Cyberscope

Audit Report

Pepe Galaxy

May 2023

Network	BSC
Address	0x890C9F9dEf560001cC013e3bf37F930d2195e1Bb
Audited by	© cyberscope

Table of Contents

Review

Contract Name	PepeGalaxy
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://bscscan.com/address/0x890c9f9def560001cc013e3bf37f930d2195e1bb
Address	0x890c9f9def560001cc013e3bf37f930d2195e1bb
Network	BSC
Symbol	PEPE G
Decimals	9
Total Supply	1,000,000,000

Audit Updates

Initial Audit	19 May 2023

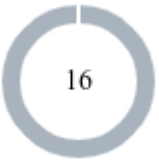
Source Files

Filename	SHA256
PepeGalaxy.sol	0b01a288ef5de7a4bf1286af290f8086de192e17a3baac6ddcacb9e29d150381

Overview

The audit scope is to check for security vulnerabilities, validate the business logic and propose potential optimizations. The contract is missing the fundamental principles of a Solidity smart contract regarding gas consumption, code readability, and data structures. According to the audit assesment findings, the contract cannot be assumed that it is in a production-ready state. Given these issues, it is not advisable to assume that the contract is in a production-ready state. The development team is strongly encouraged to re-evaluate the business logic and Solidity guidelines to ensure that the contract adheres to established best practices and security measures. It is recommended that the team review the contract's gas consumption and optimize it accordingly to minimize costs and improve the contract's efficiency. The code's readability should also be improved by simplifying function definitions and using descriptive variable names, as this will enhance the contract's auditability and maintenance.

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	16

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	16	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	Minor / Informative
Location	PepeGalaxy.sol#L838
Status	Unresolved

Description

Initially, the contract does not allow the non-excluded addresses to transfer tokens. The restriction can be resumed once the contract owner enables them.

```
if(!marketActive) {
    require(premarketUser[from],"cannot trade before the market opening");
}
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	TUU	Time Units Usage	Unresolved
●	RSD	Redundant Struct Declaration	Unresolved
●	RED	Redundant Event Declaration	Unresolved
●	UF	Unimplented Function	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	MC	Missing Check	Unresolved
●	RVD	Redundant Variable Declaration	Unresolved
●	UFM	Unused Fee Mechanism	Unresolved
●	RSF	Redundant Swap Functionality	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L18	Multiple Pragma Directives	Unresolved

●	L19	Stable Compiler Version	Unresolved

TUU - Time Units Usage

Criticality	Minor / Informative
Location	PepeGalaxy.sol#L655
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
uint public intervalSecondsForSwap = 60*60*15; // 15 hrs
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds` , `minutes` , `hours` , `days` , `weeks` and `years` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

RSD - Redundant Struct Declaration

Criticality	Minor / Informative
Location	PepeGalaxy.sol#L658
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 `userData` struct to keep track of each user's last buy. Since the struct only contains one property, it could be omitted. As a result the struct is redundant.

```
struct userData {uint lastBuyTime;}
mapping (address => userData) public userLastTradeData;
```

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.

The contract could modify the `userLastTradeData` mapping to return a uint256 integer for each address instead of a struct.

```
mapping (address => uint256) public userLastTradeData;
```

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	PepeGalaxy.sol#L665,666.674,679
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 following events but they are not used in the contract. As a result, these events are redundant.

```
event MarketingFeeCollected(uint amount);
event BuybackFeeCollected(uint amount);
event FeesSendToWalletStatusChanged(bool marketing, bool buyback);
event FeesStatusChanged(bool feesActive, bool buy, bool sell);
```

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.

UF - Unimplented Function

Criticality	Minor / Informative
Location	PepeGalaxy.sol#L782
Status	Unresolved

Description

A function with no code is a function that has been declared in a contract or interface but has no actual code that executes when the function is called. This could happen if a developer forgets to write the code for the function or intentionally leaves the function empty.

While a function with no code won't cause any syntax errors, it will prevent the contract from operating as intended since the function will not execute any actions. When calling a function with no code, the transaction will simply return without doing anything.

```
function setFeesAddress(address marketing, address buyback)
external onlyOwner {
    emit FeesAddressesChanged(marketing,buyback);
}
```

Recommendation

The team is advised to ensure that the function has the necessary code to execute the desired actions when called. If the function is intentionally left empty, it is important to document why it is empty to avoid confusion for other people who may work on the contract or interact with it in the future.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	PepeGalaxy.sol#L774,795,799.803
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 modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes.

```
function setBlockMultiBuys(bool _status) external onlyOwner {
    blockMultiBuys = _status;
    emit BlockMultiBuysChange(_status);
}
function editPremarketUser(address _target, bool _status) external
onlyOwner {
    premarketUser[_target] = _status;
    emit PremarketUserChanged(_status,_target);
}
function editExcludedFromFees(address _target, bool _status)
external onlyOwner {
    excludedFromFees[_target] = _status;
    emit ExcludeFromFeesChanged(_status,_target);
}
function editAutomatedMarketMakerPairs(address _target, bool
_status) external onlyOwner {
    automatedMarketMakerPairs[_target] = _status;
    emit AutomatedMarketMakerPairsChanged(_status,_target);
}
```

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.

MC - Missing Check

Criticality	Minor / Informative
Location	PepeGalaxy.sol#L808
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

The length of both arrays should be the same.

```
function airdrop(address[] memory _address, uint256[] memory
_amount) external onlyOwner {
    for(uint i=0; i< _amount.length; i++){
        address adr = _address[i];
        uint amnt = _amount[i] *10**decimals();
        super._transfer(owner(), adr, amnt);
    }
    // events from ERC20
}
```

Recommendation

The team is advised to properly check the variables according to the required specifications.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	PepeGalaxy.sol#L652,836
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 `trade_type` on each transaction, which stores the type of the transaction (buy, sale, trade), as well as, the variable `minimumWeiForTokenomics`. These variables are not being used in any meaningful way by the contract. As a result, they are redundant.

```
uint public minimumWeiForTokenomics = 1 * 10**17; // 0.1 BNB
uint trade_type = 0;
```

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.

UFM - Unused Fee Mechanism

Criticality	Minor / Informative
Location	PepeGalaxy.sol#L643,644,645,646,647,648,711,763
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 variables and functions as part of a fee mechanism. However, this mechanism is not being used by the transaction flow. As a result, these variables and functions are redundant.

```
uint public buyMarketingFee = 0;
uint public sellMarketingFee = 0;
uint public buyBuybackFee = 0;
uint public sellBuybackFee = 0;
uint public totalBuyFee = buyMarketingFee + buyBuybackFee;
uint public totalSellFee = sellMarketingFee + sellBuybackFee;
...
function KKPunishOn() internal { ... }
function setLaunchFee() external onlyOwner { ... }
```

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.

RSF - Redundant Swap Functionality

Criticality	Minor / Informative
Location	PepeGalaxy.sol#L864
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 implements a swap functionality. However, there is no functionality in the contract for collecting fees. Hence, the contract's balance will always be zero and the swap functionality will never trigger. As a result, this functionality is redundant.

```
bool overMinimumTokenBalance = balanceOf(address(this)) >=
minimumTokensBeforeSwap;
// marketing auto-bnb
if (swapAndLiquifyEnabled && balanceOf(pancakeV2Pair) > 0) {
    // if contract has X tokens, not sold since Y time, sell Z
tokens
    if (overMinimumTokenBalance && startTimeForSwap +
intervalSecondsForSwap <= block.timestamp) {
        startTimeForSwap = block.timestamp;
        // sell to bnb
        swapTokens(tokensToSwap);
    }
}
```

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	PepeGalaxy.sol#L692,693
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`.

```
maxSellTxAmount
maxBuyTxAmount
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	PepeGalaxy.sol#L620,638,658,711,719,732,746,754,758,774,778,785,795,799,803,808
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);
bool public KKLaunched
struct userData {uint lastBuyTime;}

function KKPunishOn() internal {
        buyMarketingFee = 49;
        sellMarketingFee = 50;
        buyBuybackFee = 49;
        sellBuybackFee = 50;
        totalBuyFee = buyMarketingFee + buyBuybackFee;
        totalSellFee = sellMarketingFee + sellBuybackFee;
    }
bool _createPair
address _pair
uint _value

...
```

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	PepeGalaxy.sol#L423
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 _burn(address account, uint256 amount) internal virtual {
        require(account != address(0), "ERC20: burn from the zero
address");

        _beforeTokenTransfer(account, address(0), amount);

        uint256 accountBalance = _balances[account];
...
        }
        _totalSupply -= amount;

        emit Transfer(account, address(0), amount);

        _afterTokenTransfer(account, address(0), amount);
    }
```

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	PepeGalaxy.sol#L724,727
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.

```
pancakeV2Pair = _pancakeV2Pair
pancakeV2Pair = _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.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	PepeGalaxy.sol#L6,91,121,148,533,613
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.17 < 0.9.0;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	PepeGalaxy.sol#L6,91,121,148,533,613
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 ^0.8.0;
pragma solidity ^0.8.17 < 0.9.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

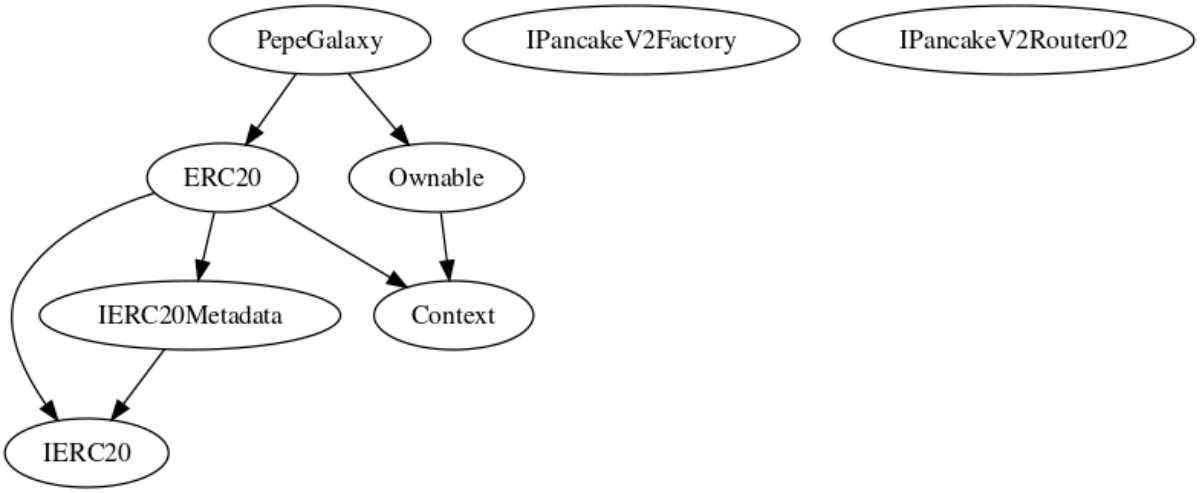
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		

ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
Ownable	Implementation	Context		

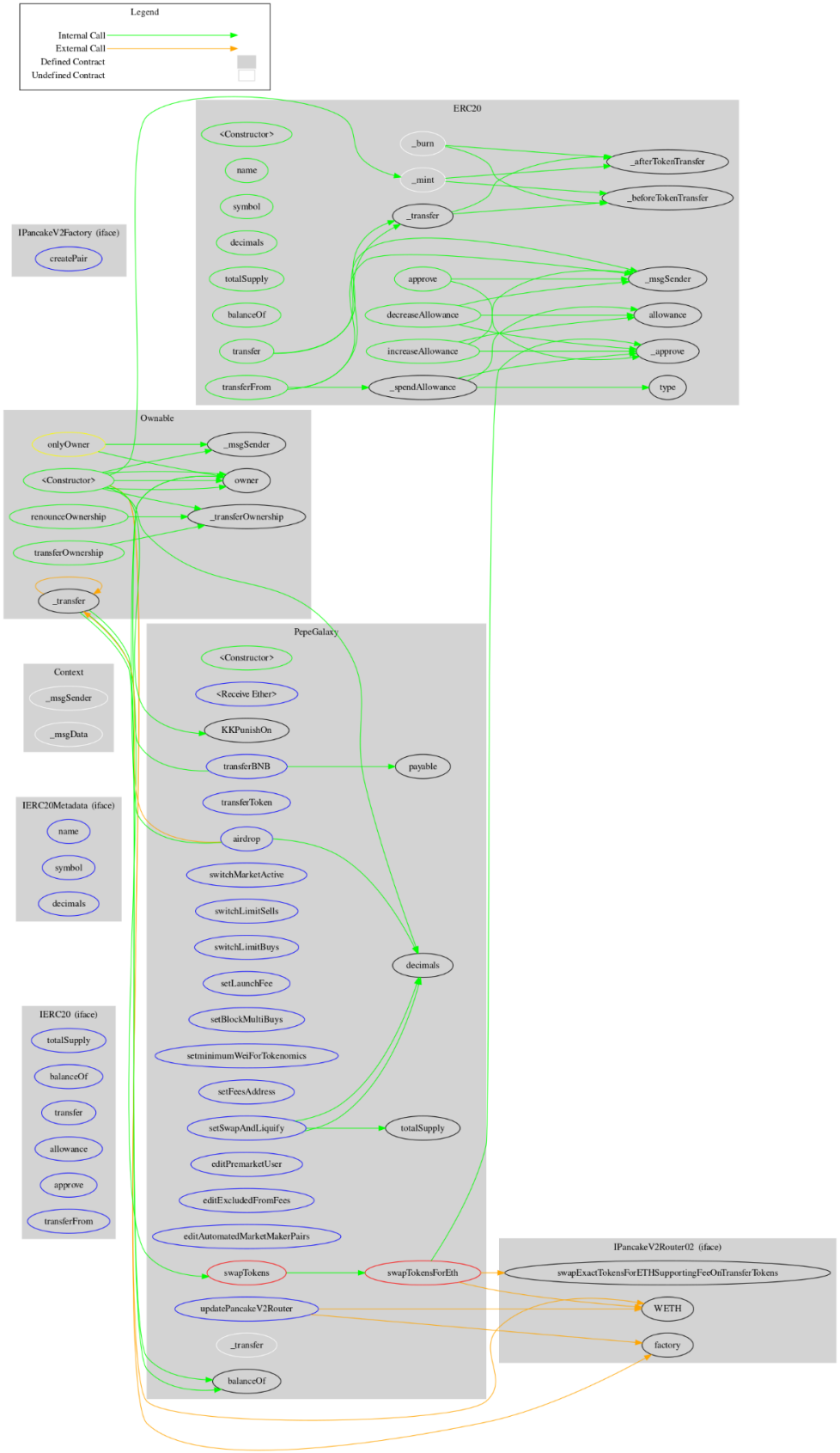
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IPancakeV2Factory	Interface			
	createPair	External	✓	-
IPancakeV2Router02	Interface			
	factory	External		-
	WETH	External		-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
PepeGalaxy	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	decimals	Public		-
		External	Payable	-
	KKPunishOn	Internal	✓	
	updatePancakeV2Router	External	✓	onlyOwner
	transferToken	External	✓	onlyOwner
	transferBNB	External	✓	onlyOwner

	switchMarketActive	External	✓	onlyOwner
	switchLimitSells	External	✓	onlyOwner
	switchLimitBuys	External	✓	onlyOwner
	setLaunchFee	External	✓	onlyOwner
	setBlockMultiBuys	External	✓	onlyOwner
	setminimumWeiForTokenomics	External	✓	onlyOwner
	setFeesAddress	External	✓	onlyOwner
	setSwapAndLiquify	External	✓	onlyOwner
	editPremarketUser	External	✓	onlyOwner
	editExcludedFromFees	External	✓	onlyOwner
	editAutomatedMarketMakerPairs	External	✓	onlyOwner
	airdrop	External	✓	onlyOwner
	swapTokensForEth	Private	✓	
	swapTokens	Private	✓	
	_transfer	Internal	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
transferFrom
_burn
_mint
_transfer
approve
decreaseAllowance
increaseAllowance
_spendAllowance
_afterTokenTransfer
_beforeTokenTransfer
_msgSender
allowance
_approve
type
IPancakeV2Factory (iface)
createPair
Ownable
onlyOwner
<Constructor>
renounceOwnership
transferOwnership
_transfer
_msgSender
owner
_transferOwnership
Context
_msgSender
_msgData
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
PepeGalaxy
<Constructor>
<Receive Ether>
KKPunishOn
transferBNB
transferToken
airdrop
switchMarketActive
switchLimitSells
switchLimitBuys
setLaunchFee
setBlockMultiBuys
setminimumWeiForTokenomics
setFeesAddress
setSwapAndLiquify
editPremarketUser
editExcludedFromFees
editAutomatedMarketMakerPairs
swapTokens
updatePancakeV2Router
_transfer
balanceOf
swapTokensForEth
payable
decimals
totalSupply
IPancakeV2Router02 (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
factory

Summary

Pepe Galaxy 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. A multi-wallet signing pattern will provide security against potential hacks.

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