Cyberscope

Audit Report

Pepe Burn

May 2023

SHA256 4d44b9fde9b0c22387b7da420137e35a9c6aba8c1901f4b723a829e9c92a53f0

Audited by © cyberscope

Table of Contents

Review

Contract Name	BEP20PEPEBURN
Testing Deploy	https://testnet.bscscan.com/address/0xc1a2cdf7e2260f43094c0cc91f1f9882b41ad164
Symbol	PEPEB
Decimals	9
Total Supply	666,420,690,000,001

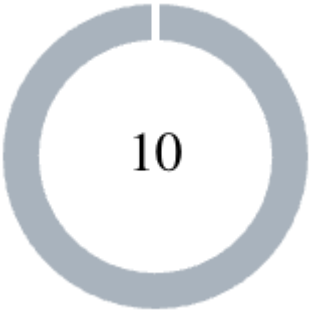
Audit Updates

Initial Audit	19 May 2023

Source Files

Filename	SHA256
contracts/testingDeploy/burn.sol	4d44b9fde9b0c22387b7da420137e35a9c6aba8c1901f4b723a829e9c92a53f0

Findings Breakdown



Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	10	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	burn.sol#L412
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enable the owner will not be able to disble them again.

```
if(!authorizations[sender] && !authorizations[recipient]){
    require(tradingOpen,"Trading not open yet");
}
```

Recommendation

The contract could embody a check for not allowing setting the _maxTxAmount less than a reasonable amount. A suggested implementation could check that the maximum amount should be more than a fixed percentage of the total supply. 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
	PTRP	Potential Transfer Revert Propagation	Unresolved
	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
	L05	Unused State Variable	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L13	Divide before Multiply Operation	Unresolved
	L20	Succeeded Transfer Check	Unresolved

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	burn.sol#L533
Status	Unresolved

Description

The contract sends funds to a `marketingWallet` 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.

```
(bool tmpSuccess,) = payable(marketingFeeReceiver).call{value:
amountBNBMarketing, gas: 30000}("");
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/testingDeploy/burn.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, 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 on 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/testingDeploy/burn.sol#L361,364,367
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
distributor
```

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/testingDeploy/burn.sol#L155,163,176
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.

```
address _tokenowner
BEP20 RWRD = BEP20(0x55d398326f99059fF7754852469990 27B3197955)
uint256 public dividendsPerShareAccuracyFactor = 10 ** 36
```

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/testingDeploy/burn.sol#L117,126,154,155,163,199,318,326,329,338,553,560,568,579,587,596,601,611,653,654,657,659,661,662,663
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.

```
event Authorize_Wallet(address Wallet, bool Status);
function WETH() external pure returns (address);
address _token
address _tokenowner
BEP20 RWRD = BEP20(0x55d398326f99059fF775485246999027B3197955)
uint256 _minDistribution
uint256 _minPeriod
address immutable WBNB
uint256 public constant totalSupply = 666_420_690_000_001 *
10**decimals
mapping (address => mapping (address => uint256)) _allowances
uint256 public constant feeDenominator  = 100

...
```

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	contracts/testingDeploy/burn.sol#L155
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.

```
address _tokenowner
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	contracts/testingDeploy/burn.sol#L200,580,588
Status	Unresolved

Description

Events are a way to record and log information about changes or actions that occur within a contract. They are often used to notify external parties or clients about events that have occurred within the contract, such as the transfer of tokens or the completion of a task.

It's important to carefully design and implement the events in a contract, and to ensure that all required events are included. It's also a good idea to test the contract to ensure that all events are being properly triggered and logged.

```
minPeriod = _minPeriod
sellMultiplier = _sell
marketingFee = _marketingFee
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	contracts/testingDeploy/burn.sol#L461,462
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 feeAmount =
amount.mul(totalFee).mul(multiplier).div(feeDenominator * 100)
uint256 burnTokens = feeAmount.mul(burnFee).div(totalFee)
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/testingDeploy/burn.sol#L277
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.

```
RWRD.transfer(shareholder, amount)
```

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**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
BEP20	Interface			
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Auth	Implementation			
		Public	✓	-
	authorize	External	✓	onlyOwner

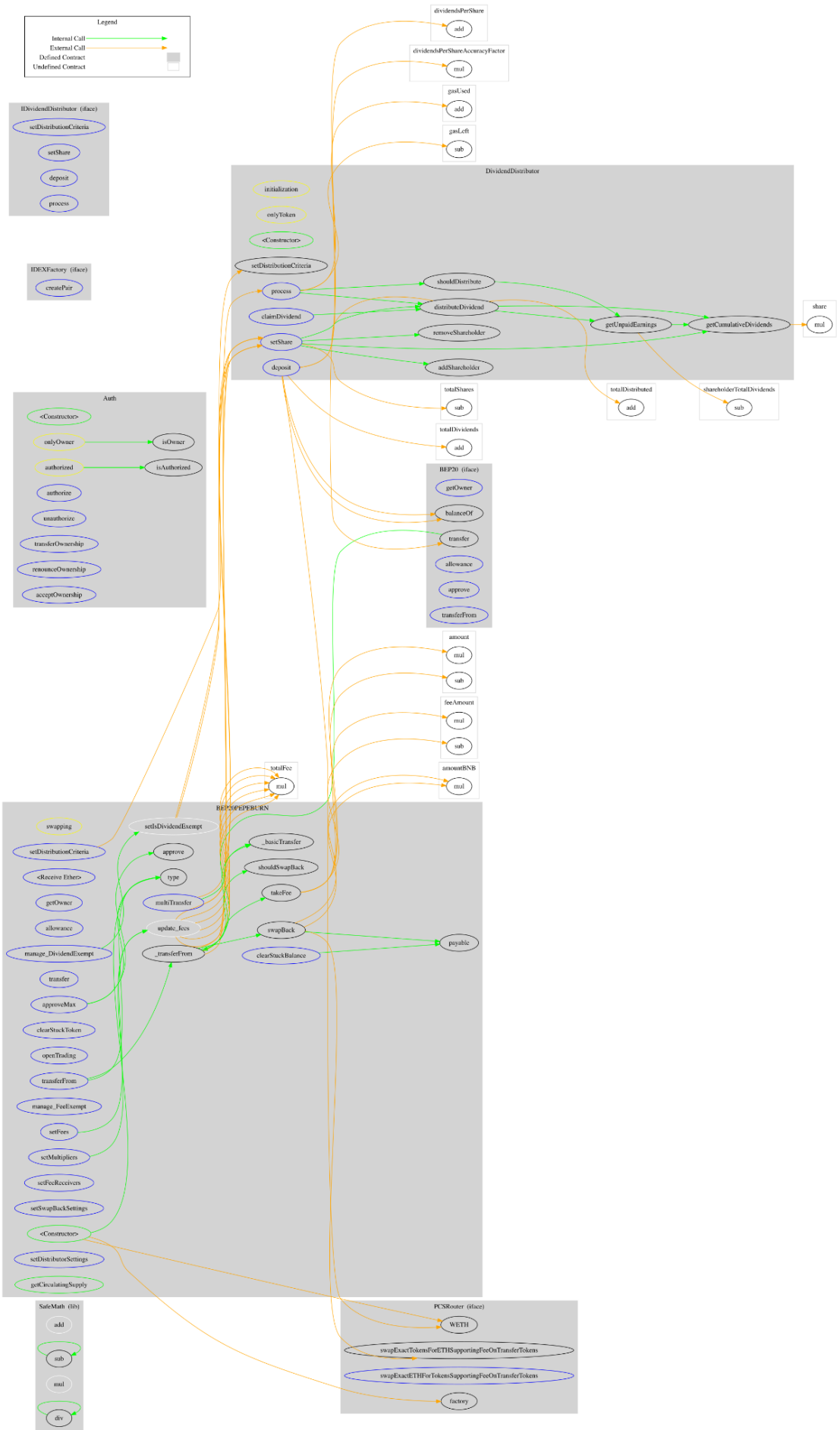
	unauthorize	External	✓	onlyOwner
	isOwner	Public		-
	isAuthorized	Public		-
	transferOwnership	External	✓	onlyOwner
	renounceOwnership	External	✓	onlyOwner
	acceptOwnership	External	✓	-
IDEXFactory	Interface			
	createPair	External	✓	-
PCSRouter	Interface			
	factory	External		-
	WETH	External		-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IDividendDistributor	Interface			
	setDistributionCriteria	External	✓	-
	setShare	External	✓	-
	deposit	External	Payable	-
	process	External	✓	-

DividendDistributor	Implementation	IDividendDistributor		
		Public	✓	-
	setDistributionCriteria	External	✓	onlyToken
	setShare	External	✓	onlyToken
	deposit	External	Payable	onlyToken
	process	External	✓	onlyToken
	shouldDistribute	Internal		
	distributeDividend	Internal	✓	
	claimDividend	External	✓	-
	getUnpaidEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
BEP20PEPEBURN	Implementation	BEP20, Auth		
		Public	✓	Auth
		External	Payable	-
	getOwner	External		-
	allowance	External		-
	approve	Public	✓	-
	approveMax	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-

	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	takeFee	Internal	✓	
	shouldSwapBack	Internal		
	clearStuckBalance	External	✓	onlyOwner
	clearStuckToken	External	✓	onlyOwner
	openTrading	External	✓	onlyOwner
	swapBack	Internal	✓	swapping
	setIsDividendExempt	Internal	✓	
	manage_DividendExempt	External	✓	authorized
	manage_FeeExempt	External	✓	authorized
	update_fees	Internal	✓	
	setMultipliers	External	✓	authorized
	setFees	External	✓	onlyOwner
	setFeeReceivers	External	✓	authorized
	setSwapBackSettings	External	✓	onlyOwner
	setDistributionCriteria	External	✓	authorized
	setDistributorSettings	External	✓	authorized
	getCirculatingSupply	Public		-
	multiTransfer	External	✓	onlyOwner

Inheritance Graph

Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IDividendDistributor (iface)
setDistributionCriteria
setShare
deposit
process
IDEXFactory (iface)
createPair
dividendsPerShare
add
dividendsPerShareAccuracyFactor
mul
gasUsed
add
gasLeft
sub
DividendDistributor
initialization
onlyToken
<Constructor>
setDistributionCriteria
process
claimDividend
setShare
deposit
shouldDistribute
distributeDividend
removeShareholder
addShareholder
getUnpaidEarnings
getCumulativeDividends
share
mul
totalDistributed
add
shareholderTotalDividends
sub
totalShares
sub
totalDividends
add
Auth
<Constructor>
onlyOwner
isOwner
authorized
isAuthorized
authorize
unauthorize
transferOwnership
renounceOwnership
acceptOwnership
BEP20 (iface)
getOwner
balanceOf
transfer
allowance
approve
transferFrom
amount
mul
sub
feeAmount
mul
sub
amountBNB
mul
totalFee
mul
BEP20PEPEBURN
swapping
setDistributionCriteria
<Receive Ether>
getOwner
allowance
manage_DividendExempt
transfer
approveMax
clearStuckToken
openTrading
transferFrom
manage_FeeExempt
setFees
setMultipliers
setFeeReceivers
setSwapBackSettings
<Constructor>
setDistributorSettings
getCirculatingSupply
setIsDividendExempt
approve
type
multiTransfer
update_fees
_transferFrom
_basicTransfer
shouldSwapBack
takeFee
swapBack
clearStuckBalance
payable
SafeMath (lib)
add
sub
mul
div
PCSRouter (iface)
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
factory

Summary

Pepe Burn 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 8% 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