Cyberscope

Audit Report

POWER PEPE

May 2023

Network BSC

Address 0xeb38fee868F46B5515a1D7377865478c0498c70d

Audited by © cyberscope

Table of Contents

Review

Contract Name	PPEPE
Compiler Version	v0.8.15+commit.e14f2714
Optimization	200 runs
Explorer	https://bscscan.com/address/0xeb38fee868f46b5515a1d7377865478c0498c70d
Address	0xeb38fee868f46b5515a1d7377865478c0498c70d
Network	BSC
Symbol	PPEPE
Decimals	9
Total Supply	1.000.000.000

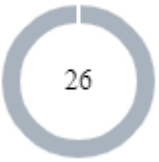
Audit Updates

Initial Audit	23 May 2023

Source Files

Filename	SHA256
PPEPE.sol	5e1700fdde3aa5e0b61873b8c3f874fda4af4943ca4191af3e65d594d32e0f27

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	26

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	26	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
●	DFEF	Duplicate Fee Exempt Functionality	Unresolved
●	RLM	Redundant Liquidity Mechanism	Unresolved
●	RC	Redundant Calculation	Unresolved
●	RMM	Redundant MultipliedFee Mechanism	Unresolved
●	TMO	Tax Mechanism Optimization	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	RLF	Redundant Launched Functionality	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RV	Redundant Variables	Unresolved
●	MSE	Missing Solidity Events	Unresolved
●	MSC	Missing Sanity Check	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RTT	Redundant Transfer Tax	Unresolved

●	RBF	Redundant Burn Functionality	Unresolved
●	RCO	Redundant Conditional Operations	Unresolved
●	RDM	Revert Descriptive Message	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
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

DFEF - Duplicate Fee Exempt Functionality

Criticality	Minor / Informative
Location	PPEPE.sol#L709,713
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 utilized two identical functions. The functions `shouldTakeFee` and `shouldTakeFeer` return the same result. Hence, the functions are identical.

```
function shouldTakeFee(address sender) internal view returns (bool) {
    return !isFeeExempt[sender];
}

function shouldTakeFeer(address recipient) internal view returns (bool) {
    return !isFeeExempt[recipient];
}
```

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. It is recommended to merge the two functions.

RLM - Redundant Liquidity Mechanism

Criticality	Minor / Informative
Location	PPEPE.sol#L785,791,797,855
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 liquidity functionality. The variables `liquiditySellFee` and `liquidityBuyFee` are initialized to zero and cannot be modified. Hence, the code segment below will never be executed. As a result, the liquidity functionality and by extension the variables `liquiditySellFee` and `liquidityBuyFee` are redundant.

```
uint256 liquidityFee = selling ? liquiditySellFee : liquidityBuyFee;

uint256 dynamicLiquidityFee = isOverLiquified(
    targetLiquidity,
    targetLiquidityDenominator
)
    ? 0
    : liquidityFee;

uint256 amountToLiquify = balanceOf(address(this))
    .mul(dynamicLiquidityFee)
    .div(totalFee)
    .div(2);

if (amountToLiquify > 0) {
    router.addLiquidityETH{value: amountBNBLiquidity}(
        address(this),
        amountToLiquify,
        0,
        0,
        autoLiquidityReceiver,
        block.timestamp
    );
    emit AutoLiquify(amountBNBLiquidity, amountToLiquify);
}
```

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. It is recommended to remove redundant variables and code segments.

RC - Redundant Calculation

Criticality	Minor / Informative
Location	PPEPE.sol#L565,566
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 performs multiplication and then division with the same value. Hence, the calculation is redundant.

```
_balances[ts_Project] = ((_totalSupply * 100) / 100);
emit Transfer(address(0), ts_Project, ((_totalSupply * 100) / 100));
```

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. It is recommended to remove redundant calculations.

RMM - Redundant MultipliedFee Mechanism

Criticality	Minor / Informative
Location	PPEPE.sol#L731
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 MultipliedFee functionality. The `getMultipliedFee` method always returns the same variable since the multiplier `10000` is equal to the `sellFeeDenominator` variable. As a result, the MultipliedFee functionality and by extension the variables launchedAtTimestamp are redundant.

```
function getMultipliedFee() public view returns (uint256) {
    if (launchedAtTimestamp + 1 days > block.timestamp) {
        return totalSellFee.mul(10000).div(sellFeeDenominator);
    }

    return totalSellFee;
}
```

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. It is recommended to remove redundant variables and functionality.

TMO - Tax Mechanism Optimization

Criticality	Minor / Informative
Location	PPEPE.sol#L497,504,511
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.

Since the buy and sell taxes are immutable and they have the same values. They could be merged so it becomes a smaller size and consumes less memory. Additionally, the fee denominator for all fees is identical too.

```
uint256 liquidityBuyFee = 0;
uint256 marketingBuyFee = 100;
uint256 projectBuyFee = 100;
uint256 totalBuyFee = 600;
uint256 buyFeeDenominator = 10000;
uint256 buyfeeburn = 0;

uint256 liquiditySellFee = 0;
uint256 marketingSellFee = 100;
uint256 projectSellFee = 100;
uint256 totalSellFee = 600;
uint256 sellFeeDenominator = 10000;
uint256 sellfeeburn = 0;

...
uint256 TransferFeeDenominator = 10000;
```

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. It is recommended to merge buy and sell taxes and one `FeeDenominator` could be used instead of three.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	PPEPE.sol#L832,843
Status	Unresolved

Description

The contract sends funds to a `marketingFeeReceiver` and `projectFeeReceiver` 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.

```
if (marketingFee > 0) {
    ...
    (bool success, ) = payable(marketingFeeReceiver).call{
        value: amountBNBMarketing,
        gas: 90000
    }("");
    require(success, "receiver rejected ETH transfer");
}

if (projectFee > 0) {
    ...
    (bool success2, ) = payable(projectFeeReceiver).call{
        value: amountBNBproject,
        gas: 90000
    }("");
    require(success2, "receiver rejected ETH transfer");
}
```

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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	PPEPE.sol#L825,832,843
Status	Unresolved

Description

Division of decimal (fixed point) numbers can result in rounding errors due to the way that division is implemented in Solidity. Thus, it may produce issues with precise calculations with decimal numbers.

Solidity represents decimal numbers as integers, with the decimal point implied by the number of decimal places specified in the type (e.g. decimal with 18 decimal places). When a division is performed with decimal numbers, the result is also represented as an integer, with the decimal point implied by the number of decimal places in the type. This can lead to rounding errors, as the result may not be able to be accurately represented as an integer with the specified number of decimal places.

Hence, the splitted shares will not have the exact precision and some funds may not be calculated as expected.

The variable `amountBNB` will not be splitted as expected.

```
if (reflectionFee > 0) {
    uint256 amountBNBReflection = amountBNB.mul(reflectionFee).div(
        totalBNBFee
    );
    ...
}

if (marketingFee > 0) {
    uint256 amountBNBMarketing = amountBNB.mul(marketingFee).div(
        totalBNBFee
    );
    ...
}

if (projectFee > 0) {
    uint256 amountBNBproject = amountBNB.mul(projectFee).div(
        totalBNBFee
    );
    ...
}
```

Recommendation

The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

RLF - Redundant Launched Functionality

Criticality	Minor / Informative
Location	PPEPE.sol#L531,878
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 launched functionality. The variable `launchedAt` is initialized to the deployed `block.number`. Hence, the launched method will always return true. As a result, launched functionality and by extension the variables `launchedAt` are redundant.

```
uint256 public launchedAt = block.number;

function launched() internal view returns (bool) {
    return launchedAt != 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. It is recommended to remove redundant variables and functionality.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	PPEPE.sol#L895,906,914
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 updates variables even if its current state is the same as the one passed as an argument. As a result, the contract performs redundant storage writes.

```
function setFeeReceivers(
    address _autoLiquidityReceiver,
    address _marketingFeeReceiver,
    address _projectFeeReceiver
) external onlyOwner {
    autoLiquidityReceiver = _autoLiquidityReceiver;
    marketingFeeReceiver = _marketingFeeReceiver;
    projectFeeReceiver = _projectFeeReceiver;
}

function setSwapBackSettings(bool _enabled, uint256 _amount)
    external
    onlyOwner
{
    swapEnabled = _enabled;
    swapThreshold = _amount;
}

function setTargetLiquidity(uint256 _target, uint256 _denominator)
    external
    onlyOwner
{
    targetLiquidity = _target;
    targetLiquidityDenominator = _denominator;
}
```

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.

RV - Redundant Variables

Criticality	Minor / Informative
Location	PPEPE.sol#L511,512,513
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 variables `liquidityTransferFee` , `marketingTransferFee` , and `projectTransferFee` are not utilized in the contract implementation. Hence, they are redundant.

```
uint256 liquidityTransferFee = 0;
uint256 marketingTransferFee = 0;
uint256 projectTransferFee = 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. It is recommended to remove redundant variables.

MSE - Missing Solidity Events

Criticality	Minor / Informative
Location	PPEPE.sol#L895
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

The method `setFeeReceivers` does not emit an event when the fee receivers addresses mutate.

```
function setFeeReceivers(
    address _autoLiquidityReceiver,
    address _marketingFeeReceiver,
    address _projectFeeReceiver
) external onlyOwner {
    autoLiquidityReceiver = _autoLiquidityReceiver;
    marketingFeeReceiver = _marketingFeeReceiver;
    projectFeeReceiver = _projectFeeReceiver;
}
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

MSC - Missing Sanity Check

Criticality	Minor / Informative
Location	PPEPE.sol#L
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 `targetLiquidity` nominator could exceed the denominator.

```
function setTargetLiquidity(uint256 _target, uint256 _denominator)
    external
    onlyOwner
{
    targetLiquidity = _target;
    targetLiquidityDenominator = _denominator;
}
```

Recommendation

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

The `targetLiquidity` variable should be lower than or equal to `targetLiquidityDenominator` variable.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	PPEPE.sol#L776
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapThreshold` 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.

```
function shouldSwapBack() internal view returns (bool) {
    return
        msg.sender != pair &&
        !inSwap &&
        swapEnabled &&
        _balances[address(this)] >= swapThreshold;
}
```

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 total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

RTT - Redundant Transfer Tax

Criticality	Minor / Informative
Location	PPEPE.sol#L748
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 transfer tax functionality. The variable `totalTransferFee` is initialized to zero and cannot be modified. Hence, the code segment will never execute with zero `feeAmount` every time. As a result, transfer tax functionality and by extension the variables totalTransferFee and TransferFeeDenominator are redundant.

```
if (!isSenderPair && !isReceiverPair) {
    uint256 feeAmount = amount.mul(totalTransferFee).div(
        TransferFeeDenominator
    );
    _balances[address(this)] = _balances[address(this)].add(feeAmount);
    emit Transfer(sender, address(this), feeAmount);
    return amount.sub(feeAmount);
}
```

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. It is recommended to remove redundant variables and code segments.

RBF - Redundant Burn Functionality

Criticality	Minor / Informative
Location	PPEPE.sol#L762
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 burn functionality. The variables `sellfeeburn` and `buyfeeburn` are initialized to zero and cannot be modified. Hence, the code segment below will execute with zero `amounttoburn` every time. As a result, the burn functionality and by extension the variables `sellfeeburn` and `buyfeeburn` are redundant.

```
uint256 feetoburn = receiver == pair ? sellfeeburn : buyfeeburn;
uint256 amounttoburn = amount.mul(feetoburn).div(feeDenominator);

...

_balances[DEAD] = _balances[DEAD].add(amounttoburn);
emit Transfer(sender, DEAD, amounttoburn);
```

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. It is recommended to remove redundant variables and code segments.

RCO - Redundant Conditional Operations

Criticality	Minor / Informative
Location	PPEPE.sol#L690,739,825,832,843
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 performs an if conditional without undertaking any operations when the if statement is fulfilled. Hence, the if statement is redundant.

```
if (sender != pair && !isOwner(sender)) {}
```

The contract performs the conditional statement `receiver == pair` multiple times. Hence, it is redundant.

```
function takeFee(
    address sender,
    address receiver,
    uint256 amount
) internal returns (uint256) {
    // Verificar se o sender ou o receiver são o par
    bool isSenderPair = sender == pair;
    bool isReceiverPair = receiver == pair;
    ...
        uint256 feeDenominator = receiver == pair
            ? sellFeeDenominator
            : buyFeeDenominator;
        uint256 feeAmount = amount.mul(getTotalFee(receiver == pair)).div(
            feeDenominator
        );
        uint256 feetoburn = receiver == pair ? sellfeeburn : buyfeeburn;
    ...
}
```

The contract performs conditional operations that are always true. The variables `reflectionFee` , `marketingFee` , and `projectFee` are set to values greater than zero and they are immutable. Hence, they are redundant.

```
if (reflectionFee > 0) {...}

if (marketingFee > 0) {...}

if (projectFee > 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.

- The contract could reuse the variable `isReceiverPair` .
- The contract could remove redundant conditional statements.

RDM - Revert Descriptive Message

Criticality	Minor / Informative
Location	PPEPE.sol#L311,317,661,886,947
Status	Unresolved

Description

The `revert()` function is used to halt the execution of a contract and revert any changes made to the contract's state. The contract does not provide a descriptive message to the `revert()` function.

```
require(!initialized);
require(msg.sender == _token);
require(_balances[sender] > 0);
require(holder != address(this) && holder != pair);
require(gas < 750000);
```

Recommendation

The team is suggested to provide a descriptive message to the `revert()` function. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	PPEPE.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	PPEPE.sol#L544,545,548
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	PPEPE.sol#L285,298,477,478,479,480,485,493,494,495,496,497,498,500,501,502,503,504,505,507,508,509,510,511,517,518,525
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 WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
uint256 public dividendsPerShareAccuracyFactor = 10**36
address USDT = 0x55d398326f99059fF775485246999027B3197955
address DEAD = 0x000000000000000000000000000000000000dEaD
address ZERO = 0x0000000000000000000000000000000000000000
uint256 _totalSupply = 1000000000 * (10**_decimals)
uint256 liquidityBuyFee = 0
uint256 marketingBuyFee = 100
uint256 projectBuyFee = 100
uint256 totalBuyFee = 600
uint256 buyFeeDenominator = 10000
uint256 buyfeeburn = 0
uint256 liquiditySellFee = 0
uint256 marketingSellFee = 100

...
```

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	PPEPE.sol#L201,276,284,285,325,326,477,478,479,480,482,483,484,485,487,488,511,525,892,893,894,901,909,924,925
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);
address _token
IBEP20 public USDT = IBEP20(0x55d398326f99059fF775485246999027B3197955)
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
uint256 _minPeriod
uint256 _minDistribution
address USDT = 0x55d398326f99059fF775485246999027B3197955
address DEAD = 0x000000000000000000000000000000000000dEaD
address ZERO = 0x0000000000000000000000000000000000000000
string constant _name = "POWER PEPE"
string constant _symbol = "PPEPE"
uint8 constant _decimals = 9
uint256 _totalSupply = 1000000000 * (10**_decimals)
mapping(address => uint256) _balances

...
```

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	PPEPE.sol#L477,507,508,509
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 USDT = 0x55d398326f99059fF775485246999027B3197955
uint256 liquidityTransferFee = 0
uint256 marketingTransferFee = 0
uint256 projectTransferFee = 0
```

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	PPEPE.sol#L328,913
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
argetLiquidity = _target;
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	PPEPE.sol#L864
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.

```
unction buyTokens(uint256 amount, address to) internal swapping {
        address[] memory path = new address[](2);
        path[0] = WBNB;
        path[1] = address(this);

        router.swapExactETHForTokensSupportingFeeOnTransferTokens{
            value: amount
        }(0, path, to, block.timestamp);
    }
```

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	PPEPE.sol#L184,896,897,898
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.

```
owner = adr
utoLiquidityReceiver = _autoLiquidityReceiver;
arketingFeeReceiver = _marketingFeeReceiver;
rojectFeeReceiver = _projectFeeReceiver;
```

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	PPEPE.sol#L410
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(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**
IERC165	Interface			
	supportsInterface	External		-
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		

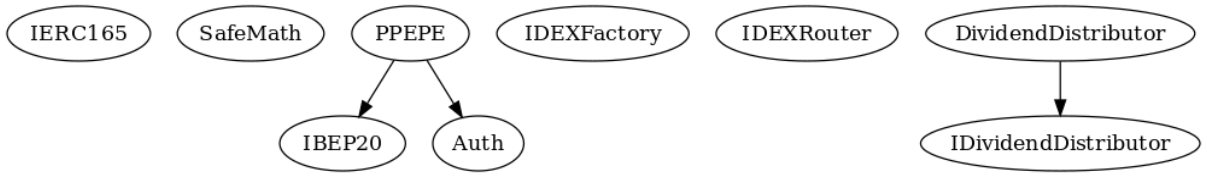
IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	burn	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Auth	Implementation			
		Public	✓	-
	isOwner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IDEXFactory	Interface			
	createPair	External	✓	-
IDEXRouter	Interface			

	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	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	✓	

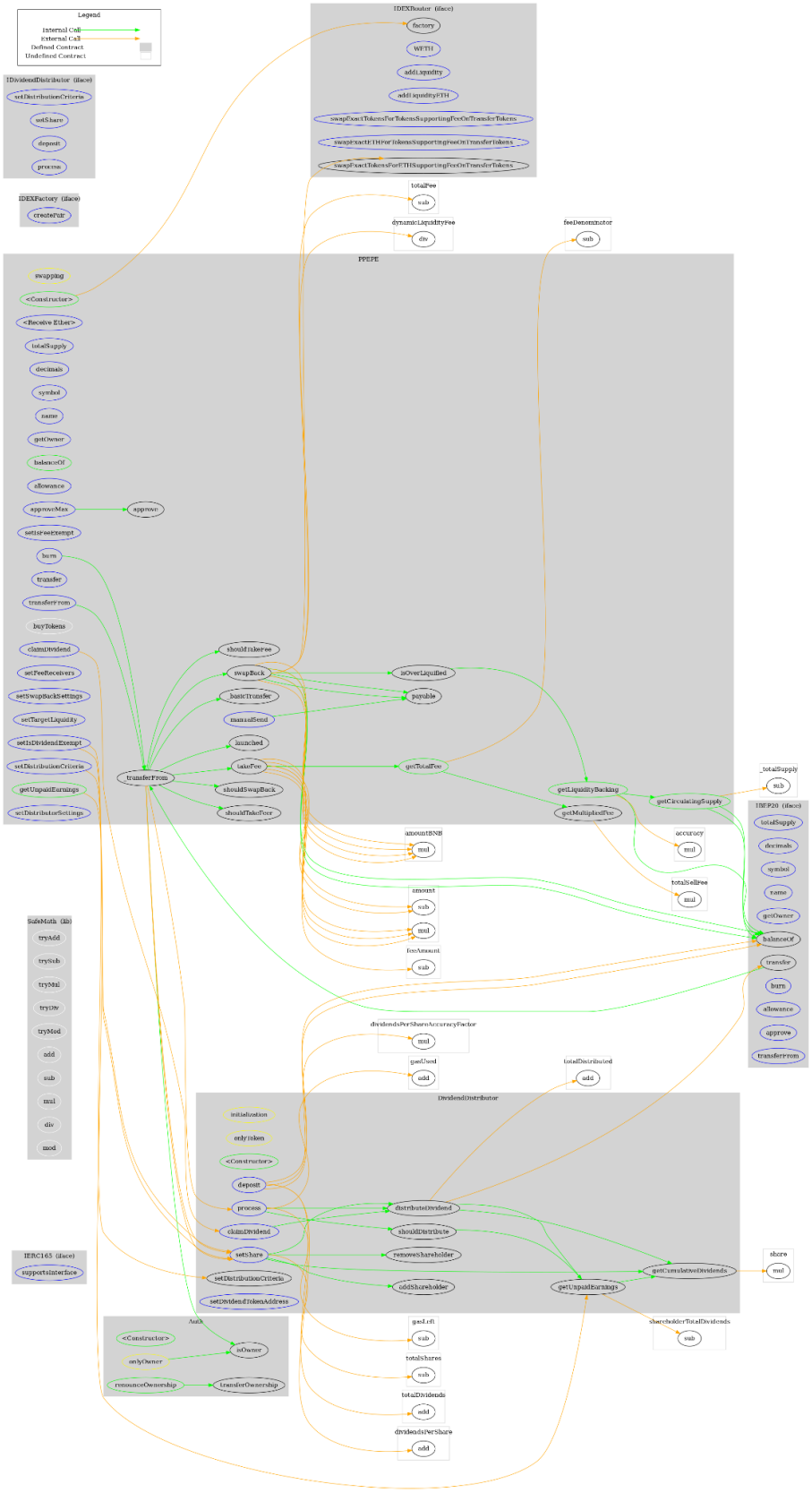
	claimDividend	External	✓	onlyToken
	getUnpaidEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
	setDividendTokenAddress	External	✓	onlyToken
PPEPE	Implementation	IBEP20, Auth		
		Public	✓	Auth
		External	Payable	-
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	Public		-
	allowance	External		-
	approve	Public	✓	-
	approveMax	External	✓	-
	setIsFeeExempt	External	✓	onlyOwner
	burn	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-

	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	shouldTakeFee	Internal		
	shouldTakeFeer	Internal		
	getTotalFee	Public		-
	getMultipliedFee	Public		-
	takeFee	Internal	✓	
	shouldSwapBack	Internal		
	swapBack	Internal	✓	swapping
	buyTokens	Internal	✓	swapping
	launched	Internal		
	setIsDividendExempt	External	✓	onlyOwner
	setFeeReceivers	External	✓	onlyOwner
	setSwapBackSettings	External	✓	onlyOwner
	setTargetLiquidity	External	✓	onlyOwner
	manualSend	External	✓	-
	setDistributionCriteria	External	✓	onlyOwner
	claimDividend	External	✓	-
	getUnpaidEarnings	Public		-
	setDistributorSettings	External	✓	onlyOwner
	getCirculatingSupply	Public		-
	getLiquidityBacking	Public		-
	isOverLiquified	Public		-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IDividendDistributor (iface)
setDistributionCriteria
setShare
deposit
process
IDEXFactory (iface)
createPair
IDEXRouter (iface)
factory
WFTH
addLiquidity
addLiquidityETH
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForETHSupportingFeeOnTransferTokens
totalFee
sub
dynamicLiquidityFee
div
feeDenominator
sub
PPEPE
swapping
<Constructor>
<Receive Ether>
totalSupply
decimals
symbol
name
getOwner
balanceOf
allowance
approveMax
approve
setIsFeeExempt
burn
transfer
transferFrom
buyTokens
claimDividend
setFeeReceivers
setSwapBackSettings
setTargetLiquidity
setIsDividendExempt
setDistributionCriteria
getUnpaidEarnings
setDistributorSettings
shouldTakeFee
swapBack
basicTransfer
manualSend
launched
takeFee
shouldSwapBack
shouldTakeFee
isOverLiquified
payable
getTotalFee
getLiquidityBacking
getMultipliedFee
getCirculatingSupply
_totalSupply
sub
IBEP20 (iface)
totalSupply
decimals
symbol
name
getOwner
balanceOf
transfer
burn
allowance
approve
transferFrom
amountBNB
mul
accuracy
mul
totalSellFee
mul
amount
sub
mul
feeAmount
sub
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
dividendsPerShareAccuracyFactor
mul
gasUsed
add
totalDistributed
add
DividendDistributor
initialization
onlyToken
<Constructor>
deposit
process
claimDividend
setShare
setDistributionCriteria
setDividendTokenAddress
distributeDividend
shouldDistribute
removeShareholder
addShareholder
getCumulativeDividends
getUnpaidEarnings
share
mul
shareholderTotalDividends
sub
gasLeft
sub
totalShares
sub
totalDividends
add
dividendsPerShare
add
IERC165 (iface)
supportsInterface
Auth
<Constructor>
isOwner
onlyOwner
renounceOwnership
transferOwnership

Summary

POWER PEPE contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. POWER 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. There is also a limit of max 6% fee. Additionally, the contract has a launch tax of 99% up to the second block after the contract launches.

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