Cyberscope

Audit Report

Peanutz

April 2024

Network BSC

Address 0xE625d1dEb48C398Ca85876e5a9F083332474dd82

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
●	ZD	Zero Division	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MU	Modifiers Usage	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	PMRM	Potential Mocked Router Manipulation	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	PNUTZToken
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://bscscan.com/address/0xe625d1deb48c398ca85876e5a9f083332474dd82
Address	0xe625d1deb48c398ca85876e5a9f083332474dd82
Network	BSC
Symbol	PNUTZ
Decimals	9
Total Supply	500,000,000
Badge Eligibility	Yes

Audit Updates

Initial Audit	01 Apr 2024

Source Files

Filename	SHA256
PNUTZToken.sol	0873660d965dc6ad14bdf3e6f111f7980477d66eedf58b3c2e2c1c5014e8547b
libraries/TaxableToken.sol	5efb9bb57cf557d638c790e17c58d03e744d4e2c0bfd568aa8f0821ff5f47df8

libraries/TaxDistributor.sol	662235c81b331dbf2270aa8e94ab2c416c dcaf7a4d42c836f5a57821149932f4
libraries/Recoverable.sol	10570b0238a33933a903632a7ed989eba9 360179edb520c347e1c11a7c1d27e4
libraries/BEP20Burnable.sol	96517b0a60b810d3db2646aa3d6d44b50 1e323b57e30479106675494c50d12a6
libraries/BEP20Base.sol	2c4f94b4ce861b0b1207040117a7314a58 63879b831cc3c5a18b7c1abf1b549f
libraries/BEP20.sol	44b0815befb3829ed77173890ded47d08b a9eb586bf86a8add1f3a863cfeb5fb
libraries/AntiWhaleToken.sol	e072dfb1cc20e261c5c53e45f6134330774 cd58c6924e1a1438aaf1ae3dc2750
interfaces/IBEP20.sol	f313a08143bc9ce32d8966602ba46fcf1a6 8668372dd3928aeab8341ff2eeb87
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router02.sol	a2900701961cb0b6152fc073856b972564f 7c798797a4a044e83d2ab8f0e8d38
@uniswap/v2-periphery/contracts/interfaces/IUniswapV2Router01.sol	0439ffe0fd4a5e1f4e22d71ddbda76d63d6 1679947d158cba4ee0a1da60cf663
@uniswap/v2-core/contracts/interfaces/IUniswapV2Factory.sol	51d056199e3f5e41cb1a9f11ce581aa3e19 0cc982db5771ffeef8d8d1f962a0d
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a2 3a4baa0b5bd9add9fb6d6a1549814a
@openzeppelin/contracts/utils/structs/EnumerableSet.sol	d8508ddcfb255875b542cef61ad9bcc9a2 d74453bf92c914ee6a546b876ba42b
@openzeppelin/contracts/token/ERC20/IERC20.sol	94f23e4af51a18c2269b355b8c7cf4db800 3d075c9c541019eb8dcf4122864d5
@openzeppelin/contracts/token/ERC20/ERC20.sol	bce14c3fd3b1a668529e375f6b70ffdf9cef 8c4e410ae99608be5964d98fa701

@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/access/Ownable.sol	9353af89436556f7ba8abb3f37a6677249aa4df6024fbfaa94f79ab2f44f3231

Findings Breakdown

Critical	2
Medium	0
Minor / Informative	12

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	2	0	0	0
Medium	0	0	0	0
Minor / Informative	12	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	PNUTZToken.sol#L226,263
Status	Unresolved

Description

The contract owner has the authority to stop transactions, as described in detail in sections `ZD`, `PMRM` and `PTRP`. As a result, the contract might operate as a honeypot.

Recommendation

It is recommended to follow the `ZD`, `PMRM` and `PTRP` findings recommendation to mitigate the `Stops Transactions` finding.

ZD - Zero Division

Criticality	Critical
Location	libraries/TaxableToken.sol#L134
Status	Unresolved

Description

The contract is using variables that may be set to zero as denominators. This can lead to unpredictable and potentially harmful results, such as a transaction revert. Specifically, the contract calculates `liquidityAmount` by multiplying the `tokenAmount` with `feeConfiguration.liquidityFeeRatio` and then dividing by `(FEE_PRECISION - feeConfiguration.burnFeeRatio)`. This operation poses a significant risk of a division by zero error, specifically if `feeConfiguration.burnFeeRatio` is set equal to `FEE_PRECISION`. In such a scenario, the denominator becomes zero, leading to an unpredictable state that could cause the transaction to revert, disrupting the contract's intended functionality and potentially leading to harmful outcomes for the contract's operations and its users.

```
uint256 liquidityAmount = (tokenAmount *
feeConfiguration.liquidityFeeRatio) /
                (FEE_PRECISION -
feeConfiguration.burnFeeRatio);
```

Recommendation

It is important to handle division by zero appropriately in the code to avoid unintended behavior and to ensure the reliability and safety of the contract. The contract should ensure that the divisor is always non-zero before performing a division operation. It should prevent the variables to be set to zero, or should not allow the execution of the corresponding statements. It is recommended to implement a safeguard check to ensure that `FEE_PRECISION` is always greater than `feeConfiguration.burnFeeRatio`.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	libraries/TaxableToken.sol#L59,60,62
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
numTokensToSwap
autoProcessFees
liquidityOwner
```

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.

MU - Modifiers Usage

Criticality	Minor / Informative
Location	libraries/TaxableToken.sol#L47,56
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(!_collectors.contains(account), "Already fee
collector");
require(_collectors.contains(account), "Not fee collector");
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	libraries/TaxableToken.sol#L180
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

Before executing the swap operation, the contract currently checks if `liquifyAmount > 0` and `if balanceOf(swapPair) == 0`, opting not to proceed with the swap if the pair has no liquidity. However, this check is insufficient to ensure the swap's success, as it only verifies the existence of any liquidity rather than the adequacy of liquidity to cover the specific swap amount.

```
address[] memory path = new address[](2);
path[0] = address(this);
path[1] = swapRouter.WETH()
_approve(address(this), address(swapRouter), tokenAmount)
// make the swap
swapRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
    tokenAmount,
    minAmountOut,
    path,
    address(this),
    block.timestamp
);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

To address this gap and further mitigate the risk of transaction reverts due to inadequate liquidity, it is recommended to adjust the condition to `balanceOf(swapPair) < liquifyAmount`. This enhanced check ensures that the swap only proceeds if there is enough liquidity in the pair to cover the amount being swapped, thereby significantly reducing the likelihood of failed transactions due to liquidity issues.

PMRM - Potential Mocked Router Manipulation

Criticality	Minor / Informative
Location	PNUTZToken.sol#L263
Status	Unresolved

Description

The contract includes a method that allows the owner to modify the router address and create a new pair. While this feature provides flexibility, it introduces a security threat. The owner could set the router address to any contract that implements the router's interface, potentially containing malicious code. In the event of a transaction triggering the swap functionality with such a malicious contract as the router, the transaction may be manipulated.

```
    function setSwapRouter(address newRouter) external override
onlyOwner {
        _setSwapRouter(newRouter);
    }
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	libraries/TaxDistributor.sol#L83
Status	Unresolved

Description

The contract sends funds to `collector` addresses as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
payable(collector).transfer(share);
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	libraries/TaxableToken.sol#L258 PNUTZToken.sol#L234
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `numTokensToSwap` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
uint256 contractTokenBalance = balanceOf(address(this));
if (contractTokenBalance >= numTokensToSwap) {
    _processFees(numTokensToSwap, 0);
}
...
    function setNumTokensToSwap(uint256 amount) external
override onlyOwner {
        numTokensToSwap = amount;
    }
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the exchange reserves. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	PNUTZToken.sol#L80
Status	Unresolved

Description

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, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function setIsExcludedFromAntiWhale(
        address account,
        bool excluded
    ) external onlyOwner {
        _excludedFromAntiWhale[account] = excluded;
        emit ExcludedFromAntiWhale(account, excluded);
    }
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	PNUTZToken.sol#L136
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `recoverTokens` function.

```
    function recoverTokens(
        address tokenAddress,
        address to,
        uint256 tokenAmount
    ) external override onlyOwner {
        _recoverTokens(tokenAddress, to, tokenAmount);
    }
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	libraries/TaxableToken.sol#L136,161,265,267
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
uint256 taxAmount = (amount * taxFee) / FEE_PRECISION
uint256 burnAmount = (taxAmount *
feeConfiguration.burnFeeRatio) / FEE_PRECISION
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	PNUTZToken.sol#L42,187
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.

```
payable(feeReceiver_).transfer(msg.value)
liquidityOwner = newOwner
```

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	PNUTZToken.sol#L3 libraries/TaxDistributor.sol#L3 libraries/TaxableToken.sol#L3 libraries/Recoverable.sol#L3 libraries/BEP20.sol#L3 libraries/AntiWhaleToken.sol#L3 interfaces/IBEP20.sol#L3
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;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	libraries/Recoverable.sol#L34
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.

```
IBEP20(tokenAddress).transfer(to, tokenAmount)
```

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**
PNUTZToken	Implementation	BEP20Base, AntiWhaleToken, BEP20Burnable, Ownable, Recoverable, TaxableToken		
		Public	Payable	BEP20Base AntiWhaleToken TaxableToken TaxDistributor
	setMaxTokenPerWallet	External	✓	onlyOwner
	isExcludedFromAntiWhale	Public		-
	setIsExcludedFromAntiWhale	External	✓	onlyOwner
	burn	External	✓	onlyOwner
	burnFrom	External	✓	onlyOwner
	_beforeTokenTransfer	Internal	✓	
	recoverEth	External	✓	onlyOwner
	recoverTokens	External	✓	onlyOwner
	setAutoprocessFees	External	✓	onlyOwner
	addFeeCollector	External	✓	onlyOwner
	setIsLpPool	External	✓	onlyOwner
	setIsExcludedFromFees	External	✓	onlyOwner
	distributeFees	External	✓	onlyOwner

	processFees	External	✓	onlyOwner
	removeFeeCollector	External	✓	onlyOwner
	setLiquidityOwner	External	✓	onlyOwner
	setNumTokensToSwap	External	✓	onlyOwner
	updateFeeCollectorShare	External	✓	onlyOwner
	setFeeConfiguration	External	✓	onlyOwner
	setSwapRouter	External	✓	onlyOwner
	_transfer	Internal	✓	
TaxableToken	Implementation	BEP20Base, TaxDistributor		
		Public	✓	-
		External	Payable	-
	isExcludedFromFees	Public		-
	_setIsExcludedFromFees	Internal	✓	
	_setIsLpPool	Internal	✓	
	isLpPool	Public		-
	_setSwapRouter	Internal	✓	
	_setFeeConfiguration	Internal	✓	
	_processFees	Internal	✓	lockTheSwap
	_swapTokensForEth	Private	✓	
	_addLiquidity	Private	✓	
	_pairFor	Internal		
	_transfer	Internal	✓	

	setAutoprocessFees	External	✓	-
	setIsLpPool	External	✓	-
	setIsExcludedFromFees	External	✓	-
	processFees	External	✓	-
	setLiquidityOwner	External	✓	-
	setNumTokensToSwap	External	✓	-
	setFeeConfiguration	External	✓	-
	setSwapRouter	External	✓	-
TaxDistributor	Implementation	BEP20Base		
		Public	✓	-
	isFeeCollector	Public		-
	feeCollectorShare	Public		-
	_addFeeCollector	Internal	✓	
	_removeFeeCollector	Internal	✓	
	_updateFeeCollectorShare	Internal	✓	
	_distributeFees	Internal	✓	
	addFeeCollector	External	✓	-
	removeFeeCollector	External	✓	-
	updateFeeCollectorShare	External	✓	-
	distributeFees	External	✓	-
Recoverable	Implementation			

	_recoverEth	Internal	✓	
	_recoverTokens	Internal	✓	
	recoverEth	External	✓	-
	recoverTokens	External	✓	-
BEP20Burnable	Implementation	Context, BEP20Base		
	burn	External	✓	-
	_burnFrom	Internal	✓	
	burnFrom	External	✓	-
BEP20Base	Implementation	BEP20		
		Public	✓	BEP20
	decimals	Public		-
BEP20	Implementation	ERC20, IBEP20		
		Public	✓	ERC20
	_beforeTokenTransfer	Internal	✓	
	_transfer	Internal	✓	
	_mint	Internal	✓	
AntiWhaleToken	Implementation	BEP20Base		
		Public	✓	-
	isExcludedFromAntiWhale	Public		-

	_setMaxTokenPerWallet	Internal	✓	
	_beforeTokenTransfer	Internal	✓	antiWhale
IBEP20	Interface	IERC20		
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-

	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-
	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
Context	Implementation			
	_msgSender	Internal		

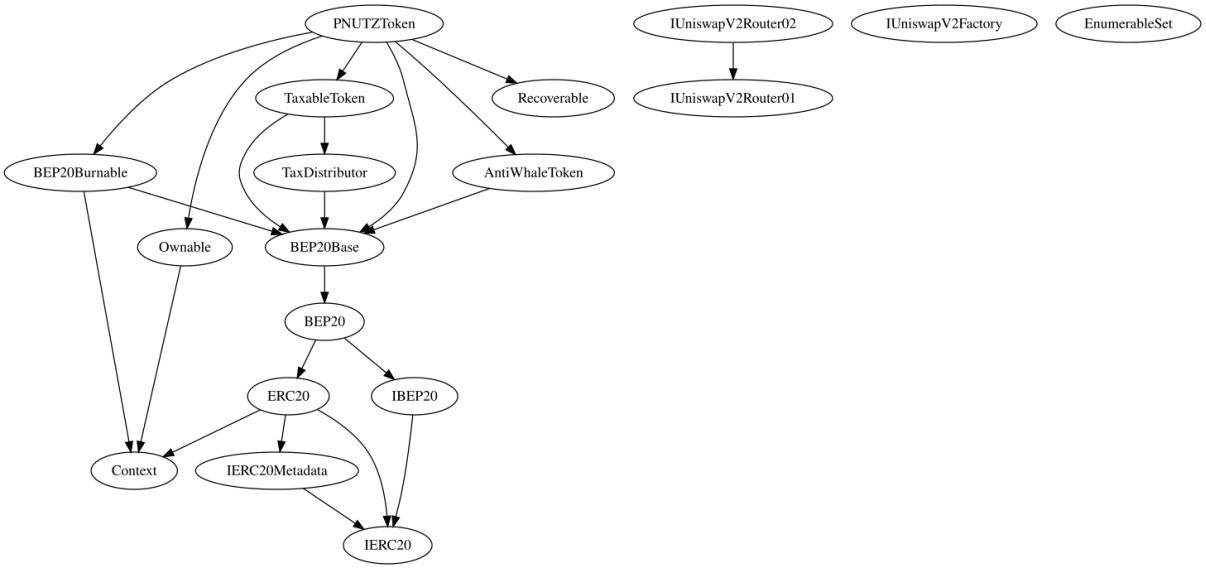
	_msgData	Internal		
EnumerableSet	Library			
	_add	Private	✓	
	_remove	Private	✓	
	_contains	Private		
	_length	Private		
	_at	Private		
	_values	Private		
	add	Internal	✓	
	remove	Internal	✓	
	contains	Internal		
	length	Internal		
	at	Internal		
	values	Internal		
	add	Internal	✓	
	remove	Internal	✓	
	contains	Internal		
	length	Internal		
	at	Internal		
	values	Internal		
	add	Internal	✓	
	remove	Internal	✓	

	contains	Internal		
	length	Internal		
	at	Internal		
	values	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-

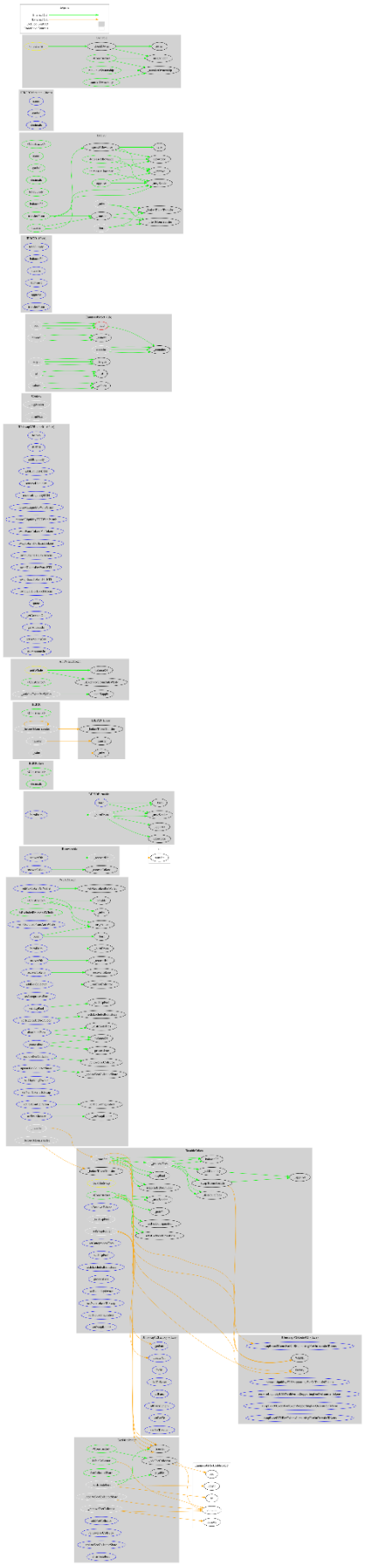
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner

	_transferOwnership	Internal	✓	

Inheritance Graph



Flow Graph



Summary

Peanutz 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 stop transactions. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 20% fees.

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