Cyberscope

Audit Report

Pepe Cyborg

June 2023

Network	BSC
Address	0x50EFe0DB00AD924800F2795bC14B2AE3D3937e77
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
●	CR	Code Repetition	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	MCM	Misleading Comment Messages	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	DKO	Delete Keyword Optimization	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	RVC	Redundant Variable Calculations	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved

●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis **1**
Diagnostics **2**
Table of Contents **4**
Review **6**
Audit Updates 6
Source Files 6
Findings Breakdown **7**
CR - Code Repetition 8
Description 8
Recommendation 9
PVC - Price Volatility Concern 10
Description 10
Recommendation 11
MCM - Misleading Comment Messages 12
Description 12
Recommendation 12
DDP - Decimal Division Precision 13
Description 13
Recommendation 14
DKO - Delete Keyword Optimization 15
Description 15
Recommendation 15
AOI - Arithmetic Operations Inconsistency 16
Description 16
Recommendation 16
RVC - Redundant Variable Calculations 17
Description 17
Recommendation 17
RSML - Redundant SafeMath Library 18
Description 18
Recommendation 18
IDI - Immutable Declaration Improvement 19
Description 19
Recommendation 19
L04 - Conformance to Solidity Naming Conventions 20
Description 20
Recommendation 21
L05 - Unused State Variable 22
Description 22

Recommendation 22
L07 - Missing Events Arithmetic 23
Description 23
Recommendation 23
L09 - Dead Code Elimination 24
Description 24
Recommendation 25
L13 - Divide before Multiply Operation 26
Description 26
Recommendation 26
L15 - Local Scope Variable Shadowing 27
Description 27
Recommendation 27
L16 - Validate Variable Setters 28
Description 28
Recommendation 28
L19 - Stable Compiler Version 29
Description 29
Recommendation 29
Functions Analysis 30
Inheritance Graph 37
Flow Graph 38
Summary 39
Disclaimer 40
About Cyberscope 41

Review

Contract Name	PepeCyborg
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x50efe0db00ad924800f2795bc14b2ae3d3937e77
Address	0x50efe0db00ad924800f2795bc14b2ae3d3937e77
Network	BSC
Symbol	PPBORG
Decimals	18
Total Supply	314,000,000,000,000

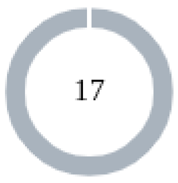
Audit Updates

Initial Audit	26 Jun 2023 https://github.com/cyberscope-io/audits/blob/main/ppborg/v1/audit.pdf
Corrected Phase 2	27 Jun 2023

Source Files

Filename	SHA256

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	17

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	17	0	0	0

CR - Code Repetition

Criticality	Minor / Informative
Location	PepeCyborg.sol#L952,1176,1211
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

Specifically, the constructor contains duplicate code for setting the `currentRouter` address.

```
constructor() ERC20("Pepe Cyborg", "PPBORG") {
...
    else if (block.chainid == 1 || block.chainid == 4) {
        currentRouter =
0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D; //Mainnet
    } else {
        currentRouter =
0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D; //Mainnet
            }
...
```

Morever the code segments that reset `tokensForLiquidity` and `tokensForMarketing` to zero are repeated in multiple places in the contract.

```
function resetTaxAmount() public onlyOwner {
    tokensForLiquidity = 0;
    tokensForMarketing = 0;
}
...
tokensForLiquidity = 0;
tokensForMarketing = 0;
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	PepeCyborg.sol#L1080
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The contract has a `swapEnabled` flag that can be toggled on and off from the owner using the `updateSwapEnabled` function. When `swapEnabled` is disabled, the contract starts to accumulate tokens in its balance. If `swapEnabled` is later enabled, the contract will swap all of its token balances at once meaning that a huge amount of tokens would be swapped.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
bool canSwap = contractTokenBalance > 0;

if(
    canSwap &&
    swapEnabled &&
    !swapping &&
    !automatedMarketMakerPairs[from] &&
    !_isExcludedFromFees[from] &&
    !_isExcludedFromFees[to]
    ) {
        swapping = true;

        swapBack();

        swapping = false;
    }
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. We recommend implementing a mechanism to limit the number of tokens that can be swapped at once. This could be a fixed limit or a percentage of the total token supply or the contract's token balance. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

MCM - Misleading Comment Messages

Criticality	Minor / Informative
Location	PepeCyborg.sol#L1105
Status	Unresolved

Description

The contract is using misleading comment messages. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code.

The comment states: "only take fees on buys/sells, do not take on wallet transfer" However, the contract has the potential to charge a transfer fee if the `transferLiquidityFee` and `transferMarketingFee` are set to a value greater than zero through the call of the `updateTransferFees` function.

As a result, the users will not comprehend the source code's actual implementation.

```
// only take fees on buys/sells, do not take on wallet
transfers
```

Recommendation

The team is advised to carefully review the comment in order to reflect the actual implementation. To improve code readability, the team should use more specific and descriptive comment messages.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	PepeCyborg.sol#L1111,1119,1125
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.

```
tokensForLiquidity += fees * sellLiquidityFee / sellTotalFees;
tokensForMarketing += fees * sellMarketingFee / sellTotalFees;
...
tokensForLiquidity += fees * buyLiquidityFee / buyTotalFees;
tokensForMarketing += fees * buyMarketingFee / buyTotalFees;
...
tokensForLiquidity += fees * transferLiquidityFee /
transferTotalFees;
tokensForMarketing += fees * transferMarketingFee /
transferTotalFees;
```

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.

DKO - Delete Keyword Optimization

Criticality	Minor / Informative
Location	PepeCyborg.sol#L1176
Status	Unresolved

Description

The contract resets variables to the default state by setting the initial values. Setting values to state variables increases the gas cost.

```
function resetTaxAmount() public onlyOwner {
    tokensForLiquidity = 0;
    tokensForMarketing = 0;
}
```

Recommendation

The team is advised to use the `delete` keyword instead of setting variables. This can be more efficient than setting the variable to a new value, using delete can reduce the gas cost associated with storing data on the blockchain.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	PepeCyborg.sol#L1200,1211
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
uint256 amountToSwapForETH =
contractBalance.sub(liquidityTokens);
...
uint256 ethForLiquidity = ethBalance - ethForMarketing;
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

RVC - Redundant Variable Calculations

Criticality	Minor / Informative
Location	PepeCyborg.sol#L1209
Status	Unresolved

Description

The contract contains redundant variable calculations. Specifically, the contract calculates the `ethForMarketing` and `ethForLiquidity` variables in a way that is unnecessarily complex and inefficient.

```
uint256 ethForMarketing =
ethBalance.mul(tokensForMarketing).div(totalTokensToSwap);
uint256 ethForLiquidity = ethBalance - ethForMarketing;
```

The contract first calculates `ethForMarketing` and then subtracts it from `ethBalance` to get `ethForLiquidity`. This involves two separate calculations and can consume more gas than necessary.

Recommendation

We recommend refactoring the code to eliminate the redundant calculation. The `ethForLiquidity` variable can be calculated directly from `ethBalance`, `tokensForLiquidity`, and `totalTokensToSwap`, without the need to first calculate `ethForMarketing`. This will make the code more efficient, gas efficient and also more readable and maintainable.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	PepeCyborg.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.

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

Recommendation

The team is advised to remove the SafeMath library. 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/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	PepeCyborg.sol#L918
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
_decimals
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	PepeCyborg.sol#L31,32,49,722,902,1016,1023,1030
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);
event marketingWalletUpdated(address indexed newWallet, address
indexed oldWallet);
uint256 _liquidityFee
uint256 _marketingFee
```

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.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	PepeCyborg.sol#L653
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.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

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	PepeCyborg.sol#L1017,1024,1031
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.

```
buyMarketingFee = _marketingFee
sellMarketingFee = _marketingFee
transferMarketingFee = _marketingFee
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	PepeCyborg.sol#L398,699,705,712
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _burn(address account, uint256 amount) internal
virtual {
        require(account != address(0), "ERC20: burn from the
zero address");

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

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

function abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

...
```

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	PepeCyborg.sol#L1110,1111,1112,1118,1119,1120,1125,1126,1127
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
fees = amount.mul(buyTotalFees).div(100)
tokensForMarketing += fees * transferMarketingFee /
transferTotalFees
```

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	PepeCyborg.sol#L916,920
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.

```
address _owner = 0xD4E1E7078F7d3562f95Ea23320f915Fae5Da93FA
uint256 totalSupply = 314000000000000 * (10**_decimals)
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	PepeCyborg.sol#L1056
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.

```
marketingWallet = newMarketingWallet
```

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	PepeCyborg.sol#L3
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 ^0.8.9;
```

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		
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		-
	mint	External	✓	-
	burn	External	✓	-
	swap	External	✓	-
	skim	External	✓	-
	sync	External	✓	-
	initialize	External	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-

	setFeeToSetter	External	✓	-
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		-

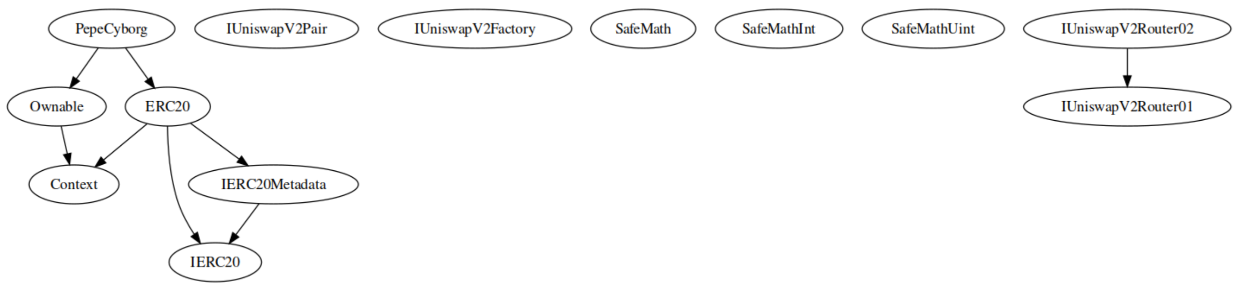
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		

		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-

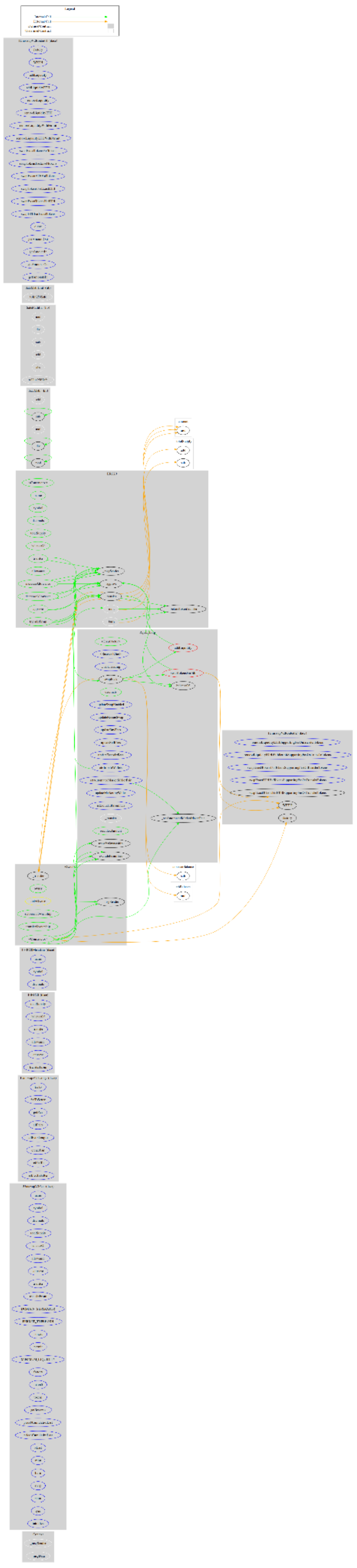
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
PepeCyborg	Implementation	ERC20, Ownable		

		Public	✓	ERC20
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	airdropToWallets	External	✓	onlyOwner
	decimals	Public		-
	updateSwapEnabled	External	✓	onlyOwner
	updateRescueSwap	External	✓	onlyOwner
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	updateTransferFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateMarketingWallet	External	✓	onlyOwner
	isExcludedFromFees	External		-
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	resetTaxAmount	Public	✓	onlyOwner
	swapBack	Private	✓	

Inheritance Graph



Flow Graph



Summary

Pepe Cyborg contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Pepe Cyborg is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. There is also a limit of max 20% fees.

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