Cyberscope

Audit Report

$PEPESMOKING

September 2023

Network	BSC
Address	0xb64e5adbe17c1ff6357cfbf88d97ef193531d2b9
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	Unresolved

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	RSD	Redundant Swap Duplication	Unresolved
	FRV	Fee Restoration Vulnerability	Unresolved
	PTRP	Potential Transfer Revert Propagation	Unresolved
	PVC	Price Volatility Concern	Unresolved
	FSA	Fixed Swap Address	Unresolved
	AOI	Arithmetic Operations Inconsistency	Unresolved
	MEE	Missing Events Emission	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	RED	Redudant Event Declaration	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved

●	L09	Dead Code Elimination	Unresolved
●	L11	Unnecessary Boolean equality	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L19	Stable Compiler Version	Unresolved

Table of Contents

Review

Contract Name	PEPESMOKING
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://bscscan.com/address/0xb64e5adbe17c1ff6357cfbf88d97ef193531d2b9
Address	0xb64e5adbe17c1ff6357cfbf88d97ef193531d2b9
Network	BSC
Symbol	PES
Decimals	18
Total Supply	210,000,000,000

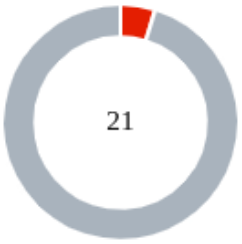
Audit Updates

Initial Audit	13 Sep 2023

Source Files

Filename	SHA256
PEPESMOKING.sol	a459b838f81a5b572664cfcffdc639895ba88b7b76664df53dd97e1b88ebe854

Findings Breakdown



Critical	1
Medium	0
Minor / Informative	20

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	0	0	0	0
Minor / Informative	20	0	0	0

BC - Blacklists Addresses

Criticality	Critical
Location	PEPESMOKING.sol#L1184
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `blackListAccount` function.

```
   function blackListAccount(address account) external
onlyOwner {
        isBlackListed[account] = true;
    }
```

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 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.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	PEPESMOKING.sol#L1003
Status	Unresolved

Description

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

```
function swapAndLiquify(uint256 contractTokenBalance) private
lockTheSwap {
        uint256 liquidityTokens =
contractTokenBalance.mul(_liquidityFee).div(_liquidityFee.add(_marketing
Fee));

        uint256 half = liquidityTokens.div(2);
        uint256 otherHalf = liquidityTokens.sub(half);
        ...
        swapTokensForEth(half);
        ...
        swapTokensForEth(contractTokenBalance.sub(liquidityTokens));
        ...
    }
```

Recommendation

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

FRV - Fee Restoration Vulnerability

Criticality	Minor / Informative
Location	PEPESMOKING.sol#L930,945,1151
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(_taxFee == 0 && _liquidityFee == 0 &&
_marketingFee==0 && _burnFee==0) return;

        _previousTaxFee = _taxFee;
        _previousLiquidityFee = _liquidityFee;
        _previousBurnFee = _burnFee;
        _previousMarketingFee = _marketingFee;

        _taxFee = 0;
        _liquidityFee = 0;
        _marketingFee = 0;
        _burnFee = 0;
    }

    function restoreAllFee() private {
        _taxFee = _previousTaxFee;
        _liquidityFee = _previousLiquidityFee;
        _burnFee = _previousBurnFee;
        _marketingFee = _previousMarketingFee;
    }

  function _tokenTransfer(address sender, address recipient,
uint256 amount) private
    {
        if(_isExcludedFromFee[sender] ||
_isExcludedFromFee[recipient])
        {
           removeAllFee();
        }
        ...
            _transferStandard(sender, recipient, amount);
        }

        if(_isExcludedFromFee[sender] ||
_isExcludedFromFee[recipient])
        {
            restoreAllFee();
        }
    }
```

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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	PEPESMOKING.sol#L1012
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
marketingWallet.transfer(address(this).balance);
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	PEPESMOKING.sol#L973,1165
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `numTokensSellToAddToLiquidity` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

Additionally. the contract can accumulate a large number of tokens if certain conditions are not met, such as `swapAndLiquifyEnabled` being set to `false` or `_marketingFee` or `_liquidityFee` being set to zero. When these conditions are eventually met, the contract could then swap a huge amount of tokens for ETH, which might not be the desired behavior.

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.

```
function _transfer(
        address from,
        address to,
        uint256 amount
    ) private {
        ...
        uint256 contractTokenBalance =
balanceOf(address(this));
        bool overMinTokenBalance = contractTokenBalance >=
numTokensSellToAddToLiquidity;
        if (
            overMinTokenBalance &&
            !inSwapAndLiquify &&
            to == uniswapV2Pair &&
            swapAndLiquifyEnabled &&
            _marketingFee > 0 &&
            _liquidityFee > 0
        ) {
            swapAndLiquify(contractTokenBalance);
        }
    ...
    }

    function setNumTokensSellToAddToLiquidity(uint256
newAmount) external onlyOwner() {
        numTokensSellToAddToLiquidity = newAmount;
    }
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	PEPESMOKING.sol#L729
Status	Unresolved

Description

The swap address is assigned once and it can not be changed. It is a common practice in decentralized exchanges to create new swap versions. A contract that cannot change the swap address may not be able to catch up to the upgrade. As a result, the contract will not be able to migrate to a new liquidity pool pair or decentralized exchange.

```
   constructor() {
        ...
        IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);

        uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(address(this),
_uniswapV2Router.WETH());

        uniswapV2Router = _uniswapV2Router;
        ...
    }
```

Recommendation

The team is advised to add the ability to change the pair and router address in order to cover potential liquidity pool migrations. It would be better to support multiple pair addresses so the token will be able to have the same behavior in all the decentralized liquidity pairs.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	PEPESMOKING.sol#L1062,1090
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.

```
require(balanceOf(recipient) + amount <= _maxWalletLimit,
"Exceeds the maxWalletSize.");
...
_rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	PEPESMOKING.sol#L832,843,1157,1184,1188
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.

```
function excludeFromReward(address account) public onlyOwner()
{
        require(!_isExcluded[account], "Account is already
excluded");
        require(account != deadAddress, "cannot exclude");
        require(account != address(this), "cannot exclude");
        if(_rOwned[account] > 0) {
            _tOwned[account] =
tokenFromReflection(_rOwned[account]);
        }
        _isExcluded[account] = true;
        _excluded.push(account);
    }

    function includeInReward(address account) external
onlyOwner() {
        ...
    }

    function setFeePercent(uint256 taxFee, uint256
liquidityFee, uint256 marketingFee, uint256 burnFee) external
onlyOwner() {

require(taxFee.add(liquidityFee).add(marketingFee).add(burnFee)
<= 10, "tax too high");
        _taxFee = taxFee;
        _liquidityFee = liquidityFee;
        _marketingFee = marketingFee;
        _burnFee = burnFee;
    }

    function blackListAccount(address account) external
onlyOwner {
        isBlackListed[account] = true;
    }

    function unBlackListAccount(address account) external
onlyOwner  {
        isBlackListed[account] = false;
    }
```

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	PEPESMOKING.sol#L1165,1169,1174,1179,1184,1188
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 updates status of variables even if their current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
    function setNumTokensSellToAddToLiquidity(uint256 newAmount)
external onlyOwner() {
        numTokensSellToAddToLiquidity = newAmount;
    }

    function setMaxTxAmount(uint256 maxTxAmount) external
onlyOwner() {
        _maxTxAmount = maxTxAmount;
        require(_maxTxAmount > totalSupply().div(200), "value
too low");
    }

    function setMaxWalletLimit(uint256 amount) external
onlyOwner() {
        require(amount > totalSupply().div(200), "value too
low");
        _maxWalletLimit = amount;
    }

    function setSwapAndLiquifyEnabled(bool _enabled) public
onlyOwner {
        swapAndLiquifyEnabled = _enabled;
        emit SwapAndLiquifyEnabledUpdated(_enabled);
    }

    function blackListAccount(address account) external
onlyOwner {
        isBlackListed[account] = true;
    }

    function unBlackListAccount(address account) external
onlyOwner  {
        isBlackListed[account] = false;
    }
```

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.

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	PEPESMOKING.sol#L713
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 event `MinTokensBeforeSwapUpdated` is declared and not being used in the contract. As a result, it is redundant.

```
event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	PEPESMOKING.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	PEPESMOKING.sol#L729,733
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.

```
uniswapV2Pair
uniswapV2Router
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	PEPESMOKING.sol#L680,684,685,686,696
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 private _tTotal = 210000000000 * (10**18)
string private _name = "$PEPESMOKING"
string private _symbol = "PES"
uint8 private _decimals = 18
address public deadAddress =
0x000000000000000000000000000000000000dEaD
```

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	PEPESMOKING.sol#L488,489,505,526,688,691,694,698,706,707,914,920,1175
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);
uint256 public _taxFee = 1
uint256 public _liquidityFee = 4
uint256 public _burnFee = 1
uint256 public _marketingFee = 2
uint256 public _maxTxAmount = 210000000000 * 10**18
uint256 public _maxWalletLimit = 210000000000 * 10**18
uint256 _amount
bool _enabled
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	PEPESMOKING.sol#L1155,1162,1166,1172
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.

```
_taxFee = taxFee
numTokensSellToAddToLiquidity = newAmount
_maxTxAmount = maxTxAmount
_maxWalletLimit = amount
```

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	PEPESMOKING.sol#L335,362,388,398,413,423,428
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 isContract(address account) internal view returns
(bool) {
        // According to EIP-1052, 0x0 is the value returned for
not-yet created accounts
        // and
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a
470 is returned
        // for accounts without code, i.e. `keccak256('')`
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a
470;
        // solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

...
```

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	PEPESMOKING.sol#L962
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.

```
require(isBlackListed[from] != true && isBlackListed[to] !=
true, "Account is Blacklisted")
```

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	PEPESMOKING.sol#L1096,1108
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.

```
uint256 tBurn = tAmount.div(100).mul(_burnFee)
```

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	PEPESMOKING.sol#L1150
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 = 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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	PEPESMOKING.sol#L342,441
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 { codehash := extcodehash(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	PEPESMOKING.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**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner

	_transferOwnership	Internal	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	
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	✓	-

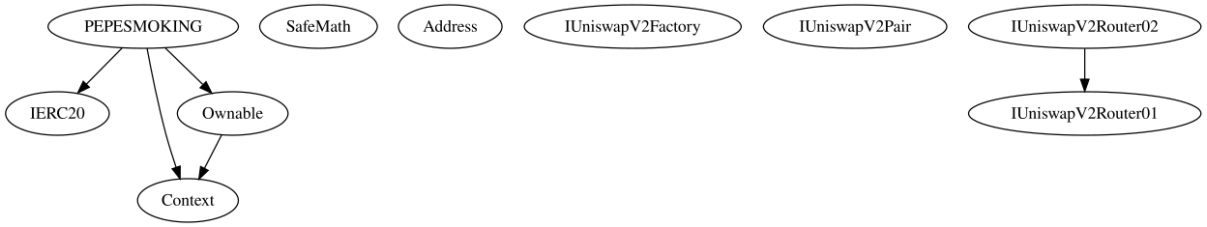
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2 Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
PEPESMOKING	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	✓	-
	isExcludedFromReward	Public		-
	totalFees	Public		-
	deliver	Public	✓	-
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	_transferBothExcluded	Private	✓	
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		

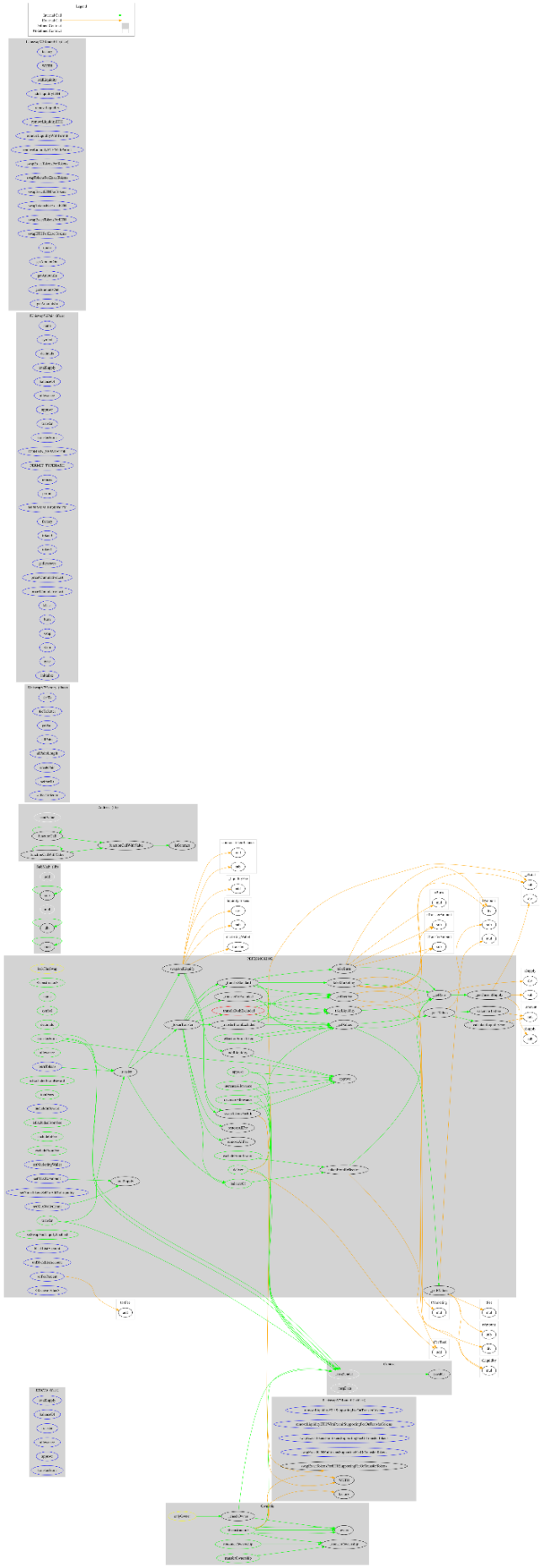
	_getCurrentSupply	Private		
	_takeLiquidity	Private	✓	
	calculateLiquidityFee	Private		
	calculateTaxFee	Private		
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	_approve	Private	✓	
	_transfer	Private	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	takeBurn	Private	✓	
	takeMarketing	Private	✓	
	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	
	isExcludedFromFee	Public		-
	includeInFee	Public	✓	onlyOwner
	excludeFromFee	Public	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	setFeePercent	External	✓	onlyOwner
	setNumTokensSellToAddToLiquidity	External	✓	onlyOwner

	setMaxTxAmount	External	✓	onlyOwner
	setMaxWalletLimit	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
	blackListAccount	External	✓	onlyOwner
	unBlackListAccount	External	✓	onlyOwner
	burnTokens	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph



Summary

$PEPESMOKING 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 massively blacklist addresses. 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 10% 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