Cyberscope

Audit Report

Moonprinter

June 2023

Network	BSC TESTNET
Address	0x3aCb247406680c28Dd5816Ec36423bE53ce318D6
Audited by	© cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

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

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	US	Untrusted Source	Unresolved
●	RCS	Redundant Conditional Statement	Unresolved
●	PAV	Pair/Router Address Validation	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RFC	Redundant Function Call	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	RE	Redundant Events	Unresolved
●	RSML	Redundant SafeMath Library	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

●	L11	Unnecessary Boolean equality	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	MoonPrinter
Compiler Version	v0.8.15+commit.e14f2714
Optimization	200 runs
Explorer	https://testnet.bscscan.com/address/0x3acb247406680c28dd5816ec36423be53ce318d6
Address	0x3acb247406680c28dd5816ec36423be53ce318d6
Network	BSC_TESTNET
Symbol	BRRR
Decimals	18
Total Supply	32.000.000.000.000

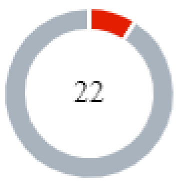
Audit Updates

Initial Audit	29 May 2023

Source Files

Filename	SHA256
MoonPrinter.sol	cf6329214b43620ee65ad37bb429121ed2e96b300e517e64ddd9e63226244853

Findings Breakdown



●	Critical	2
●	Medium	0
●	Minor / Informative	20

	Severity	Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	20	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	MoonPrinter.sol#L1447
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 enabled the owner will not be able to disable them again.

```
if (
  !_isExcludedFromFees[from] && !_isExcludedFromFees[to] && !swapping
) {
  require(tradingEnabled, "Trading not active");
}
```

Recommendation

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

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

US - Untrusted Source

Criticality	Critical
Location	MoonPrinter.sol#L1248
Status	Unresolved

Description

The contract uses an external contract in order to determine the transaction's flow. The external contract is untrusted. As a result, it may produce security issues and harm the transactions.

The `DividendTracker` could be mutated to an untrusted contract.

```
function updateDividendTracker(address newAddress) public onlyOwner {
    DividendTracker newDividendTracker = DividendTracker(
        payable(newAddress)
    );

    newDividendTracker.excludeFromDividends(
        address(newDividendTracker),
        true
    );
    newDividendTracker.excludeFromDividends(address(this), true);
    newDividendTracker.excludeFromDividends(address(router), true);
    dividendTracker = newDividendTracker;
}
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

RCS - Redundant Conditional Statement

Criticality	Minor / Informative
Location	MoonPrinter.sol#L1240
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 contract already includes a require statement, specifically `require(amountTokens > 0...)`, the subsequent if statement `if (amountTokens > 0)` becomes redundant.

```
require(amountTokens > 0, "ERC20: Enter some amount to burn");
if (amountTokens > 0) {
    uint256 AmountToBurn = amountTokens;
    totalBurned = totalBurned + amountTokens;
    _burn(sender, 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 statements.

PAV - Pair/Router Address Validation

Criticality	Minor / Informative
Location	MoonPrinter.sol#L1365,1387
Status	Unresolved

Description

The contract is missing address validation in the pair and router address arguments. The absence of validation reveals a potential vulnerability, as it lacks proper checks to ensure the integrity and validity of the pair and router address provided as an argument. The pair and router address are parameters used in certain methods of decentralized exchanges for functions like token swaps and liquidity provisions.

The absence of address validation in these addresses can introduce security risks and potential attacks. Without proper validation, if the owner's address is compromised, the contract may lead to unexpected behavior like loss of funds.

The argument `newPair` and `newRouter` are not validated.

```
function updateRouter(address newRouter) external onlyOwner {
    router = IRouter(newRouter);
}

function _setAutomatedMarketMakerPair(address newPair, bool value) private {
    require(automatedMarketMakerPairs[newPair] != value);
    automatedMarketMakerPairs[newPair] = value;

    if (value) {
        dividendTracker.excludeFromDividends(newPair, true);
    }

    emit SetAutomatedMarketMakerPair(newPair, value);
}
```

Recommendation

To mitigate the risks associated with the absence of address validation in the pair and router address arguments, it is recommended to implement comprehensive address validation mechanisms. A recommended approach could be to verify pair and router existence in the decentralized application. Prior to interacting with the contracts, perform checks to verify the existence and validity of the contract at the provided address. This can be achieved by querying the provider's contract or utilizing external libraries that provide contract verification services.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	MoonPrinter.sol#L1269,1276,1294,1298,1329,1374
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 state 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 excludeFromMaxWallet(address account, bool excluded)
    public
    onlyOwner
{
    _isExcludedFromMaxWallet[account] = excluded;
}

function excludeMultipleAccountsFromFees(
    address[] calldata accounts,
    bool excluded
) public onlyOwner {
    for (uint256 i = 0; i < accounts.length; i++) {
        _isExcludedFromFees[accounts[i]] = excluded;
    }
    emit ExcludeMultipleAccountsFromFees(accounts, excluded);
}

function setTreasuryWallet(address newtreasury) public onlyOwner {
    treasuryWallet = newtreasury;
}

function setDevWallet(address newWallet) public onlyOwner {
    devWallet = newWallet;
}

function setSwapEnabled(bool _enabled) external onlyOwner {
    swapEnabled = _enabled;
}

function setClaimEnabled(bool state) external onlyOwner {
    claimEnabled = state;
}
```

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.

RFC - Redundant Function Call

Criticality	Minor / Informative
Location	MoonPrinter.sol#L
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 variable `swapTokensAtAmount` could be set to zero, causing the method `swapAndLiquify` to be called with a zero amount for liquidity conversion. As a result, this function call becomes redundant and serves no practical purpose.

```
bool canSwap = contractTokenBalance >= swapTokensAtAmount;

if (
    canSwap &&
    !swapping &&
    swapEnabled &&
    automatedMarketMakerPairs[to] &&
    !_isExcludedFromFees[from] &&
    !_isExcludedFromFees[to]
) {
    swapping = true;

    if (totalSellTax > 0) {
        swapAndLiquify(swapTokensAtAmount);
    }

    swapping = false;
}
```

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 add additional check to prevent redundant function calls. The `swapTokensAtAmount` should be greater than zero.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	MoonPrinter.sol#L1520,1526
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 `contractrewardbalance` might not be splitted as expected.

```
uint256 devAmt = (contractrewardbalance * sellTaxes.dev) / totalTax;

uint256 treasuryAmt = (contractrewardbalance * sellTaxes.treasury) /
    totalTax;
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	MoonPrinter.sol#L1522,1530
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.

```
if (devAmt > 0) {
    (bool success, ) = payable(devWallet).call{value: devAmt}("");
    require(success, "Failed to send BNB to dev wallet");
}

if (treasuryAmt > 0) {
    (bool success, ) = payable(treasuryWallet).call{value: treasuryAmt}("");
    require(success, "Failed to send BNB to treasury wallet");
}
```

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.

RE - Redundant Events

Criticality	Minor / Informative
Location	MoonPrinter.sol#L1142
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 `GasForProcessingUpdated` , `SendDividends` , and `ProcessedDividendTracker` event is not utilized in the contract's implementation. Hence, it is redundant.

```
event GasForProcessingUpdated(
    uint256 indexed newValue,
    uint256 indexed oldValue
);
```

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 events.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	MoonPrinter.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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MoonPrinter.sol#L1164,1165
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 totalBuyTax = 5
uint256 public totalSellTax = 9
```

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	MoonPrinter.sol#L20,952,958,1026,1033,1040,1050,1214,1223,1316,1329,1650
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 public _Token
uint256 constant internal magnitude = 2**128
address _owner

function AddAirdropRewardsFromContract(uint256 amount) public onlyOwner {
        if (amount > 0) {
            IERC20 token = IERC20(address(this));
            bool success = token.transfer(address(dividendTracker), amount);
            if (success) {
                dividendTracker.distributeDividends(amount);
            }
        }
    }

...
```

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	MoonPrinter.sol#L732
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.

```
int256 private constant MAX_INT256 = ~(int256(1) << 255)
```

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	MoonPrinter.sol#L1304,1312,1324
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.

```
maxWallet = newNum * (10**18)
maxBuyAmount = maxBuy * 10**18
swapTokensAtAmount = amount * 10**18
```

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	MoonPrinter.sol#L778,1060
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

function _transfer(address from, address to, uint256 value) internal virtual
override {
    require(false);

    int256 _magCorrection =
magnifiedDividendPerShare.mul(value).toInt256Safe();
    magnifiedDividendCorrections[from] =
magnifiedDividendCorrections[from].add(_magCorrection);
    magnifiedDividendCorrections[to] =
magnifiedDividendCorrections[to].sub(_magCorrection);
  }
```

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.

L11 - Unnecessary Boolean equality

Criticality	Minor / Informative
Location	MoonPrinter.sol#L1607
Status	Unresolved

Description

Boolean equality is unnecessary when comparing two boolean values. This is because a boolean value is either true or false, and there is no need to compare two values that are already known to be either true or false.

it's important to be aware of the types of variables and expressions that are being used in the contract's code, as this can affect the contract's behavior and performance. The comparison to boolean constants is redundant. Boolean constants can be used directly and do not need to be compared to true or false.

```
value == true
```

Recommendation

Using the boolean value itself is clearer and more concise, and it is generally considered good practice to avoid unnecessary boolean equalities in Solidity code.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	MoonPrinter.sol#L1530,1626
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
AccountInfo memory info
```

Recommendation

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

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	MoonPrinter.sol#L979,1026,1033,1040,1050
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
string memory _name
string memory _symbol
address _owner
```

Recommendation

It's important to be aware of shadowing when working with local variables, as it can lead to confusion and unintended consequences if not used correctly. It's generally a good idea to choose unique names for local variables to avoid shadowing outer variables and causing confusion.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	MoonPrinter.sol#L1295,1299,1587,1651
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.

```
treasuryWallet = newtreasury
devWallet = newWallet

(bool success, ) = payable(recipient).call{
            value: address(this).balance
        }("")
_Token = _lpToken
```

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	MoonPrinter.sol#L3,56,83,166,250,279,668,724
Status	Unresolved

Description

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

```
pragma solidity ^0.8.0;
pragma solidity ^0.8.10;
pragma solidity ^0.8.6;
```

Recommendation

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

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

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	MoonPrinter.sol#L3,56,83,166,250,279,668,724
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.10;
pragma solidity ^0.8.0;
pragma solidity ^0.8.6;
```

Recommendation

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

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	MoonPrinter.sol#L1342,1580
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.

```
IERC20(tokenAddress).transfer(
            owner(),
            IERC20(tokenAddress).balanceOf(address(this))
        )

IERC20(tokenAddress).transfer(
            recipient,
            IERC20(tokenAddress).balanceOf(address(this))
        )
```

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**
IPair	Interface			
	getReserves	External		-
	token0	External		-
IFactory	Interface			
	createPair	External	✓	-
	getPair	External		-
IRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Context	Implementation			
	_msgSender	Internal		

	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-

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

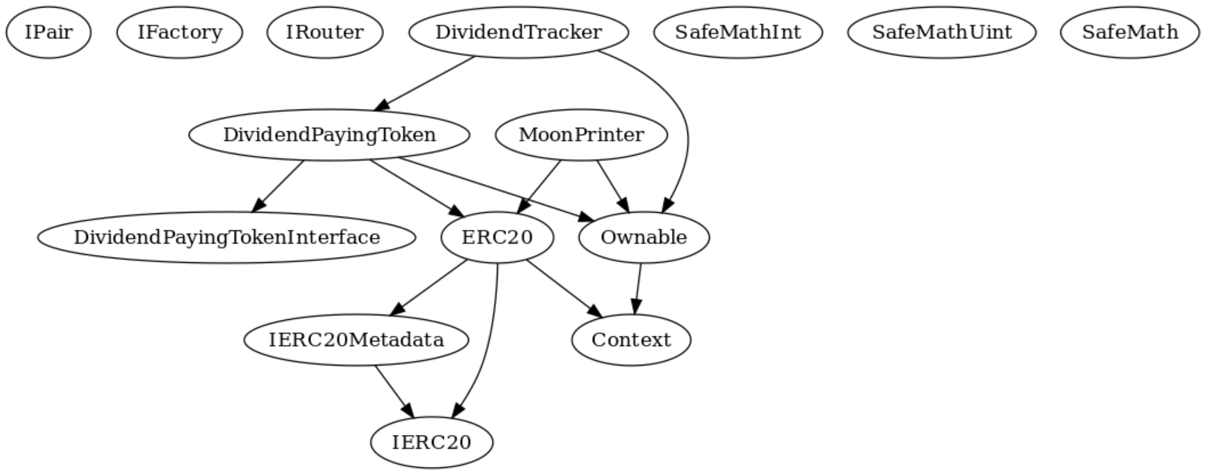
	dividendOf	External		-
	withdrawDividend	External	✓	-
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		

	div	Internal		
	mod	Internal		
	mod	Internal		
DividendPaying Token	Implementation	ERC20, DividendPayingTokenInterface, Ownable		
		Public	✓	ERC20
	distributeDividends	Public	✓	onlyOwner
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	
MoonPrinter	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	AddAirdropRewardsFromContract	Public	✓	onlyOwner

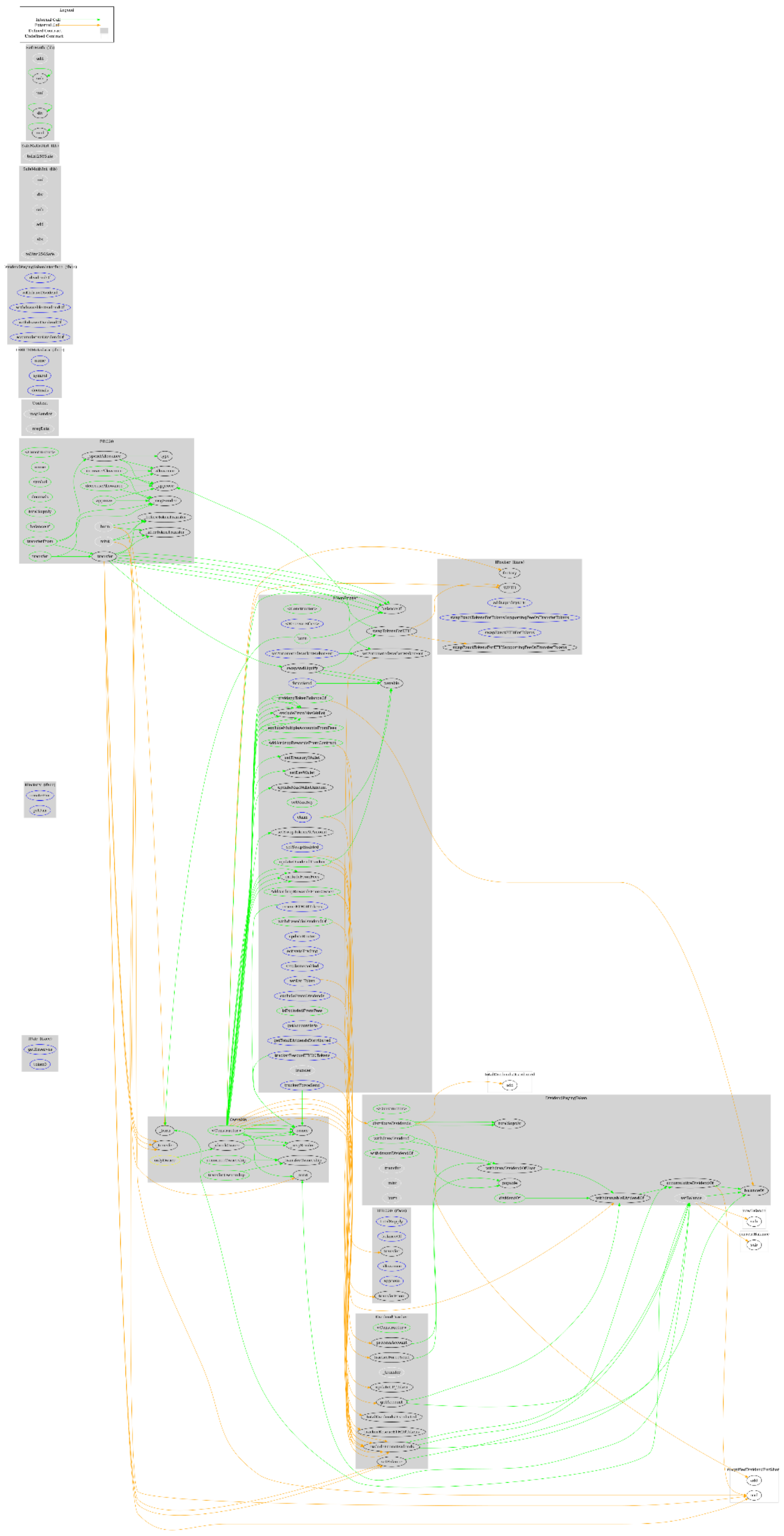
	AddAirdropRewardsFromOwner	Public	✓	onlyOwner
	burn	Public	✓	-
	updateDividendTracker	Public	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	excludeFromMaxWallet	Public	✓	onlyOwner
	excludeMultipleAccountsFromFees	Public	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	setTreasuryWallet	Public	✓	onlyOwner
	setDevWallet	Public	✓	onlyOwner
	updateMaxWalletAmount	Public	✓	onlyOwner
	setMaxBuy	Public	✓	onlyOwner
	setDiv_Token	External	✓	onlyOwner
	setSwapTokensAtAmount	Public	✓	onlyOwner
	setSwapEnabled	External	✓	onlyOwner
	claim	External	✓	-
	rescueETH20Tokens	External	✓	onlyOwner
	forceSend	External	✓	onlyOwner
	trackerRescueETH20Tokens	External	✓	onlyOwner
	trackerForceSend	External	✓	onlyOwner
	updateRouter	External	✓	onlyOwner
	activateTrading	External	✓	onlyOwner
	setClaimEnabled	External	✓	onlyOwner
	setAutomatedMarketMakerPair	External	✓	onlyOwner

	_setAutomatedMarketMakerPair	Private	✓	
	getTotalDividendsDistributed	External		-
	isExcludedFromFees	Public		-
	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-
	getAccountInfo	External		-
	_transfer	Internal	✓	
	swapAndLiquify	Private	✓	
	swapTokensForETH	Private	✓	
DividendTracker	Implementation	Ownable, DividendPayingToken		
		Public	✓	DividendPaying Token
	trackerRescueETH20Tokens	External	✓	onlyOwner
	trackerForceSend	External	✓	onlyOwner
	_transfer	Internal		
	excludeFromDividends	External	✓	onlyOwner
	getAccount	Public		-
	setBalance	External	✓	onlyOwner
	updateLP_Token	External	✓	onlyOwner
	processAccount	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Moonprinter contract implements a token mechanism. This audit investigates security issues, business logic concerns, and potential improvements. There are some functions that can be abused by the owner like stopping transactions. The contract can be converted into a honeypot and prevent users from selling if the owner abuses the admin functions. 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 fixed sell fee of 9% fee and a fixed buy fee of 5%.

Disclaimer

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