Cyberscope

Audit Report

Pepe Based Version

May 2023

Network	BSC
Address	0x488237365040c58CF068e9B1740025FEa4b3157A
Audited by	© cyberscope

Table of Contents

Review

Contract Name	PepeBaseVersion
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0x488237365040c58cf068e9b1740025fea4b3157a
Address	0x488237365040c58cf068e9b1740025fea4b3157a
Network	BSC
Symbol	PBV
Decimals	9
Total Supply	420,000,000,000,000,000

Audit Updates

Initial Audit	10 May 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249a a4df6024fbfaa94f79ab2f44f3231
@openzeppelin/contracts/token/ERC20/ERC20.sol	bce14c3fd3b1a668529e375f6b70ffdf9cef 8c4e410ae99608be5964d98fa701
@openzeppelin/contracts/token/ERC20/extensions /IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166 689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db800 3d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a2 3a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/math/SafeMath.sol	0dc33698a1661b22981abad8e5c6f5ebca 0dfe5ec14916369a2935d888ff257a
contracts/Dividend/DividendPayingToken.sol	c06a38008dcbb7aa93741eb5284a7eb8f8 26bce61cf28255ced01098b526fb85
contracts/Dividend/DividendTracker.sol	f6f10584e1a385eff04b310a27d7d05cc41a c0d09eea14242083ca8b44b44b3d
contracts/Dividend/interfaces/DividendPayingToke nInterface.sol	c33782daca4a0163da5cc19bb3a328bf9d 4d1a946d2169dd1600c1b54245348a
contracts/Dividend/interfaces/DividendPayingToke nOptionalInterface.sol	0f4707bce9bd52788713a1019d79d6bc4a 5132bf50b0f474338e49f744183958
contracts/Dividend/interfaces/IWETH.sol	ac46c73f5638fd375114cfa5832527a9080 b531d0ec7a91197f657b2face9ba3
contracts/Dividend/libs/IterableMapping.sol	cc9ce9f7052817ffa93364588acf206e6a09 088cac97149fc000446077a31244

contracts/Dividend/libs/SafeMathInt.sol	350b52b749fba3b900af7f5ba567b508032b9f4465f0c178c08acd4b48b0c6b0
contracts/Dividend/libs/SafeMathUint.sol	1e6f62e6060c1fbe49992be9bc45c541c3651b6efc863871a36fcb6b992901eb
contracts/PepeBaseVersion.sol	27236f83599b0896607a8bc965992a7159aae2d13b62421ca5d5d0e7810c5252

Findings Breakdown

Critical	0
Medium	0
Minor / Informative	9

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	9	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
●	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
●	L11	Unnecessary Boolean equality	Unresolved
●	L12	Using Variables before Declaration	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/PepeBaseVersion.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, and 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 at 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	contracts/PepeBaseVersion.sol#L77,78,79,81
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`.

```
swapPair
marketing
usdt
dividendTracker
```

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	contracts/PepeBaseVersion.sol#L52,72
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 supply = 420 * 1e6 * 1e9 * 1e9
uint256 public gasForProcessing = 300000
```

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	contracts/PepeBaseVersion.sol#L7,42,58
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 view returns (address);
address public constant zeroAddr = address(0)
uint256 private constant reward = 3
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	contracts/PepeBaseVersion.sol#L205
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.

```
starting == true && block.number > (startBlock + 3)
```

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.

L12 - Using Variables before Declaration

Criticality	Minor / Informative
Location	contracts/PepeBaseVersion.sol#L251,252,253
Status	Unresolved

Description

The contract is using a variable before the declaration. This is usually happening either if it has not been declared yet or if the variable has been declared in a different scope. It is not a good practice to use a local variable before it has been declared.

```
uint256 iterations
uint256 claims
uint256 lastProcessedIndex
```

Recommendation

By declaring local variables before using them, the contract ensures that it operates correctly. It's important to be aware of this rule when working with local variables, as using a variable before it has been declared can lead to unexpected behavior and can be difficult to debug.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/PepeBaseVersion.sol#L251,252,253
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.

```
uint256 iterations
uint256 claims
uint256 lastProcessedIndex
```

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	contracts/PepeBaseVersion.sol#L78
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.

```
marketing = marketing_
```

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	contracts/PepeBaseVersion.sol#L276
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.

```
usdt.transfer(address(dividendTracker), usdtAmount)
```

Recommendation

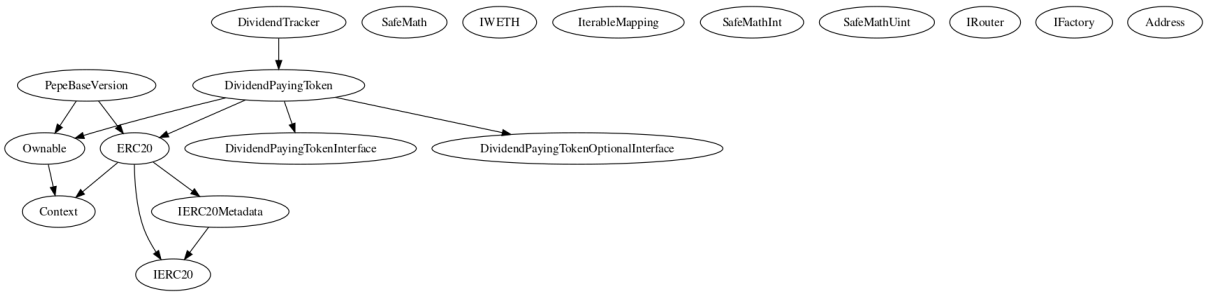
The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

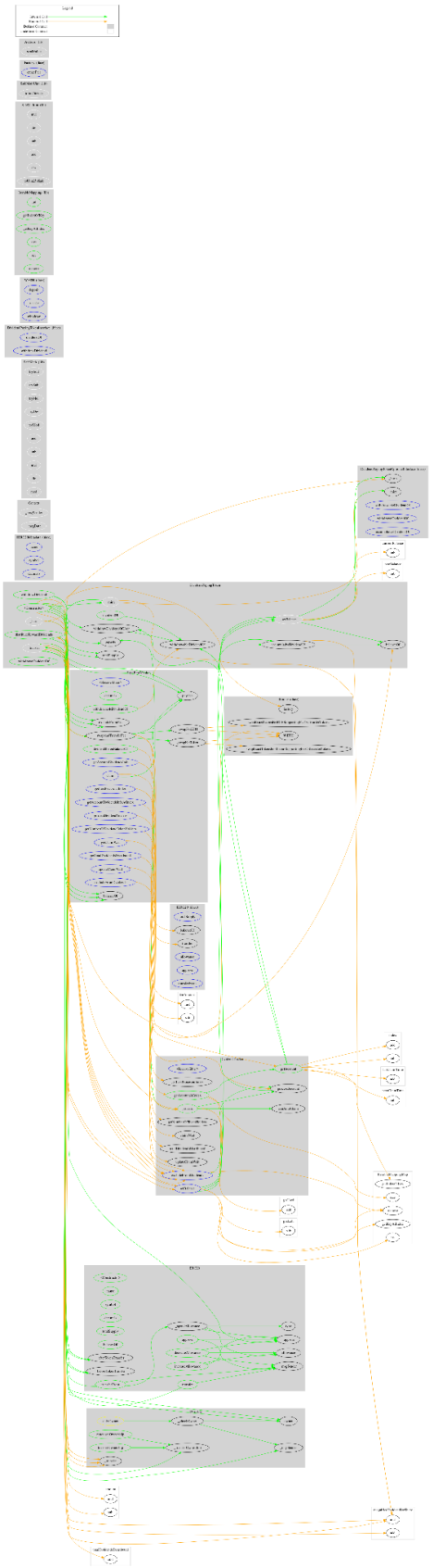
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IRouter	Interface			
	WETH	External		-
	factory	External		-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
IFactory	Interface			
	createPair	External	✓	-
Address	Library			
	sendValue	Internal	✓	
PepeBaseVersion	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	decimals	Public		-
	updateClaimWait	External	✓	onlyOwner

	getClaimWait	External		-
	getTotalDividendsDistributed	External		-
	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-
	excludeFromDividends	External	✓	onlyOwner
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	getLastProcessedIndex	External		-
	getNumberOfDividendTokenHolders	External		-
	excludeFromFee	Public	✓	onlyOwner
	_transfer	Internal	✓	
	_swapAndTransferFee	Private	✓	
	_swapForETH	Private	✓	
	_swapForToken	Private	✓	

Inheritance Graph



Flow Graph



Summary

Pepe Based Version contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. Pepe Based Version 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. The fees are locked at 10%, excluding the first sale which is 15%.

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