC20.sol#L4 ERC20Burnable.sol#L4 ERC20.sol#L4 draft-IERC6093.sol#L3 Context.sol#L4 CoinDividendTracker.sol#L5 Address.sol#L4
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.20;
pragma solidity ^0.8.19;
pragma solidity >=0.5.0;
pragma solidity >=0.6.2;
agma solidity 0.8.25;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Initializable.sol#L3 Context.sol#L4
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;
pragma solidity ^0.8.20;
```

Recommendation

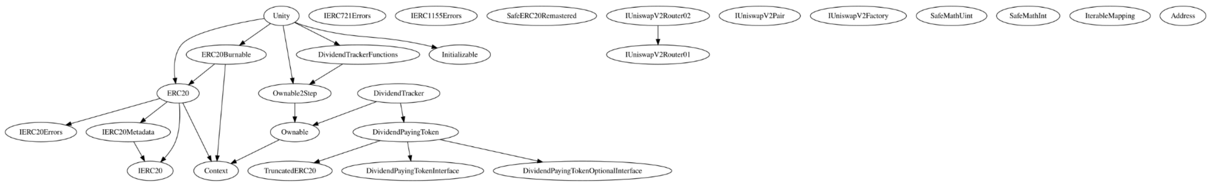
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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

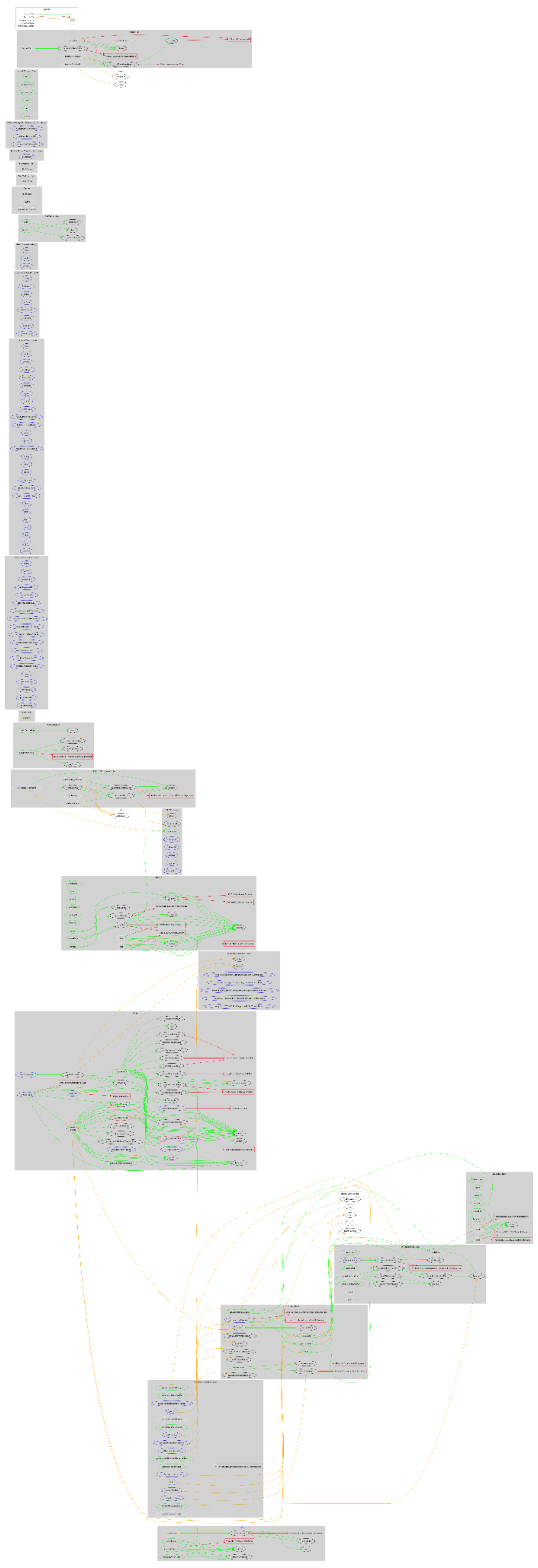
Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Unity	Implementation	ERC20, ERC20Burnable, Ownable2Step, DividendTrackerFunctions, Initializable		
		Public	✓	ERC20 Ownable
	afterConstructor	External	✓	initializer
	decimals	Public		-
	recoverToken	External	✓	onlyOwner
	recoverForeignERC20	External	✓	onlyOwner
		External	Payable	-
	_swapTokensForCoin	Private	✓	
	updateSwapThreshold	Public	✓	onlyOwner
	getSwapThresholdAmount	Public		-
	getAllPending	Public		-
	marketingAddressSetup	Public	✓	onlyOwner
	marketingFeesSetup	Public	✓	onlyOwner
	projectsAddressSetup	Public	✓	onlyOwner
	projectsFeesSetup	Public	✓	onlyOwner
	_sendDividends	Private	✓	
	excludeFromDividends	External	✓	onlyOwner

	_excludeFromDividends	Internal	✓	
	rewardsFeesSetup	Public	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	_updateRouterV2	Private	✓	
	setAMM	External	✓	onlyOwner
	_setAMM	Private	✓	
	_update	Internal	✓	
	_beforeTokenUpdate	Internal		
	_afterTokenUpdate	Internal	✓	

Inheritance Graph



Flow Graph



Summary

UNITY contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. The fees are permanently set to 4% for buys and sells. Transfers have 0% fees.

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/0xe7216cfc34179014a78a5152ed7754bc7e30ab962f8907cb8e6f5efe885b3ee8

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