Cyberscope

Audit Report

Starlink Satire

July 2023

Network	BSC
Address	0xec87844448a05a6856799dBb78aa2142dA234a97
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	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
●	TAM	Transfer Amount Miscalculation	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	StarlinkSatire_BEP20
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0xec87844448a05a6856799dbb78aa2142da234a97
Address	0xec87844448a05a6856799dbb78aa2142da234a97
Network	BSC
Symbol	STR
Decimals	18
Total Supply	1,000,000,000,000

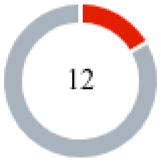
Audit Updates

Initial Audit	02 Jul 2023

Source Files

Filename	SHA256
StarlinkSatire_BEP20.sol	65a679eade6f42b0641b27226cda8f3afd5b0aa7ef063593aede8da8d0c384a3

Findings Breakdown



●	Critical	2
●	Medium	0
●	Minor / Informative	10

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	10	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	StarlinkSatire_BEP20.sol#L200,208
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the owner. The owner can enable the transactions for all users. Once the transactions are enabled the owner will not be able to disable them again.

```
if (from == pairAddress) {
    if (tradeEnabled == true || msg.sender == _owner) {
        _balances[from] = fromBalance - amount;
        _balances[to] += amount / 100 * 100 - btax;  // WE ARE GETTING TAX
WITH THIS LINE, THIS IS BUY TAX AND IT %5
        _balances[address(this)] += amount / 100;
    } else {
        revert("Trade is not opened");
    }
} else if (to == pairAddress) {
    if (tradeEnabled == true || msg.sender == _owner) {
        _balances[from] = fromBalance - amount;
        _balances[to] += amount / 100 * 100 - stax;  // WE ARE GETTING TAX
WITH THIS LINE, THIS IS BUY TAX AND IT %5
        _balances[address(this)] += amount / 100;
    } else {
        revert("Trade is not opened");
    }
}
```

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.

TAM - Transfer Amount Miscalculation

Criticality	Critical
Location	StarlinkSatire_BEP20.sol#L198,206
Status	Unresolved

Description

As part of the transfer flow, a tax is applied to the sender on buys and sales. However, the token distribution appears to be miscalculated, resulting in an incorrect addition of tokens to the contract's address, as well as, an incorrect deduction from the sender's balance. The following example demonstrates the miscalculation. Assume the contract is in the following state:

- Sender's balance: 200
- Recipient's balance: 0
- Contract's balance: 0
- Transfer amount: 200
- Tax: 5%

Based on the current implementation:

1. Sender's Balance: 200 - 200 = 0 tokens
2. Recipient's Balance: 0 + (200 / 100 * 100 - 5) = 195 tokens
3. Contract's Address Balance: 0 + (200 / 100) = 2 tokens

As a result, a portion of the transferred tokens will be lost.

```
_balances[from] = fromBalance - amount;
_balances[to] += amount / 100 * 100 - btax;  // WE ARE GETTING TAX WITH
THIS LINE, THIS IS BUY TAX AND IT %5
_balances[address(this)] += amount / 100;
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the calculations are executed as expected and not tokens are lost during transactions. A recommended approach would be the following:

```
_balances[from] = fromBalance - amount;
uint taxAmount = amount * btax / 100;
_balances[to] += amount - taxAmount;  // WE ARE GETTING TAX WITH THIS LINE, THIS IS BUY TAX AND IT %5
_balances[address(this)] += taxAmount;
```

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	StarlinkSatire_BEP20.sol#L303,307,311
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.

```
tradeEnabled = true;
pairAddress = newPairAddress;
marketingWallet = newWallet;
```

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.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	StarlinkSatire_BEP20.sol#L299,300
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 declares certain variables that are not used in a meaningful way by the contract. As a result, these variables are redundant.

```
uint256 constant public buyTax = 5;
uint256 constant public sellTax = 5;
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	StarlinkSatire_BEP20.sol#L105,106,107,298
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 btax = 5
uint256 stax = 5
address public _owner
uint256 maxSupply = 1000000000000
```

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	StarlinkSatire_BEP20.sol#L107,296,299,300
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.

```
address public _owner

...
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	StarlinkSatire_BEP20.sol#L237
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);

        uint256 accountBalance = _balances[account];
...
            _totalSupply -= amount;
        }

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	StarlinkSatire_BEP20.sol#L195,203
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.

```
tradeEnabled == true || msg.sender == _owner
```

Recommendation

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

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	StarlinkSatire_BEP20.sol#L197,205
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.

```
_balances[to] += amount / 100 * 100 - stax
```

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	StarlinkSatire_BEP20.sol#L307,311
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.

```
pairAddress = newPairAddress
marketingWallet = newWallet
```

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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	StarlinkSatire_BEP20.sol#L15,28,69,87,96,294
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 ^0.8.17;
pragma solidity ^0.8.0;
```

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	StarlinkSatire_BEP20.sol#L15,28,69,87,96,294
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 ^0.8.17;
pragma solidity ^0.8.0;
```

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.

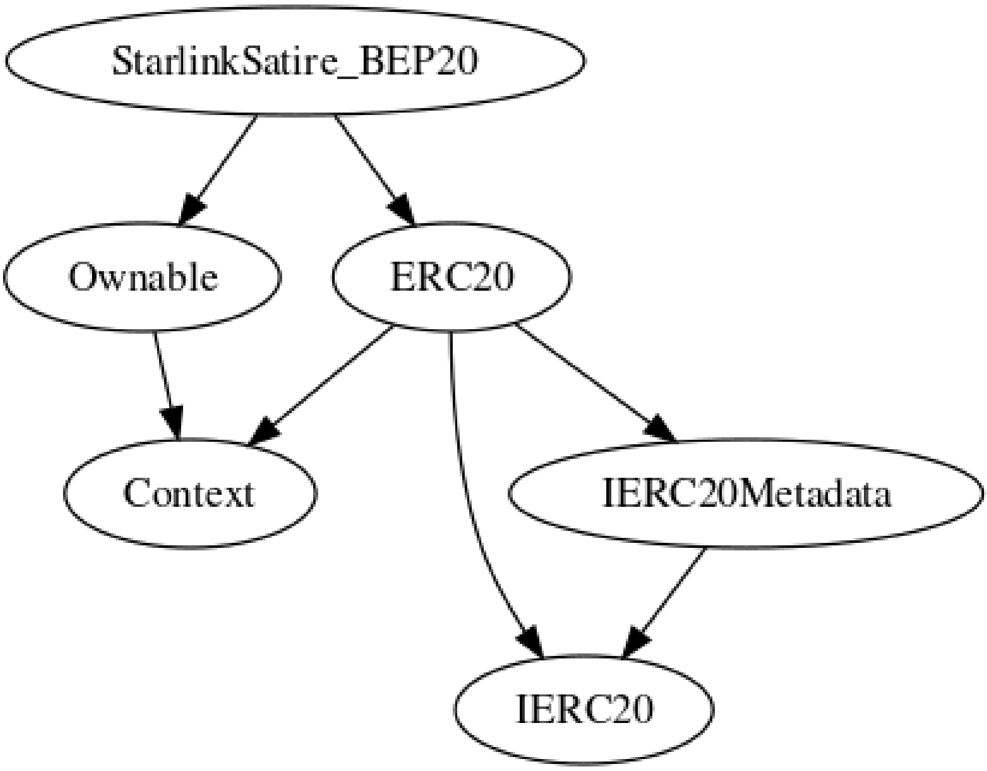
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-

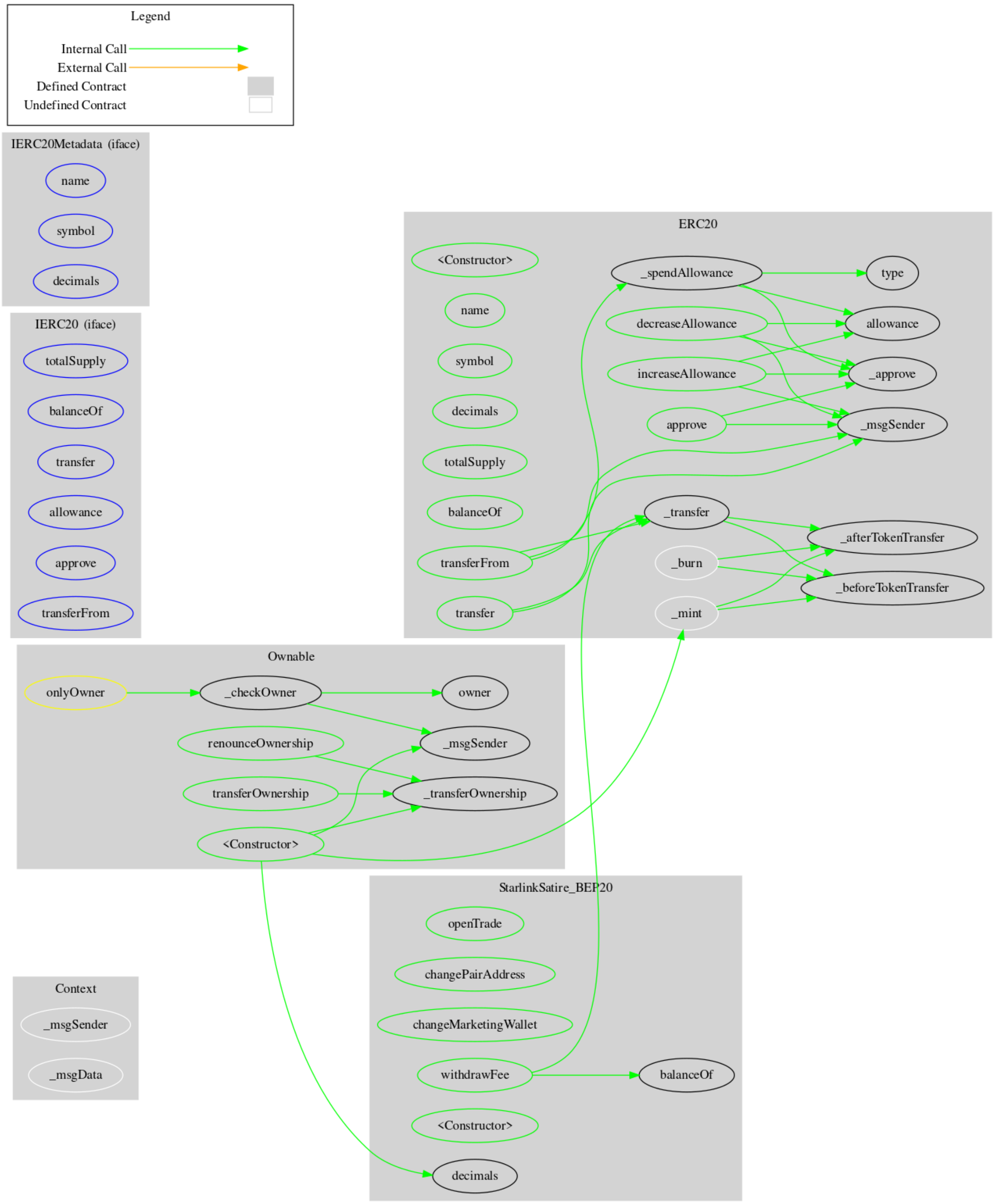
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	

	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
StarlinkSatire_BEP20	Implementation	ERC20, Ownable		
	openTrade	Public	✓	onlyOwner
	changePairAddress	Public	✓	onlyOwner
	changeMarketingWallet	Public	✓	onlyOwner
	withdrawFee	Public	✓	onlyOwner
		Public	✓	ERC20

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transferFrom
transfer
_spendAllowance
decreaseAllowance
increaseAllowance
approve
_transfer
_burn
_mint
type
allowance
_approve
_msgSender
_afterTokenTransfer
_beforeTokenTransfer
Ownable
onlyOwner
_checkOwner
owner
renounceOwnership
_msgSender
transferOwnership
_transferOwnership
<Constructor>
StarlinkSatire_BEP20
openTrade
changePairAddress
changeMarketingWallet
withdrawFee
balanceOf
<Constructor>
decimals
Context
_msgSender
_msgData

Summary

Starlink Satire 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. The contract implements a fee mechanism. During the audit process, it was identified that the contract's fee mechanism includes incorrect fee calculations, as described in detail at the TAM section.

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