Cyberscope

Audit Report

The Pepefather

May 2023

Network	BSC
Address	0x8539600C9AEC29d2ab196F9388e59ea250f7B0F9
Audited by	© cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Source Files 3
Findings Breakdown 4
Analysis 5
Diagnostics 6
L02 - State Variables could be Declared Constant 7
Description 7
Recommendation 7
L04 - Conformance to Solidity Naming Conventions 8
Description 8
Recommendation 8
L05 - Unused State Variable 10
Description 10
Recommendation 10
L14 - Uninitialized Variables in Local Scope 11
Description 11
Recommendation 11
L16 - Validate Variable Setters 12
Description 12
Recommendation 12
L20 - Succeeded Transfer Check 13
Description 13
Recommendation 13
Functions Analysis 14
Inheritance Graph 18
Flow Graph 19
Summary 20
Disclaimer 21
About Cyberscope 22

Review

Contract Name	ThePepeFather
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	500 runs
Explorer	https://bscscan.com/address/0x8539600c9aec29d2ab196f9388e59ea250f7b0f9
Address	0x8539600c9aec29d2ab196f9388e59ea250f7b0f9
Network	BSC
Symbol	PEPEFATHER
Decimals	18
Total Supply	1,000,000,000,000

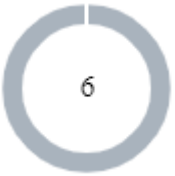
Audit Updates

Initial Audit	08 May 2023 https://github.com/cyberscope-io/audits/blob/main/pepefather/v1/audit.pdf
Corrected Phase 2	23 May 2023 https://github.com/cyberscope-io/audits/blob/main/pepefather/v2/audit.pdf
Corrected Phase 3	24 May 2023

Source Files

Filename	SHA256
ThePepeFather.sol	9b36c774fb2b4271e073e745ffa42091281780255e2229df95976feee1ed9cbf

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	6	0	0	0

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
	ST	Stops Transactions	Passed
	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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	ThePepeFather.sol#L100,112
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 timeSinceLastPair = 0
bool public taxesAreLocked
```

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	ThePepeFather.sol#L33,106,107,108,109,110,118,230
Status	Unresolved

Description

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

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

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

```
function WETH() external pure returns (address);
uint256 constant private startingSupply = 999_999_999_999
string constant private _name = "The PepeFather"
string constant private _symbol = "PEPEFATHER"
uint8 constant private _decimals = 18
uint256 constant private _tTotal = startingSupply * 10**_decimals
bool public _hasLiqBeenAdded = false
address _initializer
```

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	ThePepeFather.sol#L100
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 timeSinceLastPair = 0
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	ThePepeFather.sol#L234
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
address constructorLP
address router
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	ThePepeFather.sol#L175
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.

```
operator = newOperator
```

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	ThePepeFather.sol#L336
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.

```
TOKEN.transfer(_owner, TOKEN.balanceOf(address(this)))
```

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

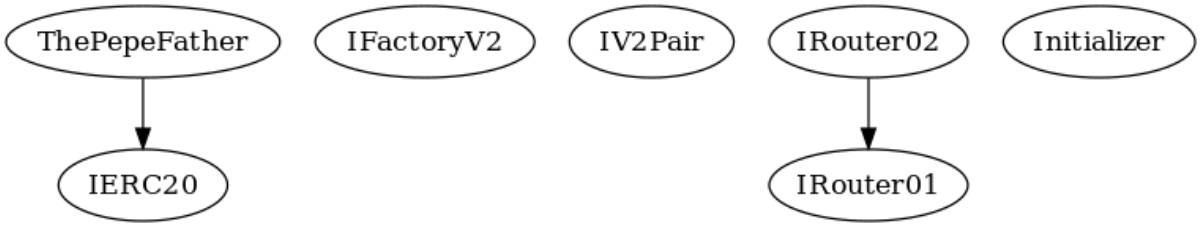
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
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	✓	-
Initializer	Interface			
	setLaunch	External	✓	-
	getConfig	External	✓	-
	setLpPair	External	✓	-

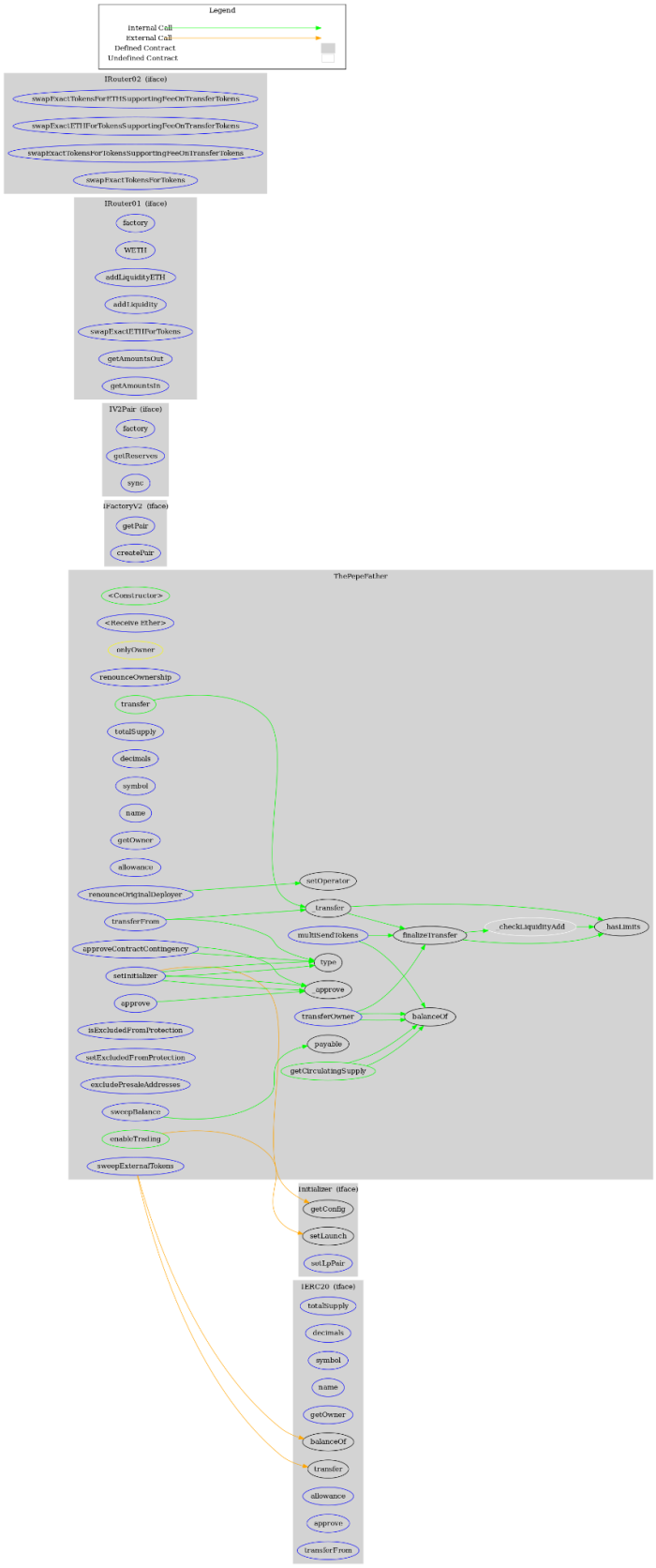
ThePepeFather	Implementation	IERC20		
		Public	Payable	-
		External	Payable	-
	transferOwner	External	✓	onlyOwner
	renounceOwnership	External	✓	onlyOwner
	setOperator	Public	✓	-
	renounceOriginalDeployer	External	✓	-
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	allowance	External		-
	balanceOf	Public		-
	transfer	Public	✓	-
	approve	External	✓	-
	_approve	Internal	✓	
	approveContractContingency	External	✓	onlyOwner
	transferFrom	External	✓	-
	setInitializer	External	✓	onlyOwner
	isExcludedFromProtection	External		-
	setExcludedFromProtection	External	✓	onlyOwner
	getCirculatingSupply	Public		-

	excludePresaleAddresses	External	✓	onlyOwner
	_hasLimits	Internal		
	_transfer	Internal	✓	
	_checkLiquidityAdd	Internal	✓	
	enableTrading	Public	✓	onlyOwner
	sweepBalance	External	✓	onlyOwner
	sweepExternalTokens	External	✓	onlyOwner
	multiSendTokens	External	✓	onlyOwner
	finalizeTransfer	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IRouter02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokens
IRouter01 (iface)
factory
WETH
addLiquidityETH
addLiquidity
swapExactETHForTokens
getAmountsOut
getAmountsIn
IV2Pair (iface)
factory
getReserves
sync
IFactoryV2 (iface)
getPair
createPair
ThePepeFather
<Constructor>
<Receive Ether>
onlyOwner
renounceOwnership
transfer
totalSupply
decimals
symbol
name
getOwner
allowance
renounceOriginalDeployer
transferFrom
approveContractContingency
setInitializer
approve
isExcludedFromProtection
setExcludedFromProtection
excludePresaleAddresses
sweepBalance
enableTrading
sweepExternalTokens
setOperator
transfer
multiSendTokens
type
approve
transferOwner
payable
getCirculatingSupply
finalizeTransfer
checkLiquidityAdd
hasLimits
balanceOf
initializer (iface)
getConfig
setLaunch
setLpPair
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom

Summary

The Pepefather contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. The Pepefather is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors 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.

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

Cyberscope

Audit Report

PepeZilla

May 2023

Network BSC

Address 0xE7d00b2Cc01604D84B8f9E04C0516C00c7e8c042

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Source Files 2
Findings Breakdown 3
Analysis 4
Diagnostics 5
IDI - Immutable Declaration Improvement 6
Description 6
Recommendation 6
L02 - State Variables could be Declared Constant 7
Description 7
Recommendation 7
L04 - Conformance to Solidity Naming Conventions 8
Description 8
Recommendation 8
L05 - Unused State Variable 10
Description 10
Recommendation 10
L14 - Uninitialized Variables in Local Scope 11
Description 11
Recommendation 11
L16 - Validate Variable Setters 12
Description 12
Recommendation 12
Functions Analysis 13
Inheritance Graph 17
Flow Graph 18
Summary 19
Disclaimer 20
About Cyberscope 21

Review

Contract Name	PepeZilla
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	500 runs
Explorer	https://bscscan.com/address/0xe7d00b2cc01604d84b8f9e04c0516c00c7e8c042
Address	0xe7d00b2cc01604d84b8f9e04c0516c00c7e8c042
Network	BSC
Symbol	PEPEZILLA
Decimals	18
Total Supply	69.420.000.000

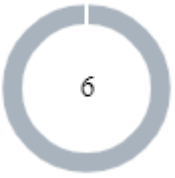
Audit Updates

Initial Audit	08 May 2023

Source Files

Filename	SHA256
PepeZilla.sol	8b26631d105251cdb08582a0b7372139b6b61b785fa4a8c525f4d5d345783970

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	6	0	0	0

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	IDI	Immutable Declaration Improvement	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	PepeZilla.sol#L131,145
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`.

```
dexRouter
lpPair
```

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	PepeZilla.sol#L102,113
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 timeSinceLastPair = 0
bool public taxesAreLocked
```

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	PepeZilla.sol#L33,107,108,109,110,111,119,271
Status	Unresolved

Description

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

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

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

```
function WETH() external pure returns (address);
uint256 constant private startingSupply = 69_420_000_000
string constant private _name = "PepeZilla"
string constant private _symbol = "PEPEZILLA"
uint8 constant private _decimals = 18
uint256 constant private _tTotal = startingSupply *
10**_decimals
bool public _hasLiqBeenAdded = false
bool _antiBlock
bool _antiSnipe
```

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	PepeZilla.sol#L102
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 timeSinceLastPair = 0
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	PepeZilla.sol#L365,366
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
bool checked
bool check
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	PepeZilla.sol#L195
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.

```
operator = newOperator
```

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.

Functions Analysis

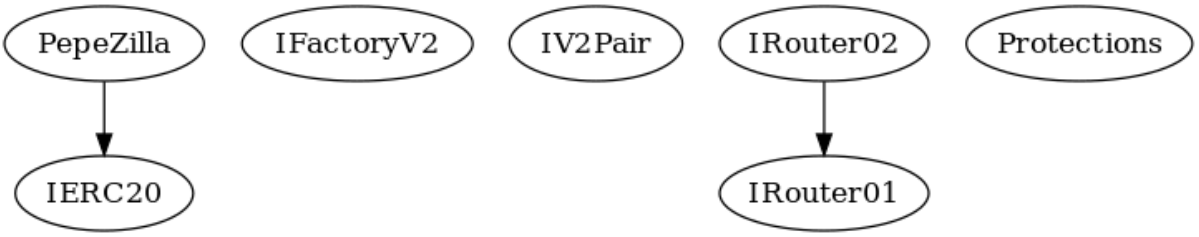
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
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	✓	-
Protections	Interface			
	checkUser	External	✓	-
	setLaunch	External	✓	-
	setLpPair	External	✓	-
	setProtections	External	✓	-
	removeSniper	External	✓	-

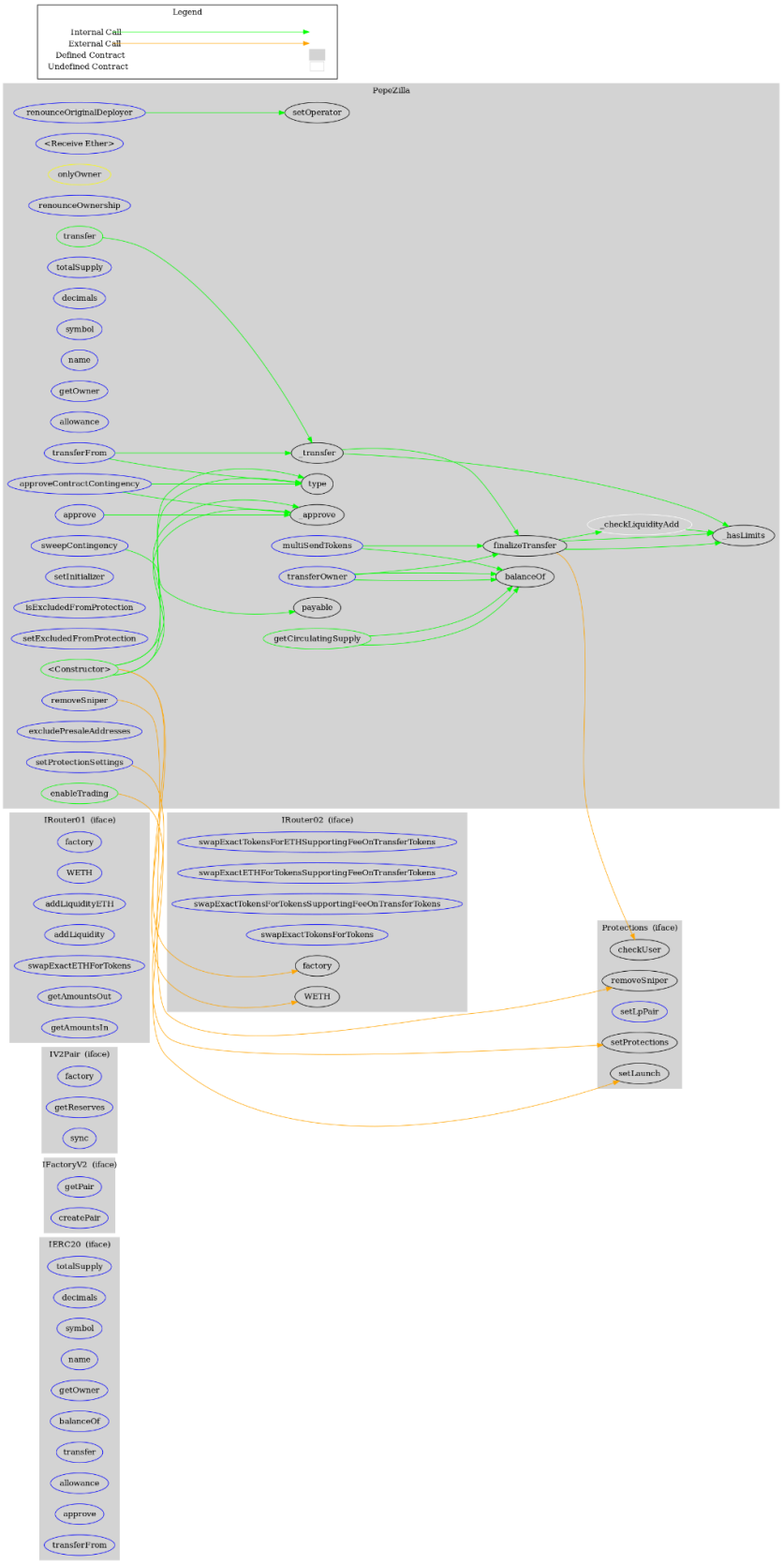
PepeZilla	Implementation	IERC20		
		Public	Payable	-
		External	Payable	-
	transferOwner	External	✓	onlyOwner
	renounceOwnership	External	✓	onlyOwner
	setOperator	Public	✓	-
	renounceOriginalDeployer	External	✓	-
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	allowance	External		-
	balanceOf	Public		-
	transfer	Public	✓	-
	approve	External	✓	-
	_approve	Internal	✓	
	approveContractContingency	External	✓	onlyOwner
	transferFrom	External	✓	-
	setInitializer	External	✓	onlyOwner
	isExcludedFromProtection	External		-
	setExcludedFromProtection	External	✓	onlyOwner
	getCirculatingSupply	Public		-
	removeSniper	External	✓	onlyOwner
	setProtectionSettings	External	✓	onlyOwner

	excludePresaleAddresses	External	✓	onlyOwner
	_hasLimits	Internal		
	_transfer	Internal	✓	
	_checkLiquidityAdd	Internal	✓	
	enableTrading	Public	✓	onlyOwner
	sweepContingency	External	✓	onlyOwner
	multiSendTokens	External	✓	onlyOwner
	finalizeTransfer	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
PepeZilla
renounceOriginalDeployer
setOperator
<Receive Ether>
onlyOwner
renounceOwnership
transfer
totalSupply
decimals
symbol
name
getOwner
allowance
transferFrom
transfer
approveContractContingency
type
approve
approve
checkLiquidityAdd
sweepContingency
multiSendTokens
finalizeTransfer
hasLimits
setInitializer
transferOwner
balanceOf
isExcludedFromProtection
payable
setExcludedFromProtection
getCirculatingSupply
<Constructor>
removeSniper
excludePresaleAddresses
setProtectionSettings
enableTrading
IRouter01 (iface)
factory
WETH
addLiquidityETH
addLiquidity
swapExactETHForTokens
getAmountsOut
getAmountsIn
IRouter02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokens
factory
WETH
Protections (iface)
checkUser
removeSniper
setLpPair
setProtections
setLaunch
IV2Pair (iface)
factory
getReserves
sync
IFactoryV2 (iface)
getPair
createPair
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom

Summary

PepeZilla contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. PepeZilla is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors 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.

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

Cyberscope

Audit Report

The Pepefather

May 2023

Network BSC

Address 0x3499e7Ecd7230c6fA37a81eEd404960714B83c2E

Audited by © cyberscope

Table of Contents

Table of Contents 1
Review 2
Audit Updates 2
Source Files 2
Findings Breakdown 3
Analysis 4
Diagnostics 5
L02 - State Variables could be Declared Constant 6
Description 6
Recommendation 6
L04 - Conformance to Solidity Naming Conventions 7
Description 7
Recommendation 7
L05 - Unused State Variable 9
Description 9
Recommendation 9
L14 - Uninitialized Variables in Local Scope 10
Description 10
Recommendation 10
L16 - Validate Variable Setters 11
Description 11
Recommendation 11
L20 - Succeeded Transfer Check 12
Description 12
Recommendation 12
Functions Analysis 13
Inheritance Graph 17
Flow Graph 18
Summary 19
Disclaimer 20
About Cyberscope 21

Review

Contract Name	ThePepeFather
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	500 runs
Explorer	https://bscscan.com/address/0x3499e7ecd7230c6fa37a81eed404960714b83c2e
Address	0x3499e7ecd7230c6fa37a81eed404960714b83c2e
Network	BSC
Symbol	PEPEFATHER
Decimals	18
Total Supply	1000.000.000.000

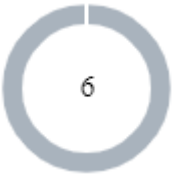
Audit Updates

Initial Audit	08 May 2023 https://github.com/cyberscope-io/audits/blob/main/pepezilla/v1/audit.pdf
Corrected Phase 2	23 May 2023

Source Files

Filename	SHA256
ThePepeFather.sol	9b36c774fb2b4271e073e745ffa42091281780255e2229df95976feee1ed9cbf

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	6

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	6	0	0	0

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	ThePepeFather.sol#L100,112
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 timeSinceLastPair = 0
bool public taxesAreLocked
```

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	ThePepeFather.sol#L33,106,107,108,109,110,118,230
Status	Unresolved

Description

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

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

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

```
function WETH() external pure returns (address);
uint256 constant private startingSupply = 999_999_999_999
string constant private _name = "The PepeFather"
string constant private _symbol = "PEPEFATHER"
uint8 constant private _decimals = 18
uint256 constant private _tTotal = startingSupply * 10**_decimals
bool public _hasLiqBeenAdded = false
address _initializer
```

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	ThePepeFather.sol#L100
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 timeSinceLastPair = 0
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	ThePepeFather.sol#L234
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
address constructorLP
address router
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	ThePepeFather.sol#L175
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.

```
operator = newOperator
```

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	ThePepeFather.sol#L336
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.

```
TOKEN.transfer(_owner, TOKEN.balanceOf(address(this)))
```

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

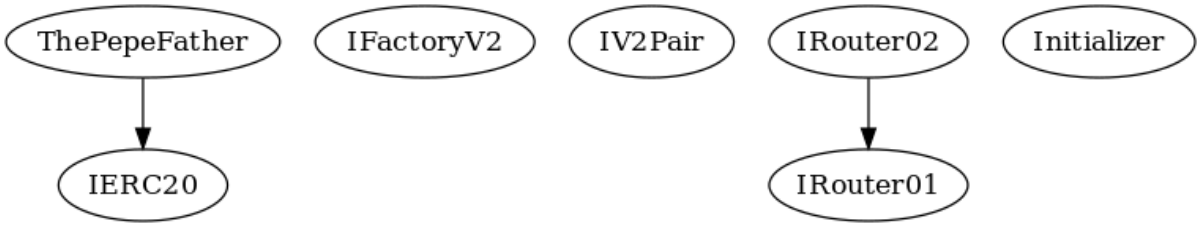
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
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	✓	-
Initializer	Interface			
	setLaunch	External	✓	-
	getConfig	External	✓	-
	setLpPair	External	✓	-

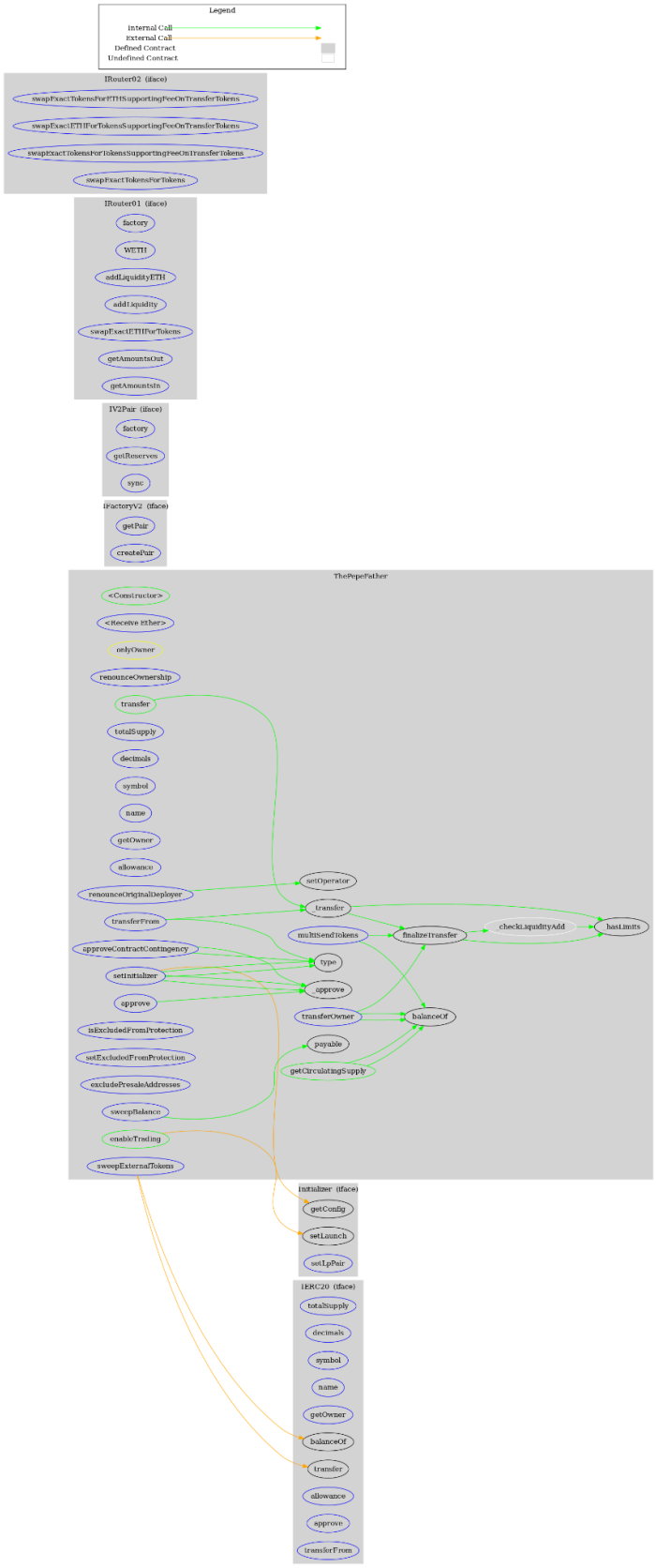
ThePepeFather	Implementation	IERC20		
		Public	Payable	-
		External	Payable	-
	transferOwner	External	✓	onlyOwner
	renounceOwnership	External	✓	onlyOwner
	setOperator	Public	✓	-
	renounceOriginalDeployer	External	✓	-
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	allowance	External		-
	balanceOf	Public		-
	transfer	Public	✓	-
	approve	External	✓	-
	_approve	Internal	✓	
	approveContractContingency	External	✓	onlyOwner
	transferFrom	External	✓	-
	setInitializer	External	✓	onlyOwner
	isExcludedFromProtection	External		-
	setExcludedFromProtection	External	✓	onlyOwner
	getCirculatingSupply	Public		-

	excludePresaleAddresses	External	✓	onlyOwner
	_hasLimits	Internal		
	_transfer	Internal	✓	
	_checkLiquidityAdd	Internal	✓	
	enableTrading	Public	✓	onlyOwner
	sweepBalance	External	✓	onlyOwner
	sweepExternalTokens	External	✓	onlyOwner
	multiSendTokens	External	✓	onlyOwner
	finalizeTransfer	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IRouter02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactTokensForTokens
IRouter01 (iface)
factory
WETH
addLiquidityETH
addLiquidity
swapExactETHForTokens
getAmountsOut
getAmountsIn
IV2Pair (iface)
factory
getReserves
sync
IFactoryV2 (iface)
getPair
createPair
ThePepeFather
<Constructor>
<Receive Ether>
onlyOwner
renounceOwnership
transfer
totalSupply
decimals
symbol
name
getOwner
allowance
renounceOriginalDeployer
transferFrom
approveContractContingency
setInitializer
approve
isExcludedFromProtection
setExcludedFromProtection
excludePresaleAddresses
sweepBalance
enableTrading
sweepExternalTokens
setOperator
transfer
multiSendTokens
type
approve
transferOwner
payable
getCirculatingSupply
finalizeTransfer
checkLiquidityAdd
hasLimits
balanceOf
initializer (iface)
getConfig
setLaunch
setLpPair
IERC20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
allowance
approve
transferFrom

Summary

The Pepefather contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. The Pepefather is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors 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.

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