Cyberscope

Audit Report

Biggest Token

December 2024

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
⚪	FRV	Fee Restoration Vulnerability	Unresolved
⚪	MEE	Missing Events Emission	Unresolved
⚪	MSC	Missing Sanity Check	Unresolved
⚪	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
⚪	RRA	Redundant Repeated Approvals	Unresolved
⚪	RSML	Redundant SafeMath Library	Unresolved
⚪	L02	State Variables could be Declared Constant	Unresolved
⚪	L04	Conformance to Solidity Naming Conventions	Unresolved
⚪	L05	Unused State Variable	Unresolved
⚪	L07	Missing Events Arithmetic	Unresolved
⚪	L16	Validate Variable Setters	Unresolved
⚪	L17	Usage of Solidity Assembly	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	BiggestToken
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0xc623b0621835f008cefe5bfc9e000b651eccf32f
Address	0xc623b0621835f008cefe5bfc9e000b651eccf32f
Network	BSC
Symbol	BITO
Decimals	18
Total Supply	100,000,000,000
Badge Eligibility	Yes

Audit Updates

Initial Audit	17 Dec 2024

Source Files

Filename	SHA256
BiggestToken.sol	bf526f20bdedc4868a91eaf8c2d087c4454c0a05f0cfea86fa068737e5bb9abe

Findings Breakdown
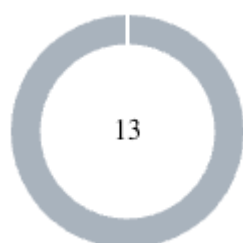
	Critical	0
	Medium	0
	Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	13	0	0	0

FRV - Fee Restoration Vulnerability

Criticality	Minor / Informative
Location	BiggestToken.sol#L540
Status	Unresolved

Description

The contract demonstrates a potential vulnerability upon removing and restoring the fees. This vulnerability can occur when the fees have been set to zero. During a transaction, if the fees have been set to zero, then both remove fees and restore fees functions will be executed. The remove fees function is executed to temporarily remove the fees, ensuring the sender is not taxed during the transfer. However, the function prematurely returns without setting the variables that hold the previous fee values.

As a result, when the subsequent restore fees function is called after the transfer, it restores the fees to their previous values. However, since the previous fee values were not properly set to zero, there is a risk that the fees will retain their non-zero values from before the fees were removed. This can lead to unintended consequences, potentially causing incorrect fee calculations or unexpected behavior within the contract.

```
function removeAllFee() private {
    if(_TotalFee == 0 && Bito_buy_fee == 0 && Bito_sell_fee == 0) return;

    _previousBuyFee = Bito_buy_fee;
    _previousSellFee = Bito_sell_fee;
    _previousTotalFee = _TotalFee;
    Bito_buy_fee = 0;
    Bito_sell_fee = 0;
    _TotalFee = 0;
}
```

Recommendation

The team is advised to modify the remove fees function to ensure that the previous fee values are correctly set to zero, regardless of their initial values. A recommended approach would be to remove the early return when both fees are zero.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	BiggestToken.sol#L491,495,513,527,660,665,669
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.

```
checknofee_transfer[account] = true;
checknofee_transfer[account] = false;
Admin_Wallet = wallet;
checkfeetransfer_ = true_or_false;
uniswapV2Router = _newPCSRouter;
```

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.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	BiggestToken.sol#L660,665,669
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The `New_Router_and_Update_Pair` , `New_Router_Address` , and `New_Pair_Address` methods modify the state of the `uniswapV2Router` and `uniswapV2Pair` variables. Without any validation, these addresses could be set any value, such as the zero or dead address. As a result, subsequent transactions involving swapping tokens would revert.

```
uniswapV2Router = _newPCSRouter;
uniswapV2Pair = newPair;
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	BiggestToken.sol#L640
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.

```solidity
function swapTokensForBNB(uint256 tokenAmount) private {
    address[] memory path = new address[](2);
    path[0] = address(this);
    path[1] = uniswapV2Router.WETH();
    _approve(address(this), address(uniswapV2Router), tokenAmount);
    uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
        tokenAmount,
        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.

RRA - Redundant Repeated Approvals

Criticality	Minor / Informative
Location	BiggestToken.sol#L645
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(
    tokenAmount,
    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	BiggestToken.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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	BiggestToken.sol#L376,377,378,379,380,385
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.

```solidity
string private _name = "Biggest Token"
string private _symbol = "BITO"
uint8 private _decimals = 18
uint256 private _tTotal = 100000000000 * 10**18
uint256 private _tFeeTotal
uint256 private maxPossibleFee = 20
```

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	BiggestToken.sol#L201,202,215,232,371,374,391,392,393,395,399,499,507,512,517,526,530,534,630,657,663,668
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);
mapping (address => bool) public checknofee_transfer
address payable public Admin_Wallet =
payable(0x8F3eD1d551F6b3d9838Ef1D7C7836365F2260ffA)
uint256 private _TotalFee = 20
uint256 public Bito_buy_fee = 10
uint256 public Bito_sell_fee = 10
uint256 public _maxBitoHold = _tTotal.mul(2).div(100)
uint256 public _maxTrx_bito = _tTotal.mul(2).div(100)


...
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	BiggestToken.sol#L380,396,400
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
uint256 private _tFeeTotal
uint256 private _previousMaxWalletToken = _maxBitoHold
uint256 private _previousMaxTxAmount = _maxTrx_bito
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	BiggestToken.sol#L502,518,531,535
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.

```
Bito_sell_fee = Bito_sell_update
swapTrigger = number_of_transactions
_maxBitoHold = _tTotal*maxWallPercent_x100/10000
_maxTrx_bito = _tTotal*maxTxPercent_x100/10000
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	BiggestToken.sol#L513,669
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.

```
Admin_Wallet = wallet
uniswapV2Pair = newPair
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	BiggestToken.sol#L80,135
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.

```
assembly { size := extcodesize(account) }

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	BiggestToken.sol#L15
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 ^0.8.7;
```

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**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	sub	Internal		
	div	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		

Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-

	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Pair	Interface			
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	External		-
	allowance	External		-
	approve	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	DOMAIN_SEPARATOR	External		-
	PERMIT_TYPEHASH	External		-
	nonces	External		-
	permit	External	✓	-
	MINIMUM_LIQUIDITY	External		-
	factory	External		-

	token0	External		-
	token1	External		-
	getReserves	External		-
	price0CumulativeLast	External		-
	price1CumulativeLast	External		-
	kLast	External		-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-

	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2 Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
BiggestToken	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-

decimals	Public		-
totalSupply	Public		-
balanceOf	Public		-
transfer	Public	✓	-
allowance	Public		-
approve	Public	✓	-
transferFrom	Public	✓	-
increaseAllowance	Public	✓	-
decreaseAllowance	Public	✓	-
excludeTaxLimit	Public	✓	onlyOwner
includeInTaxLimit	Public	✓	onlyOwner
_bito_fee_settings	External	✓	onlyOwner
set_Swap_And_Liquify_Enabled	Public	✓	onlyOwner
Admin_Update	Public	✓	onlyOwner
set_Number_Of_Transactions_Before_Liquify_Trigger	Public	✓	onlyOwner
	External	Payable	-
Transfers_tax_Update	External	✓	onlyOwner
_maxBitoHold_update	External	✓	onlyOwner
_maxTrx_bito_update	External	✓	onlyOwner
removeAllFee	Private	✓	
restoreAllFee	Private	✓	
_approve	Private	✓	
_transfer	Private	✓	

|				
sendToWallet	Private	✓	
swapAndLiquify	Private	✓	lockTheSwap
process_Transaction	Public	✓	onlyOwner
swapTokensForBNB	Private	✓	
New_Router_and_Update_Pair	Public	✓	onlyOwner
New_Router_Address	Public	✓	onlyOwner
New_Pair_Address	Public	✓	onlyOwner
_tokenTransfer	Private	✓	
_transferTokens	Private	✓	
_getValues	Private		

Inheritance Graph
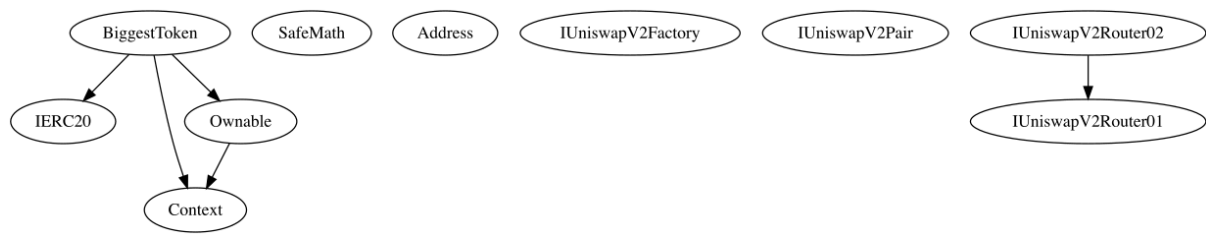
Flow Graph
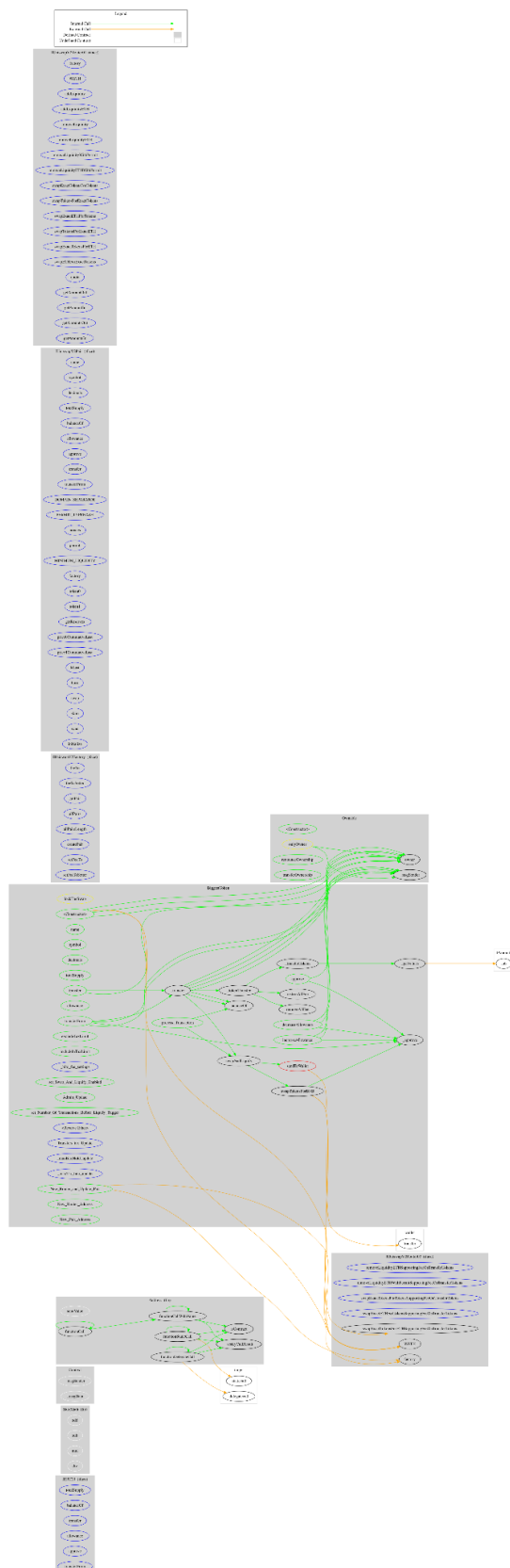
Summary

Biggest Token contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements.The contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link:

https://bscscan.com/tx/0xae11011f32fb39b85c410c4b68c988fee98616af52f9d0849560ec7b714f8dc6.

The fees are locked at 10% on both buy and sell transactions.

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