Cyberscope

Audit Report

Payinnov Protocol

Aug 2023

Network	BSC
Address	0xbf5b885808E34174eDd178CFA455Ecdb1c6B2656
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	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
●	DPI	Decimals Precision Inconsistency	Unresolved
●	MRM	Missing Revert Messages	Unresolved
●	ULTW	Transfers Liquidity to Team Wallet	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RTCI	Reward Token Change Inconsistency	Unresolved
●	MC	Missing Check	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	GO	Gas Optimization	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	MEE	Missing Events Emission	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

●	L07	Missing Events Arithmetic	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	Token
Compiler Version	v0.8.16+commit.07a7930e
Optimization	200 runs
Explorer	https://bscscan.com/address/0xbf5b885808e34174edd178cfa455ecdb1c6b2656
Address	0xbf5b885808e34174edd178cfa455ecdb1c6b2656
Network	BSC
Symbol	PNV
Decimals	18
Total Supply	10,000,000,000

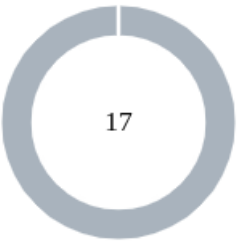
Audit Updates

Initial Audit	02 Aug 2023

Source Files

Filename	SHA256
Token.sol	91b3b55a7a1205db40376d45593577d64b02a26691b1e30e80480649c959c5dd

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	17

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	17	0	0	0

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	Token.sol#L629
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.

```
uint256 amountBNB = address(this).balance.sub(balanceBefore);
uint256 totalBNBFee = totalFee.sub(dynamicLiquidityFee.div(2));
uint256 amountBNBLiquidity =
amountBNB.mul(dynamicLiquidityFee).div(totalBNBFee).div(2);
uint256 amountBNBReflection =
amountBNB.mul(reflectionFee).div(totalBNBFee);
uint256 amountBNBStaking = amountBNB.mul(stakingFee).div(totalBNBFee);
uint256 amountBNBMarketing =
amountBNB.mul(marketingFee).div(totalBNBFee);
```

Recommendation

To avoid these issues, the contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue. It is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to

scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

ERC20	Decimals
Token 1	6
Token 2	9
Token 3	18

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

MRM - Missing Revert Messages

Criticality	Minor / Informative
Location	Token.sol#L203,209,676,736
Status	Unresolved

Description

The contract is missing error messages. These missing error messages are making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(!initialized);
require(msg.sender == _token); _;
require(holder != address(this) && holder != pair);
require(gas < 750000);
```

Recommendation

The team is advised to carefully review the source code in order to address these issues. To accelerate the debugging process and mitigate these issues, the team should use more specific and descriptive error messages.

ULTW - Transfers Liquidity to Team Wallet

Criticality	Minor / Informative
Location	Token.sol#L610
Status	Unresolved

Description

The contract owner has the authority to set the `totalFee` to zero. As a result, the function `swapBack` will swap the contract's tokens to the native currency and send the swapped amount to the `marketingFeeReceiver` .

```
function swapBack() internal swapping {
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = WBNB;
        uint256 balanceBefore = address(this).balance;

        if (totalFee > 0) {
        ...
        } else if (balanceOf(address(this)) > 0) {
            router.swapExactTokensForETHSupportingFeeOnTransferTokens(
                balanceOf(address(this)),
                0,
                path,
                address(this),
                block.timestamp
            );
            uint256 amountBNB =
address(this).balance.sub(balanceBefore);
            (bool success,) = payable(marketingFeeReceiver).call{value:
amountBNB, gas: 30000}("");
            require(success, "Failed to send BNB to marketing fee
receiver");
        }
    }
```

Recommendation

The contract could embody a check for the maximum amount of funds that can be swapped, since a huge amount may volatile the token's price. 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 time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Token.sol#L587,624,708,722,729,735,743
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 variable 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 setIsFeeExempt(address holder, bool exempt) external
onlyOwner {
        isFeeExempt[holder] = exempt;
    }


    function setIsDividendExempt(address holder, bool exempt) external
onlyOwner {
        require(holder != address(this) && holder != pair);
        isDividendExempt[holder] = exempt;
        if(exempt){
            distributor.setShare(holder, 0);
        }else{
            distributor.setShare(holder, _balances[holder]);
        }
    }

    function changeWallets(address _marketingReceiver, address
_stakingReceiver, address _autoLiquidityReceiver) external isAutorizer {
        require(_marketingReceiver != address(0), "ERC20: transfer from
the zero address");
        require(_stakingReceiver != address(0), "ERC20: transfer to the
zero address");
        require(_autoLiquidityReceiver != address(0), "ERC20: transfer
to the zero address");
        stakingReceiver = _stakingReceiver;
        marketingFeeReceiver = _marketingReceiver;
        autoLiquidityReceiver = _autoLiquidityReceiver;
    }

    function setAutorizerAddress(address _autorizer, bool yesno)
external isAutorizer {
        isAutorizerAddress[_autorizer] = yesno;
    }

        REWARD = _rewardTokenAddress;
        distributor.setRewardToken(_rewardTokenAddress);
    }
```

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

        emit
TargetLiquidityUpdated(targetLiquidity,targetLiquidityDenominator);
    }

    function setDistributionCriteria(uint256 _minPeriod, uint256
_minDistribution) external onlyOwner {
        distributor.setDistributionCriteria(_minPeriod,
_minDistribution);

        emit DistributionCriteriaUpdated(_minPeriod, _minDistribution);
    }

    function setDistributorSettings(uint256 gas) external onlyOwner {
        require(gas < 750000);
        distributorGas = gas;
    }
```

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.

RTCI - Reward Token Change Inconsistency

Criticality	Minor / Informative
Location	Token.sol#L473
Status	Unresolved

Description

The contract owner has the authority to change the distribution token address by calling the `setRewardToken()` method. This may produce several issues in the distribution process.

1. There should be a valid pair address between the ETH and the new token address.
2. The internal state of the distributor should reset since variables like `dividendsPerShare` are based on the previous token's balance.

```
    function setRewardToken(address _rewardTokenAddress) external
onlyOwner {
        require(
            _rewardTokenAddress != DEAD &&
            _rewardTokenAddress != pair &&
            _rewardTokenAddress != owner,
            "Invalid reward token address"
        );

        REWARD = _rewardTokenAddress;
        distributor.setRewardToken(_rewardTokenAddress);
    }
```

Recommendation

The team is advised to either remove the option of changing the reward address or resolve the side-effects of the change.

MC - Missing Check

Criticality	Minor / Informative
Location	Token.sol#L724,735
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.

Specifically, in the `setTargetLiquidity` function, there is no validation to ensure that the `_denominator`, which sets the `targetLiquidityDenominator` is greater than zero.

Additionaly the `setDistributorSettings` function allows the owner to set the `distributorGas` value. While there is a check in place to ensure that the provided `gas` value is less than 750,000, there is no validation to ensure that the gas is above a certain minimum threshold. Setting the `distributorGas` to a value that's too low could lead to failed transactions or other unintended behaviors, especially if the gas provided is insufficient for the operations that rely on this setting.

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

        emit
TargetLiquidityUpdated(targetLiquidity,targetLiquidityDenominator);
    }

    function setDistributorSettings(uint256 gas) external onlyOwner {
        require(gas < 750000);
        distributorGas = gas;
    }
```

Recommendation

The team is advised to properly check the variables according to the required specifications. It is recommended to include a `require` statement within the `setTargetLiquidity` function to validate that the `_denominator` is greater than zero. Additionaly the team is advised to enhance the require statement within the `setDistributorSettings` function to also validate that the gas value is greater than a fixed minimum value. This will ensure that the `distributorGas` is set within a safe and operational range.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Token.sol#L717
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.

Additionaly, when the `swapEnabled` variable is set to false, the contract will continue to accumulate tokens without triggering the swap. This can result in the contract holding a vast amount of tokens over time. Once `swapEnabled` is set to true, the contract will attempt to swap all the accumulated tokens in one transaction, up to the limit defined by `swapThreshold`.

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 setSwapBackSettings(bool _enabled, uint256 _amount) external
onlyOwner {
        swapEnabled = _enabled;
        swapThreshold = _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 total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

Similarly, ensure that the minimum amount to trigger a swap should be larger than a fixed percentage of the total supply, preventing frequent small swaps that could be inefficient.

GO - Gas Optimization

Criticality	Minor / Informative
Location	Token.sol#L717
Status	Unresolved

Description

Gas optimization refers to the process of reducing the amount of gas required to execute a transaction. Gas is the unit of measurement used to calculate the fees paid to miners for including a transaction in a block on the blockchain.

The contract variable `swapThreshold` is used to set a threshold where the contract will trigger the swap functionality. Since there is no input validation at the `setSwapBackSettings` function, the `swapThreshold` could be set to any amount such as zero. As a result, the contract will execute the swap functionality every time a transaction is taking place and consume more gas.

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

Recommendation

The contract could embody a check for not allowing setting the swapThreshold 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 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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	Token.sol#L637,646,670
Status	Unresolved

Description

The contract sends funds to the `stakingReceiver` , `marketingFeeReceiver` and `marketingFeeReceiver` addresses as part of the transfer flow. These addresses 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 (stakingFee > 0) {
    (bool success,) = payable(stakingReceiver).call{value:
amountBNBStaking, gas: 30000}("");
    require(success, "Failed to send BNB to staking receiver");
}
...

if (marketingFee > 0) {
    (bool success,) = payable(marketingFeeReceiver).call{value:
amountBNBMarketing, gas: 30000}("");
    require(success, "Failed to send BNB to marketing fee receiver");
}
...

uint256 amountBNB = address(this).balance.sub(balanceBefore);
(bool success,) = payable(marketingFeeReceiver).call{value: amountBNB,
gas: 30000}("");
require(success, "Failed to send BNB to marketing fee receiver");
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	Token.sol#L241,591,675,717,735
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.

```
   function setRewardToken(address _rewardTokenAddress) external
onlyOwner {
        require(
            _rewardTokenAddress != DEAD &&
            _rewardTokenAddress != pair &&
            _rewardTokenAddress != owner,
            "Invalid reward token address"
        );

        REWARD = _rewardTokenAddress;
        distributor.setRewardToken(_rewardTokenAddress);
    }

    function changeLpPair(address newPair,bool yesno) external
isAutorizer {
        isLpPair[newPair] = yesno;
    }

    function setIsDividendExempt(address holder, bool exempt) external
onlyOwner {
        require(holder != address(this) && holder != pair);
        isDividendExempt[holder] = exempt;
        if(exempt){
            distributor.setShare(holder, 0);
        }else{
            distributor.setShare(holder, _balances[holder]);
        }
    }
    function setSwapBackSettings(bool _enabled, uint256 _amount)
external onlyOwner {
        swapEnabled = _enabled;
        swapThreshold = _amount;
    }

    function setDistributorSettings(uint256 gas) external onlyOwner {
        require(gas < 750000);
        distributorGas = gas;
    }
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Token.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	Token.sol#L410,411
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.

```
router
pair
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Token.sol#L188,347,353
Status	Unresolved

Description

State variables can be declared as constant using the constant keyword. This means that the value of the state variable cannot be changed after it has been set. Additionally, the constant variables decrease gas consumption of the corresponding transaction.

```
uint256 public dividendsPerShareAccuracyFactor = 10 ** 36
address public MKT = 0x1cAcb4979dACAEF191b9A67341e739ce6997dEF5
uint256 _totalSupply = 10_000_000_000 * (10**_decimals)
```

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	Token.sol#L110,166,174,175,215,236,240,343,347,353,468,565,570,589,594,599,680,689,703,712,717,724,738
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
address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c
IBEP20 REWARD = IBEP20(WBNB)
uint256 _minDistribution
uint256 _minPeriod
address _address
address public REWARD = 0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56
address public MKT = 0x1cAcb4979dACAEF191b9A67341e739ce6997dEF5
uint256 _totalSupply = 10_000_000_000 * (10**_decimals)
address _rewardTokenAddress

...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Token.sol#L216
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
```

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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Token.sol#L641,665
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
bool success
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Token.sol#L302,601
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.

```
REWARD.transfer(shareholder, amount)
IBEP20(_tokenAddress).transfer(address(msg.sender), _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**
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
IBEP20	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-

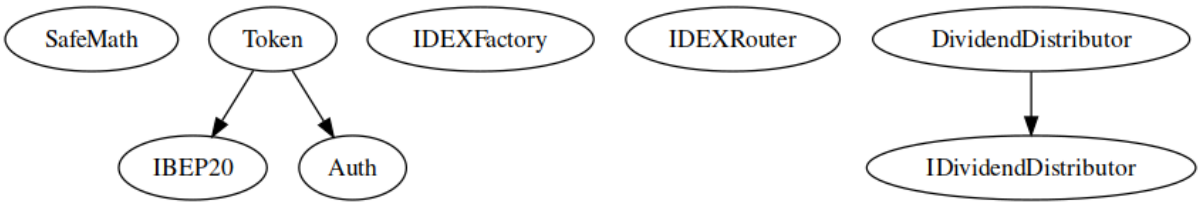
	transferFrom	External	✓	-
Auth	Implementation			
		Public	✓	-
	isOwner	Public		-
	transferOwnership	Public	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner
IDEXFactory	Interface			
	createPair	External	✓	-
IDEXRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupporting FeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFee OnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFee OnTransferTokens	External	✓	-
IDividendDistri butor	Interface			
	setDistributionCriteria	External	✓	-

	setShare	External	✓	-
	deposit	External	Payable	-
	process	External	✓	-
DividendDistributor	Implementation	IDividendDistributor		
		Public	✓	-
	setDistributionCriteria	External	✓	onlyToken
	setShare	External	✓	onlyToken
	setRewardToken	External	✓	onlyToken
	setWBNB	External	✓	onlyToken
	deposit	External	Payable	onlyToken
	process	External	✓	onlyToken
	shouldDistribute	Internal		
	distributeDividend	Internal	✓	
	claimDividend	External	✓	-
	getUnpaidEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
Token	Implementation	IBEP20, Auth		
		Public	✓	Auth
		External	Payable	-

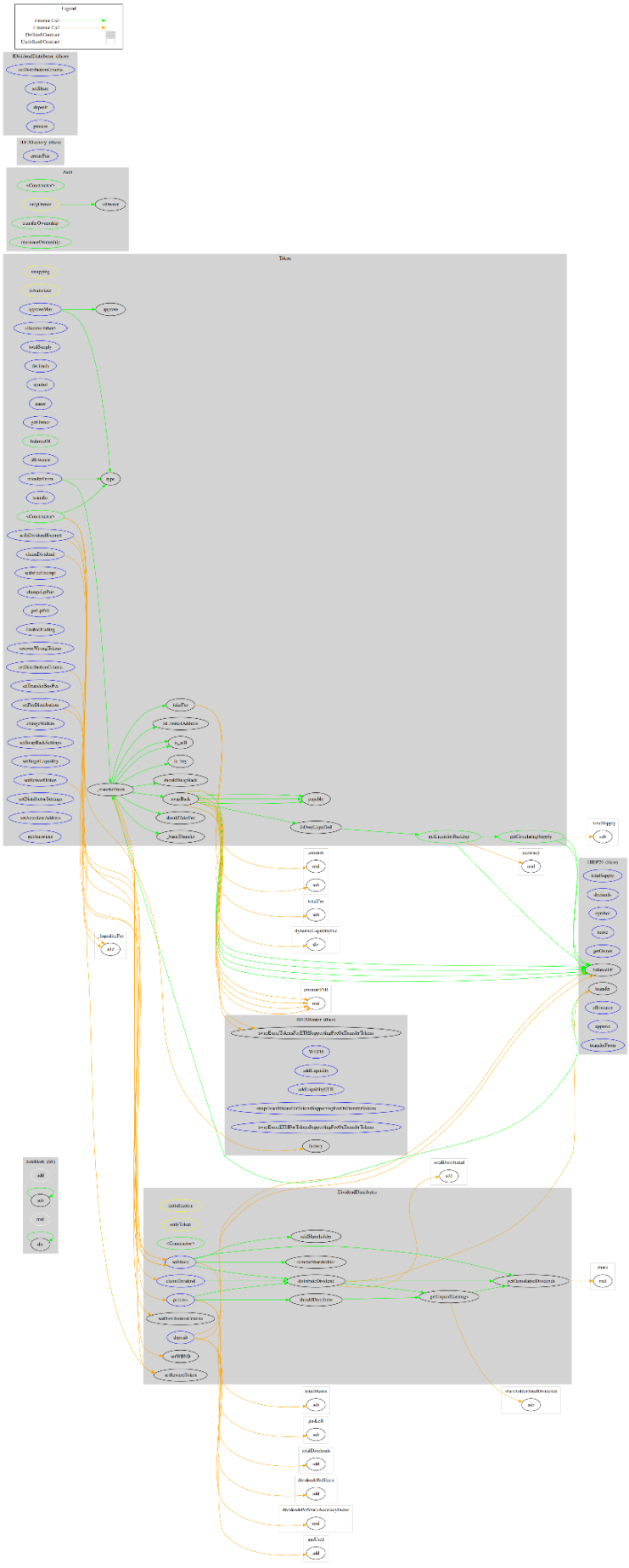
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	getOwner	External		-
	balanceOf	Public		-
	allowance	External		-
	approve	Public	✓	-
	approveMax	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	setRewardToken	External	✓	onlyOwner
	claimDividend	External	✓	-
	_transferFrom	Internal	✓	
	_basicTransfer	Internal	✓	
	shouldTakeFee	Internal		
	takeFee	Internal	✓	
	shouldSwapBack	Internal		
	is_buy	Internal		
	is_sell	Internal		
	isLimitedAddress	Internal		
	setIsFeeExempt	External	✓	onlyOwner
	changeLpPair	External	✓	isAutorizer

	getLpPair	External		-
	EnableTrading	External	✓	onlyOwner
	recoverWrongTokens	External	✓	isAutorizer
	swapBack	Internal	✓	swapping
	setIsDividendExempt	External	✓	onlyOwner
	setTransferBuyFee	External	✓	onlyOwner
	setFeeDistribution	External	✓	onlyOwner
	changeWallets	External	✓	isAutorizer
	setSwapBackSettings	External	✓	onlyOwner
	setTargetLiquidity	External	✓	onlyOwner
	setDistributionCriteria	External	✓	onlyOwner
	setDistributorSettings	External	✓	onlyOwner
	getCirculatingSupply	Public		-
	setAutorizerAddress	External	✓	isAutorizer
	getAutorizer	External		-
	getLiquidityBacking	Public		-
	isOverLiquified	Public		-

Inheritance Graph



Flow Graph



Summary

Payinnov Protocol contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Payinnov Protocol is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error 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 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