Cyberscope

Audit Report

PepeSuperGrow

May 2023

Network BSC Testnet

Address 0x4b566937E8A32568FA4177528c1a8a8048FE128C

Audited by © cyberscope

Table of Contents

Review

Contract Name	PepeSuperGrow
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://testnet.bscscan.com/address/0x4b566937e8a32568fa4177528c1a8a8048fe128c
Address	0x4b566937e8a32568fa4177528c1a8a8048fe128c
Network	BSC_TESTNET
Symbol	PEPEG
Decimals	18
Total Supply	10.000.000.000

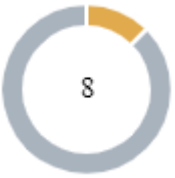
Audit Updates

Initial Audit	08 May 2023

Source Files

Filename	SHA256
PepeSuperGrow.sol	1f553734c9be76a314bb65d8b40dc35162b11f92296195c2c356e6aac7d19b8c

Findings Breakdown



Critical	0
Medium	1
Minor / Informative	7

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	1	0	0	0
Minor / Informative	7	0	0	0

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

ST - Stops Transactions

Criticality	Medium
Location	PepeSuperGrow.sol#L303
Status	Unresolved

Description

The contract owner has the authority to perform transactions when the trading is not open. The owner may take advantage of it by performing transactions when the trading is closed.

```
if (isLimitedAddress(from,to)) {
  require(isTradingEnabled,"Trading is not enabled");
}
```

Recommendation

The contract should not allow transactions when the trading is close. The team should carefully manage the private keys of the owner's account. 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.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	VO	Variables Optimization	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

VO - Variables Optimization

Criticality	Minor / Informative
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.

Since the variables `maxBuyFee` and `maxSellFee` share the same information and are immutable, it is redundant to include both.

```
uint256 private maxSellFee = 400;
uint256 private maxBuyFee = 400;
```

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.

- It is recommended to remove one of these variables from the contract.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	PepeSuperGrow.sol#L205
Status	Unresolved

Description

The contract is using variables that initialize them only in the constructor. The other functions are not mutating the variables. These variables are not defined as `immutable`.

```
swapRouter
```

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	PepeSuperGrow.sol#L170,171,181,192
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 maxSellFee = 400
uint256 private maxBuyFee = 400
bool private canSwapFees = true
bool public LiquidityAdded = false
```

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	PepeSuperGrow.sol#L69,186,187,188,192,267,272,277,393,399
Status	Unresolved

Description

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

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

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

```
function WETH() external pure returns (address);
string constant private _name = "Pepe SuperGrow"
string constant private _symbol = "PEPEG"
uint8 constant private _decimals = 18
bool public LiquidityAdded = false

function is_buy(address ins, address out) internal view returns
(bool) {
        bool _is_buy = !isLpPair[out] && isLpPair[ins];
        return _is_buy;
    }

function is_sell(address ins, address out) internal view
returns (bool) {
        bool _is_sell = isLpPair[out] && !isLpPair[ins];
        return _is_sell;
    }

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	PepeSuperGrow.sol#L277
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 is_transfer(address ins, address out) internal view
returns (bool) {
        bool _is_transfer = !isLpPair[out] && !isLpPair[ins];
        return _is_transfer;
    }
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	PepeSuperGrow.sol#L289,332,333
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.

```
lpPair = newPair
marketingAddress = payable(marketing)
rewardsAddress = payable(rewards)
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	PepeSuperGrow.sol#L424
Status	Unresolved

Description

According to the ERC20 specification, the transfer methods should be checked if the result is successful. Otherwise, the contract may wrongly assume that the transfer has been established.

```
IERC20(tokenAdd).transfer(owner(), amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

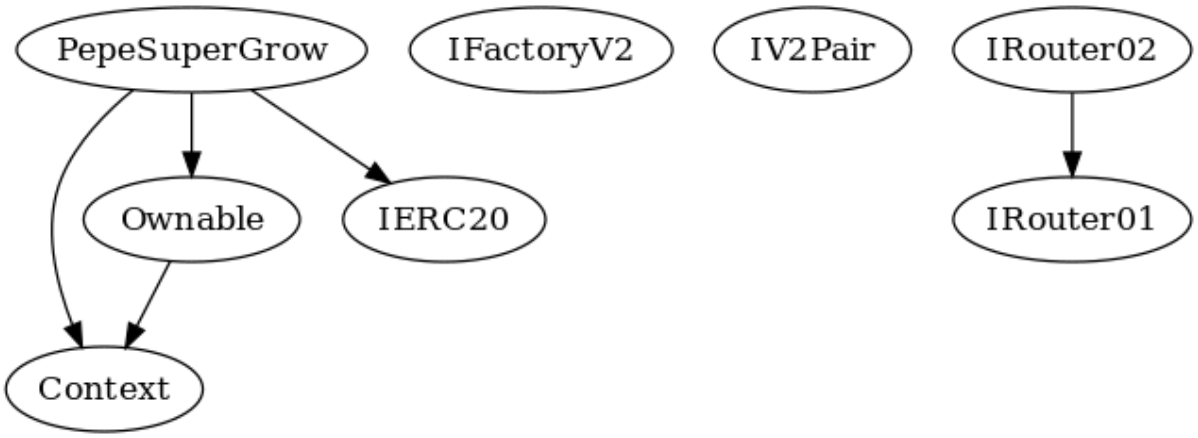
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
		Public	✓	-
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_setOwner	Private	✓	
IFactoryV2	Interface			
	getPair	External		-
	createPair	External	✓	-
IV2Pair	Interface			
	factory	External		-

	getReserves	External		-
	sync	External	✓	-
IRouter01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	addLiquidity	External	✓	-
	swapExactETHForTokens	External	Payable	-
	getAmountsOut	External		-
	getAmountsIn	External		-
IRouter02	Interface	IRouter01		
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokens	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-

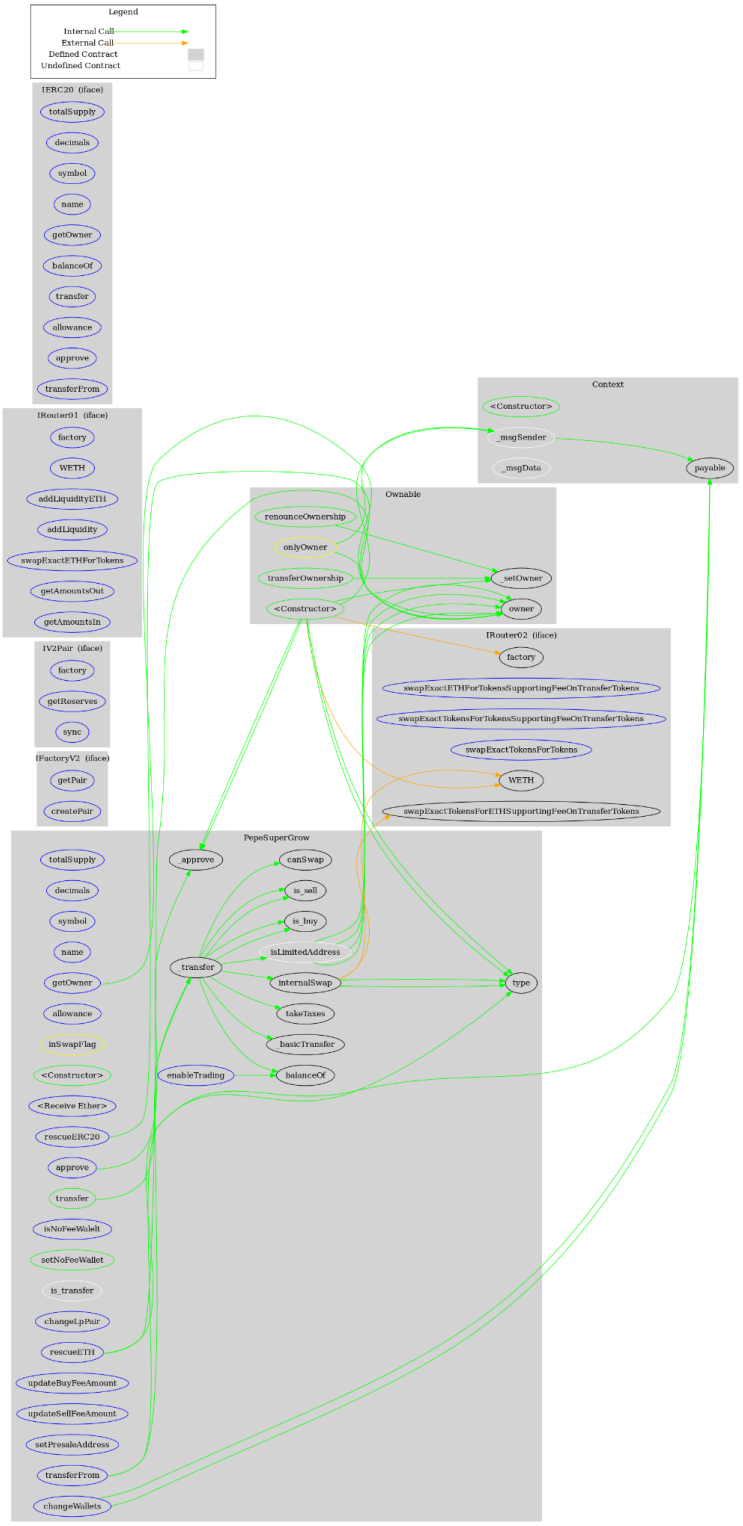
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
PepeSuperGrow	Implementation	Context, Ownable, IERC20		
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	allowance	External		-
	balanceOf	Public		-
		Public	✓	-
		External	Payable	-
	transfer	Public	✓	-
	approve	External	✓	-
	_approve	Internal	✓	
	transferFrom	External	✓	-
	isNoFeeWalelt	External		-
	setNoFeeWallet	Public	✓	onlyOwner

	isLimitedAddress	Internal		
	is_buy	Internal		
	is_sell	Internal		
	is_transfer	Internal		
	canSwap	Internal		
	changeLpPair	External	✓	onlyOwner
	_transfer	Internal	✓	
	_basicTransfer	Internal	✓	
	changeWallets	External	Payable	onlyOwner
	takeTaxes	Internal	✓	
	internalSwap	Internal	✓	inSwapFlag
	updateBuyFeeAmount	External	✓	onlyOwner
	updateSellFeeAmount	External	✓	onlyOwner
	setPresaleAddress	External	✓	onlyOwner
	enableTrading	External	✓	onlyOwner
	rescueETH	External	✓	onlyOwner
	rescueERC20	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom
IRouter01 (iface)
factory
WETH
addLiquidityETH
addLiquidity
swapExactETHForTokens
getAmountsOut
getAmountsIn
IV2Pair (iface)
factory
getReserves
sync
IFactoryV2 (iface)
getPair
createPair
Context
<Constructor>
_msgSender
_msgData
payable
Ownable
renounceOwnership
onlyOwner
transferOwnership
<Constructor>
setOwner
owner
IRouter02 (iface)
factory
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokens
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
PepeSuperGrow
totalSupply
decimals
symbol
name
getOwner
allowance
inSwapFlag
<Constructor>
<Receive Ether>
rescueERC20
approve
transfer
isNoFeeWalelt
setNoFeeWallet
is_transfer
changeLpPair
rescueETH
updateBuyFeeAmount
updateSellFeeAmount
setPresaleAddress
transferFrom
changeWallets
approve
transfor
enableTrading
canSwap
is_sell
is_buy
isLimitedAddress
internalSwap
takeTaxes
basicTransfer
balanceOf
type

Summary

Pepe Super Grow contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like stopping transactions. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. 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 4% fee.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes nor may copies be delivered to any other person other than the Company without Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives false negatives and other unpredictable results. The services may access and depend upon multiple layers of third parties.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io