Cyberscope

Audit Report

Papi Pepe

May 2023

Network	BSC
Address	0x7678082AE812C8Cdb83A90C117557D098FCa4795
Audited by	© cyberscope

Table of Contents

Review

Contract Name	PPEPE
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://bscscan.com/address/0x7678082ae812c8cdb83a90c117557d098fca4795
Address	0x7678082ae812c8cdb83a90c117557d098fca4795
Network	BSC
Symbol	PPEPE
Decimals	9
Total Supply	1,000,000,000

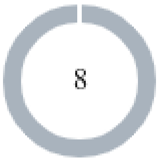
Audit Updates

Initial Audit	03 May 2023

Source Files

Filename	SHA256
contracts/PipaPepe.sol	78638277d50331057f82712a2a7f093f244bfae8f5fdb8d32dc02ab64219aab3

Findings Breakdown



●	Critical	0
●	Medium	0
●	Minor / Informative	8

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	8	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
●	RV	Redundant Variables	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
●	L19	Stable Compiler Version	Unresolved
●	L22	Potential Locked Ether	Unresolved

RV - Redundant Variables

Criticality	Minor / Informative
Location	contracts/PipaPepe.sol#L138,179
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 declares the variable `swapping` along with the `lockTheSwap` modifier. Since the contract does not include any swapping functionality, these variables are redundant.

```
bool private swapping;
...
modifier lockTheSwap() {
    swapping = true;
    _;
    swapping = 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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/PipaPepe.sol#L192,193
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`.

```
router
pair
```

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/PipaPepe.sol#L146,149
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 = 1 * 10 ** 9 * 10 ** _decimals
address public deadWallet = 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	contracts/PipaPepe.sol#L89,143,152,153,169
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);
uint8 private constant _decimals = 9
string private constant _name = "PAPI PEPE"
string private constant _symbol = "PPEPE"

struct valuesFromGetValues {
        uint256 rAmount;
        uint256 rTransferAmount;
        uint256 rRfi;
        uint256 rMarketing;
        uint256 tTransferAmount;
        uint256 tRfi;
        uint256 tMarketing;
    }
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, and 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	contracts/PipaPepe.sol#L113
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 sendValue(address payable recipient, uint256 amount) internal {
        require(
            address(this).balance >= amount,
            "Address: insufficient balance"
        );

        (bool success, ) = recipient.call{value: amount}("");
        require(
            success,
            "Address: unable to send value, recipient may have reverted"
        );
    }
```

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	contracts/PipaPepe.sol#L193
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.

```
pair = _pair
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/PipaPepe.sol#L2
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 ^0.8.17;
```

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.

L22 - Potential Locked Ether

Criticality	Minor / Informative
Location	contracts/PipaPepe.sol#L514
Status	Unresolved

Description

The contract contains Ether that has been placed into a Solidity contract and is unable to be transferred. Thus, it is impossible to access the locked Ether. This may produce a financial loss for the users that have called the payable method.

```
receive() external payable {}
```

Recommendation

The team is advised to either remove the payable method or add a withdraw functionality. it is important to carefully consider the risks and potential issues associated with locked Ether.

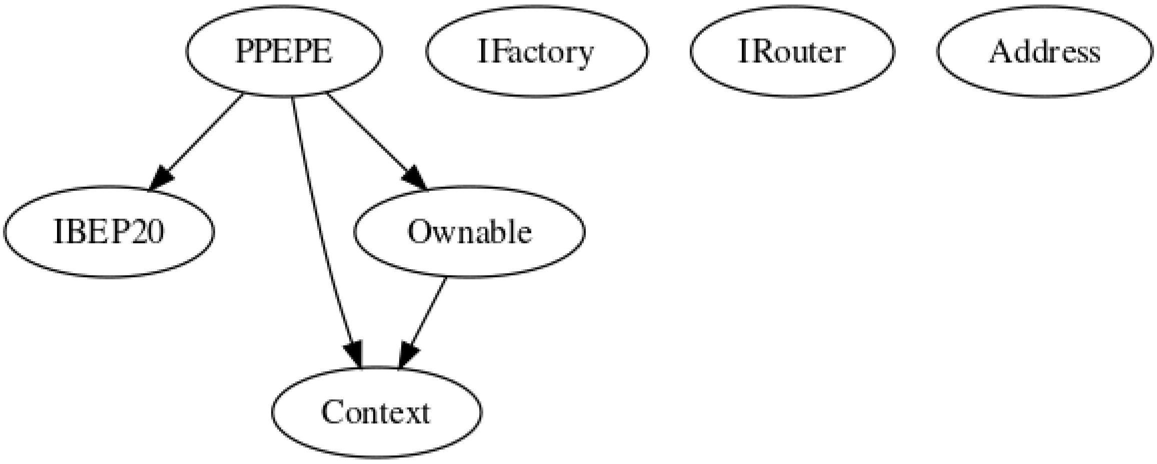
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IBEP20	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		-
	renounceOwnership	Public	✓	onlyOwner
	_setOwner	Private	✓	

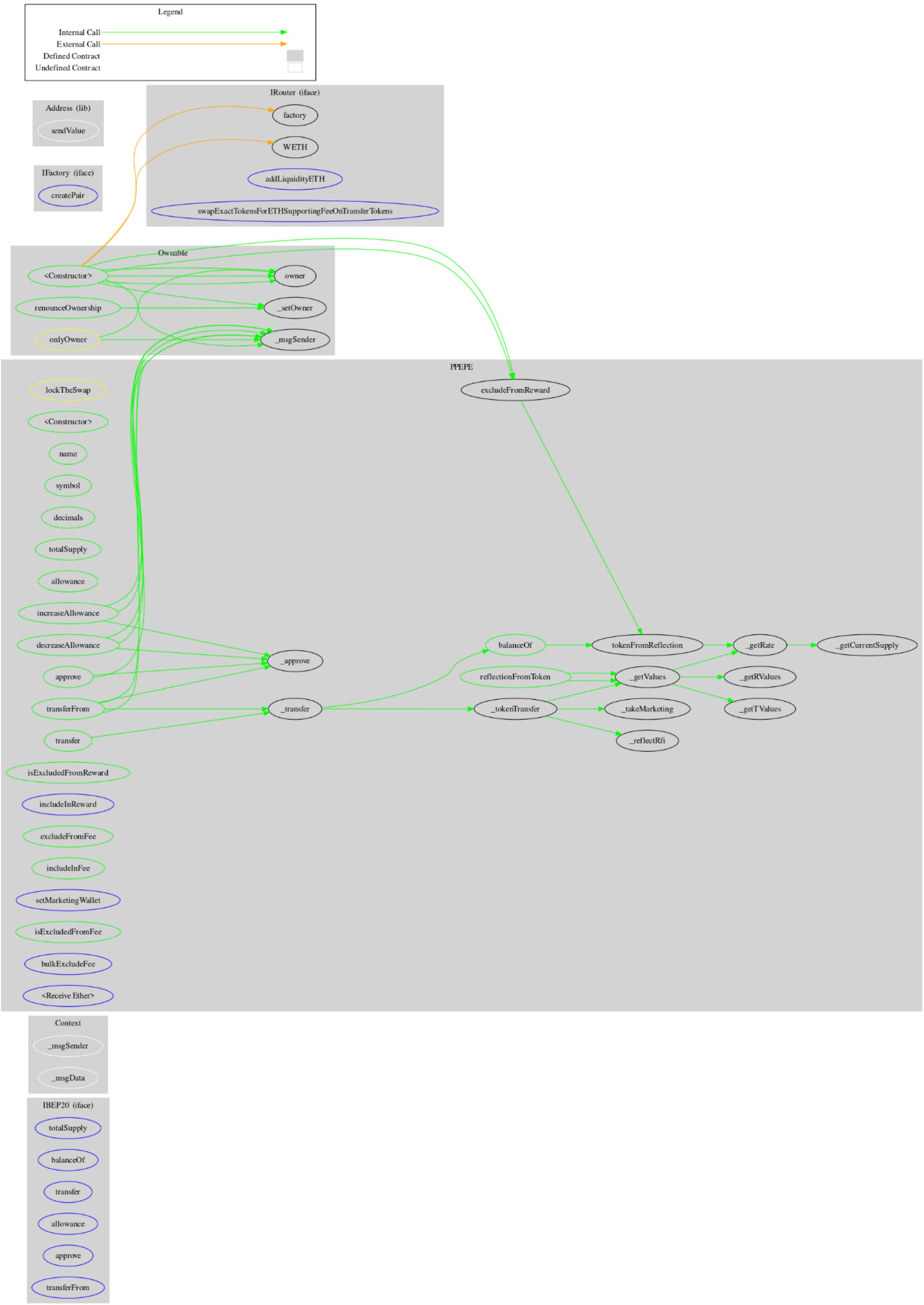
IFactory	Interface			
	createPair	External	✓	-
IRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Address	Library			
	sendValue	Internal	✓	
PPEPE	Implementation	Context, IBEP20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-

	decreaseAllowance	Public	✓	-
	transfer	Public	✓	-
	isExcludedFromReward	Public		-
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	setMarketingWallet	External	✓	onlyOwner
	isExcludedFromFee	Public		-
	_reflectRfi	Private	✓	
	_takeMarketing	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_approve	Private	✓	
	_transfer	Private	✓	
	_tokenTransfer	Private	✓	
	bulkExcludeFee	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IRouter (iface)
factory
WETH
addLiquidityETH
swapExactTokensForETHSupportingFeeOnTransferTokens
Address (lib)
sendValue
IFactory (iface)
createPair
Ownable
<Constructor>
owner
renounceOwnership
_setOwner
onlyOwner
_msgSender
PPEPE
excludeFromReward
lockTheSwap
<Constructor>
name
symbol
decimals
totalSupply
allowance
increaseAllowance
decreaseAllowance
approve
transferFrom
transfer
isExcludedFromReward
includeInReward
excludeFromFee
includeInFee
setMarketingWallet
isExcludedFromFee
bulkExcludeFee
<Receive Ether>
_approve
_transfer
balanceOf
tokenFromReflection
_getRate
_getCurrentSupply
reflectionFromToken
_getValues
_getRValues
_tokenTransfer
_takeMarketing
_getTValues
_reflectRfi
Context
_msgSender
_msgData
IBEP20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

Papi Pepe contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. Papi Pepe 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 2%.

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