Cyberscope

Audit Report

ChipToken

June 2023

Network	BSC Testnet
Address	0xcfda384cfa214508f4df0430f445286cf20b863a
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
●	RVD	Redundant Variable Declaration	Unresolved
●	RC	Redundant Calculations	Unresolved
●	MVN	Misleading Variables Naming	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	ChipToken
Compiler Version	v0.8.0+commit.c7dfd78e
Optimization	200 runs
Explorer	https://testnet.bscscan.com/address/0xcfda384cfa214508f4df0430f445286cf20b863a
Address	0xcfda384cfa214508f4df0430f445286cf20b863a
Network	BSC_TESTNET
Symbol	CHIPT
Decimals	18
Total Supply	1,000,000

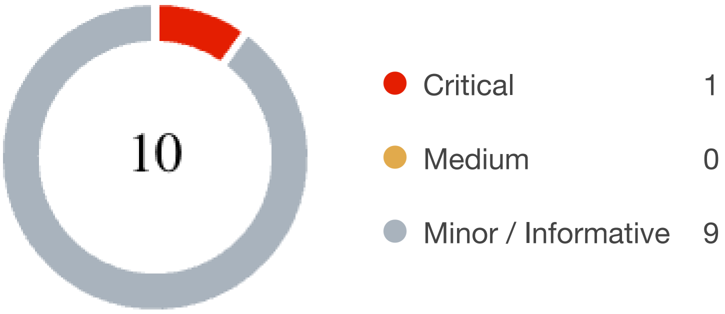
Audit Updates

Initial Audit	01 Jun 2023
Corrected Phase 2	03 Jun 2023

Source Files

Filename	SHA256
ChipToken.sol	360c1684ff576a984865763b80e9e579a290c108ed8dcf8c3f27d8d005122544

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	1	0	0	0
● Medium	0	0	0	0
● Minor / Informative	9	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	ChipToken.sol#L457
Status	Unresolved

Description

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage of it by setting either of the following wallets to the zero address:

- `teamWallet`
- `marketingWallet`
- `treasuryWallet`

```
super._transfer(sender, teamWallet, teamFeeAmount);
super._transfer(sender, marketingWallet, marketingFeeAmount);
super._transfer(sender, treasuryWallet, treasuryFeeAmount);
```

Recommendation

The contract could embody a check for not allowing setting the wallet addresses to the zero address.

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 time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	ChipToken.sol#L216
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 public buyFeeTeam = 25; // 0.25%
uint256 public buyFeeMarketing = 100; // 1%
uint256 public buyFeeLiquidity = 125; // 1.25%
uint256 public buyFeeBurn = 250; // 2.5%
uint256 public buyFeeTreasury = 300; // 3%
```

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.

RC - Redundant Calculations

Criticality	Minor / Informative
Location	ChipToken.sol#L353
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 calculates the same value more than once. The arithmetic operation `sellFeeTeam + sellFeeMarketing + sellFeeLiquidity + sellFeeTreasury` is the accumulation of the contract's fee percentages and is used as the denominator for each transfer. As a result, the contract performs redundant calculations.

```
uint256 teamFeeAmount = fee * sellFeeTeam / (sellFeeTeam +
sellFeeMarketing + sellFeeLiquidity + sellFeeTreasury);
uint256 marketingFeeAmount = fee * sellFeeMarketing / (sellFeeTeam
+ sellFeeMarketing + sellFeeLiquidity + sellFeeTreasury);
```

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.

MVN - Misleading Variables Naming

Criticality	Minor / Informative
Status	Unresolved

Description

Variables can have misleading names if their names do not accurately reflect the value they contain or the purpose they serve. The contract uses some variable names that are too generic or do not clearly convey the information stored in the variable. Misleading variable names can lead to confusion, making the code more difficult to read and understand.

- The variables `buyFeeLiquidity` , `sellFeeLiquidity` , `sellFeeBurn` , `buyFeeBurn` are not used by the contract
- The liquidity fee is transferred to the treasury wallet.
- The contract may tax only with `sellFeeTreasury` (or `sellFeeLiquidity` , `sellFeeMarketing`) fee and send the fees to the team, marketing and treasury wallet.

Recommendation

It's always a good practice for the contract to contain variable names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	ChipToken.sol#L210,218,219,220,221,222,230,231,232,233,234
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.

```
address public burnAddress =
0x000000000000000000000000000000000000dEaD
uint256 public MaxbuyFeeTeam = 100
uint256 public MaxbuyFeeMarketing = 100
uint256 public MaxbuyFeeLiquidity = 500
uint256 public MaxbuyFeeBurn = 500
uint256 public MaxbuyFeeTreasury = 500
uint256 public MaxsellFeeTeam = 100
uint256 public MaxsellFeeMarketing = 100
uint256 public MaxsellFeeLiquidity = 500
uint256 public MaxsellFeeBurn = 500
uint256 public MaxsellFeeTreasury = 500
```

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	ChipToken.sol#L10,218,219,220,221,222,230,231,232,233,234
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function WETH() external pure returns (address);
uint256 public MaxbuyFeeTeam = 100
uint256 public MaxbuyFeeMarketing = 100
uint256 public MaxbuyFeeLiquidity = 500
uint256 public MaxbuyFeeBurn = 500
uint256 public MaxbuyFeeTreasury = 500
uint256 public MaxsellFeeTeam = 100
uint256 public MaxsellFeeMarketing = 100
uint256 public MaxsellFeeLiquidity = 500
uint256 public MaxsellFeeBurn = 500
uint256 public MaxsellFeeTreasury = 500
```

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	ChipToken.sol#L145
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");
        require(_balances[account] >= amount, "ERC20: burn amount
exceeds balance");

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

        _balances[account] -= amount;
        _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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	ChipToken.sol#L330,349,350
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.

```
fee = amount * sellFeeTreasury / 10000
uint256 marketingFeeAmount = fee * sellFeeMarketing / (sellFeeTeam
+ sellFeeMarketing + sellFeeLiquidity + sellFeeTreasury)
```

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	ChipToken.sol#L249,254,259,264
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.

```
teamWallet = wallet
marketingWallet = wallet
treasuryWallet = wallet
liquidityPool = pool
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	ChipToken.sol#L2
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 ^0.8.0;
```

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.

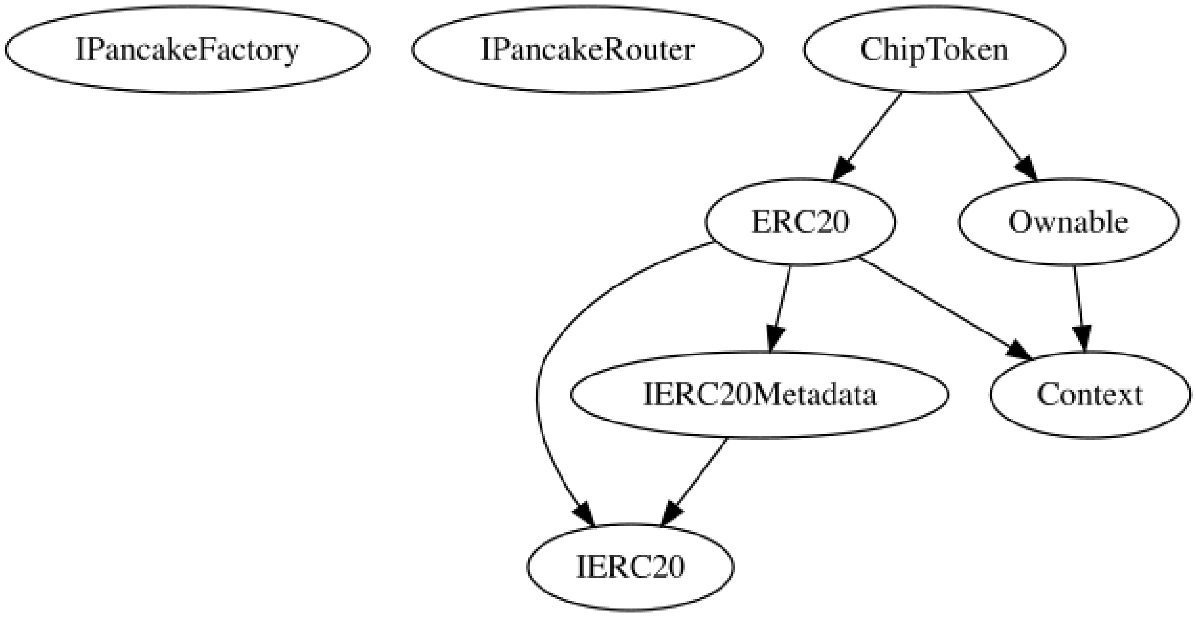
Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
IPancakeFactory	Interface			
	createPair	External	✓	-
	getPair	External		-
IPancakeRouter	Interface			
	WETH	External		-
	factory	External		-
	addLiquidity	External	✓	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-

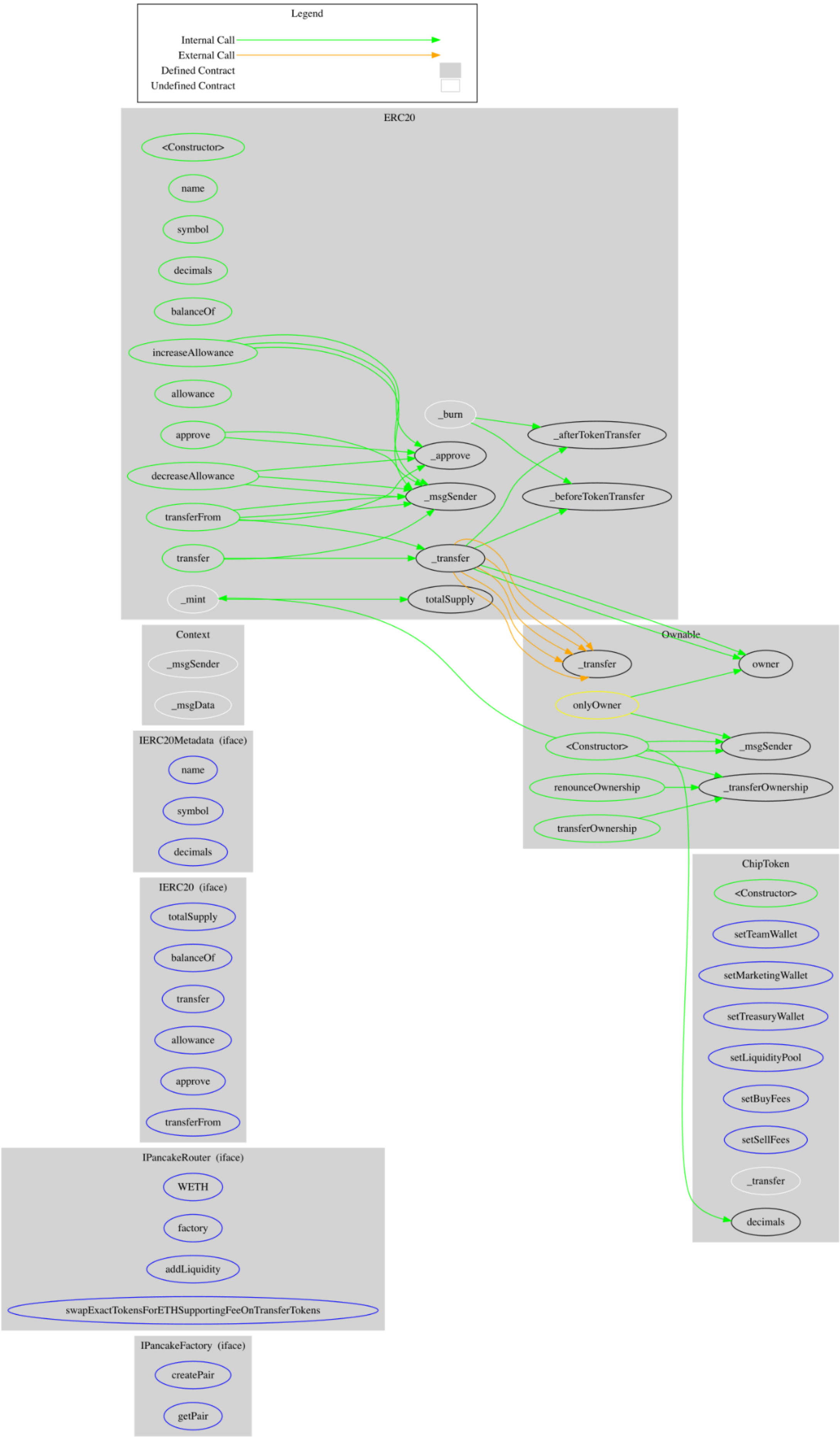
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public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

	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ChipToken	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	setTeamWallet	External	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setTreasuryWallet	External	✓	onlyOwner
	setLiquidityPool	External	✓	onlyOwner
	setBuyFees	External	✓	onlyOwner
	setSellFees	External	✓	onlyOwner
	_transfer	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC20
<Constructor>
name
symbol
decimals
balanceOf
increaseAllowance
allowance
_burn
_afterTokenTransfer
approve
_approve
decreaseAllowance
_msgSender
_beforeTokenTransfer
transferFrom
transfer
_transfer
_mint
totalSupply
Context
_msgSender
_msgData
Ownable
_transfer
owner
onlyOwner
<Constructor>
_msgSender
renounceOwnership
_transferOwnership
transferOwnership
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
ChipToken
<Constructor>
setTeamWallet
setMarketingWallet
setTreasuryWallet
setLiquidityPool
setBuyFees
setSellFees
_transfer
decimals
IPancakeRouter (iface)
WETH
factory
addLiquidity
swapExactTokensForETHSupportingFeeOnTransferTokens
IPancakeFactory (iface)
createPair
getPair

Summary

ChipToken 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. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 6% fee.

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