Cyberscope

Audit Report

Moonprinter

June 2023

Network	GOERLI
Address	0x76352b61F118e3bB83327b73FFCeCEd769682E5e
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	Unresolved
🔵	MT	Mints Tokens	Passed
🔵	BT	Burns Tokens	Passed
🔴	BC	Blacklists Addresses	Unresolved

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	RE	Redundant Events	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	PAV	Pair/Router Address Validation	Unresolved
●	RSML	Redundant SafeMath Library	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
●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved

●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	bigstufff
Compiler Version	v0.8.15+commit.e14f2714
Optimization	200 runs
Explorer	https://goerli.etherscan.io/address/0x76352b61f118e3bb83327b73ffceced769682e5e
Address	0x76352b61f118e3bb83327b73ffceced769682e5e
Network	GOERLI
Symbol	BB
Decimals	18
Total Supply	31,999,999,900,000

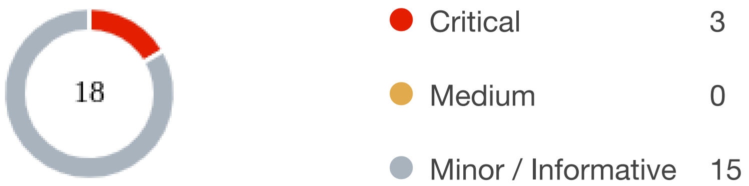
Audit Updates

Initial Audit	29 May 2023 https://github.com/cyberscope-io/audits/blob/main/brrr/v1/audit.pdf
Corrected Phase 2	20 Jun 2023

Source Files

Filename	SHA256
@openzeppelin/contracts/access/Ownable.sol	a8e4e1ae19d9bd3e8b0a6d46577eec098c01fbaffd3ec1252fd20d799e73393b
@openzeppelin/contracts/token/ERC20/ERC20.sol	d20d52b4be98738b8aa52b5bb0f88943f62128969b33d654fbca731539a7fe0a
@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol	af5c8a77965cc82c33b7ff844deb9826166689e55dc037a7f2f790d057811990
@openzeppelin/contracts/token/ERC20/IERC20.sol	7ebde70853ccafcf1876900dad458f46eb9444d591d39bfc58e952e2582f5587
@openzeppelin/contracts/utils/Context.sol	1458c260d010a08e4c20a4a517882259a23a4baa0b5bd9add9fb6d6a1549814a
DividendPayingTokenInterface.sol	c3cd9d950a49f9df6d867f38fc023fe091994bc8e22287a94e0eb7186004509d
IDEX.sol	3c0745ca3b62ab0e1f800e8b9885415f075afaae7a8f9e7caf7fc60ef2d8c9ff
lala.sol	70ef3e91c21226fe5da94c0393be03671689ec2a45ab6f1529f48040a808c9fa
MoonPrinterDividendPayingToken.sol	95b95b100d6b276d35cf4c1a6ca7caff851a046cb920b7a4e5512cc51cedf502

Findings Breakdown



Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	3	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	15	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	lala.sol#L360
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.

The `setMaxBuy` and `updateMaxWalletAmount` functions allow the contract owner to set the maximum amount of tokens that can be bought in a single transaction and the maximum amount of tokens that can be held in a single wallet, respectively.

However, the functions do not prevent the contract owner from setting `maxBuyAmount` and `maxWallet` to 0. If `maxBuyAmount` is set to 0, it would prevent all users from buying tokens. Similarly, if `maxWallet` is set to 0, it would prevent all users from receiving tokens.

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

function updateMaxWalletAmount(uint256 newNum) public onlyOwner {

  maxWallet = newNum * (10**18);
}

function setMaxBuy(uint256 maxBuy)
    public
    onlyOwner
{

    maxBuyAmount = maxBuy * 10**18;
}
```

Recommendation

To mitigate the risks associated with the contract owner setting `maxBuyAmount` and `maxWallet` to 0, it is recommended to implement safeguards in the `setMaxBuy` and `updateMaxWalletAmount` functions.

Moreover 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.

ELFM - Exceeds Fees Limit

Criticality	Critical
Status	Unresolved

Description

The smart contract initializes a high selling fee of 60% in the constructor. A selling fee of this magnitude is unusually high and could deter users from selling their tokens due to the substantial reduction in their returns.

The contract provides a mechanism to adjust this selling fee through the `normalizeTax` function, which can only be called by the contract owner. When invoked, this function reduces the selling fee from 60% to a more reasonable 7%.

However, once the normalize function has been called and the selling fee has been reduced, there is no mechanism in the contract to adjust the selling fee again.

If the contract owner does not call the normalize function, users would be subject to the high selling fee.

Also once the normalize function has been called, the contract lacks the flexibility to adjust the selling fee in response to changing market conditions as the selling fee is permanently set to 7%.

```
constructor(address _treasury, address _devWallet)
    ERC20("biggg", "BB")
{
    ...
    totalSellTax = 60;
    ...
}
...
function normalizeTax() public onlyOwner {
  totalBuyTax = 5;
  totalSellTax = 7;
  buyTaxes = Taxes(3, 2);
  sellTaxes = Taxes(4, 3);
}
```

Recommendation

The contract could embody a check for the maximum acceptable value or initial a more reasonable selling fee in the constructor. 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.

BC - Blacklists Addresses

Criticality	Critical
Location	lala.sol#L156
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `setBot` and `setBulkBot` functions.

```
function setBot(address bot, bool value) external onlyOwner{
    require(_isBot[bot] != value);
    _isBot[bot] = value;
}

function setBulkBot(address[] memory bots, bool value) external
onlyOwner{
    for(uint256 i; i<bots.length; i++){
        _isBot[bots[i]] = value;
    }
}
```

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 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.

RE - Redundant Events

Criticality	Minor / Informative
Location	lala.sol#L42
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 contracts 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.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	lala.sol#L436
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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	lala.sol#L435
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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	lala.sol#L181,188
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);
}
```

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.

PAV - Pair/Router Address Validation

Criticality	Minor / Informative
Location	lala.sol#L279,301
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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	MoonPrinterDividendPayingToken.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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	MoonPrinterDividendPayingToken.sol#L234,240,308,315,322,332lala.sol#L12,26,120,129,230,564
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.

```
address public _Token
uint256 constant internal magnitude = 2**128
address _owner

...
```

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	MoonPrinterDividendPayingToken.sol#L14
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	lala.sol#L218,226,238
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

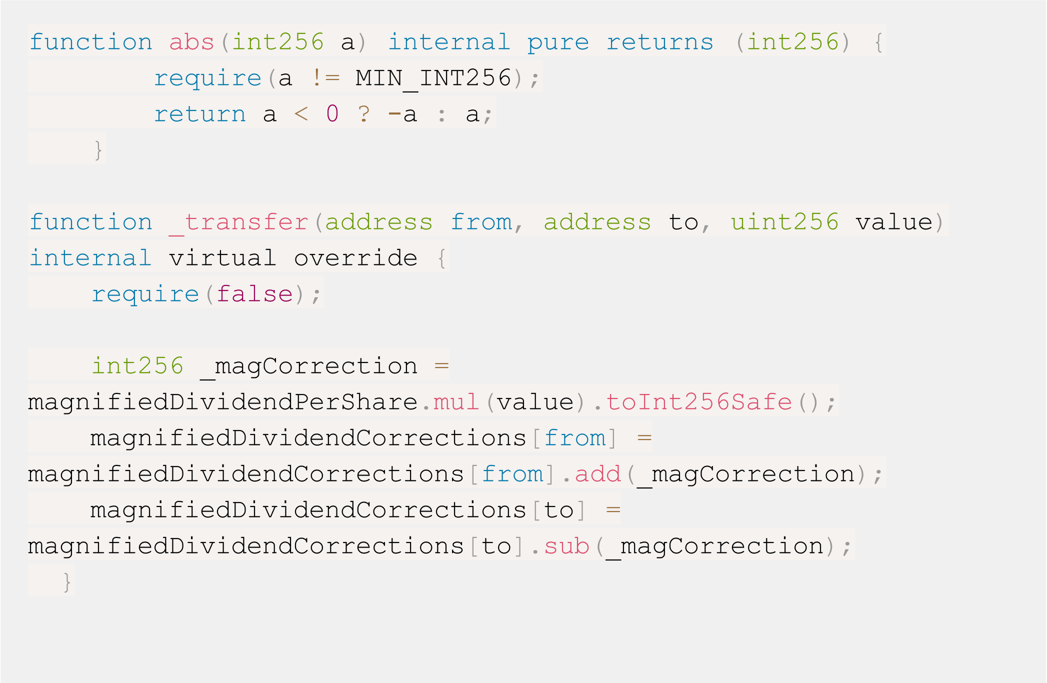
Criticality	Minor / Informative
Location	MoonPrinterDividendPayingToken.sol#L60,342
Status	Unresolved

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	lala.sol#L521
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	lala.sol#L162,445,540
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.

```
uint256 i
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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	lala.sol#L501,565
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.

```
(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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	MoonPrinterDividendPayingToken.sol#L3
Status	Unresolved

Description

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

```
pragma solidity ^0.8.10;
```

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	lala.sol#L256,494
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**
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-

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

	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
DividendPaying TokenInterface	Interface			
	dividendOf	External		-
	withdrawDividend	External	✓	-
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
IPair	Interface			
	getReserves	External		-
	token0	External		-
IFactory	Interface			
	createPair	External	✓	-
	getPair	External		-
IRouter	Interface			
	factory	External		-

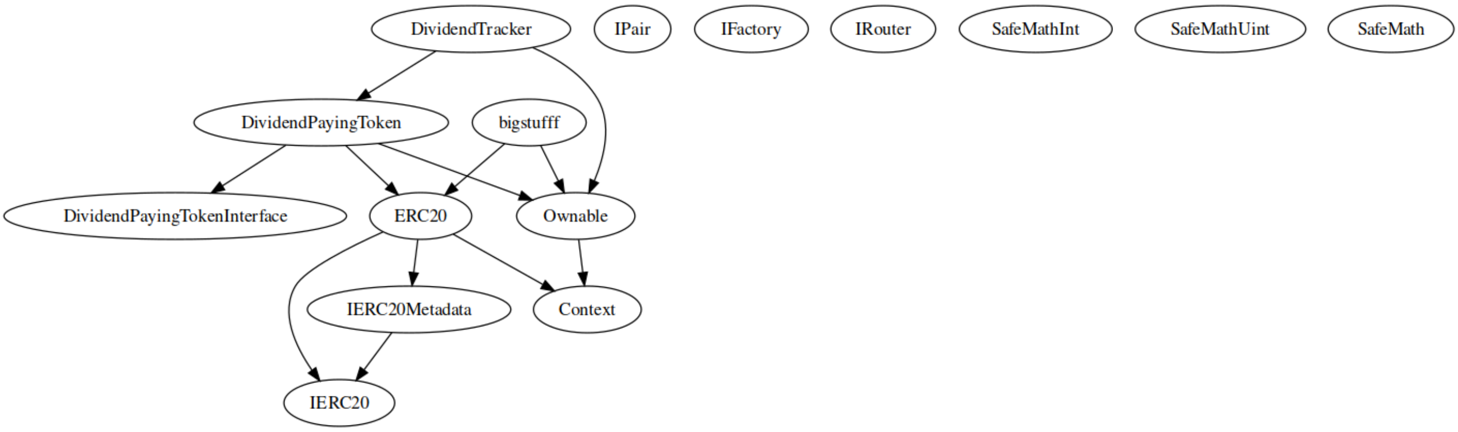
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
bigstufff	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	AddAirdropRewardsFromContract	Public	✓	onlyOwner
	AddAirdropRewardsFromOwner	Public	✓	onlyOwner
	burn	Public	✓	-
	setBot	External	✓	onlyOwner
	setBulkBot	External	✓	onlyOwner
	normalizeTax	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		

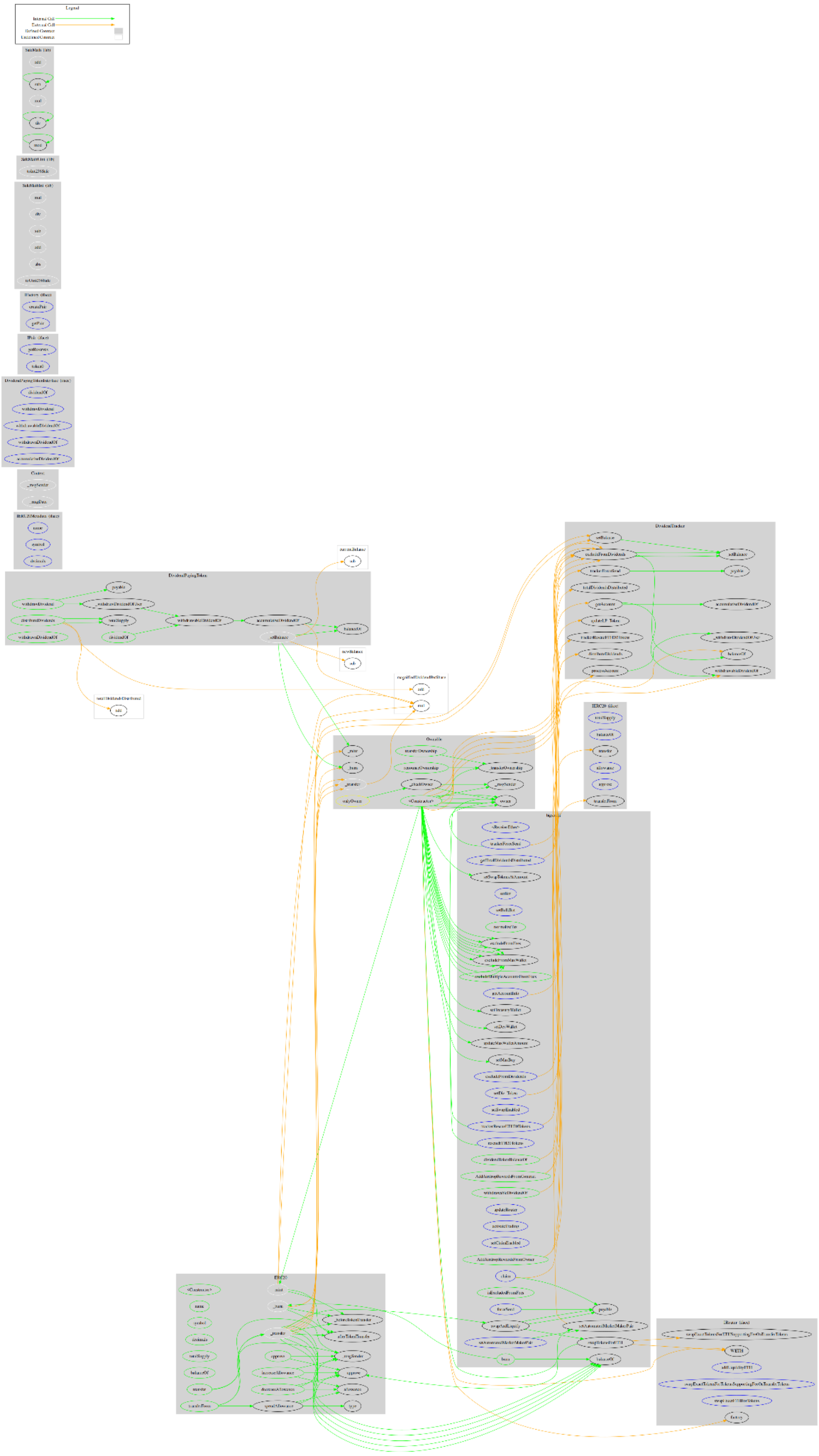
		Public	✓	DividendPaying Token
	trackerRescueETH20Tokens	External	✓	onlyOwner
	trackerForceSend	External	✓	onlyOwner
	_transfer	Internal		
	excludeFromDividends	External	✓	onlyOwner
	getAccount	Public		-
	setBalance	External	✓	onlyOwner
	updateLP_Token	External	✓	onlyOwner
	processAccount	External	✓	onlyOwner
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	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
SafeMath (lib)
add
sub
mul
div
mod
SafeMathUint (lib)
toInt256Safe
SafeMathInt (lib)
mul
div
sub
add
abs
toUint256Safe
IFactory (iface)
createPair
getPair
IPair (iface)
getReserves
token0
DividendPayingTokenInterface (iface)
dividendOf
withdrawDividend
withdrawableDividendOf
withdrawnDividendOf
accumulativeDividendOf
Context
_msgSender
_msgData
IERC20Metadata (iface)
name
symbol
decimals
DividendPayingToken
payable
withdrawDividend
_withdrawDividendOfUser
distributeDividends
totalSupply
withdrawableDividendOf
accumulativeDividendOf
balanceOf
withdrawnDividendOf
dividendOf
_setBalance
currentBalance
sub
newBalance
sub
magnifiedDividendPerShare
add
mul
totalDividendsDistributed
add
DividendTracker
setBalance
excludeFromDividends
_setBalance
trackerForceSend
payable
totalDividendsDistributed
getAccount
accumulativeDividendOf
updateLP_Token
trackerRescueETH20Tokens
_withdrawDividendOfUser
distributeDividends
balanceOf
processAccount
withdrawableDividendOf
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Ownable
_mint
transferOwnership
_burn
renounceOwnership
_transferOwnership
_transfer
_checkOwner
_msgSender
onlyOwner
<Constructor>
owner
<Receive Ether>
trackerForceSend
getTotalDividendsDistributed
setSwapTokensAtAmount
setFee
setBulkBot
normalizeFee
excludeFromFees
excludeFromMaxWallet
excludeMultipleAccountsFromFees
getAccountInfo
setTreasuryWallet
setDevWallet
updateMaxWalletAmount
setMaxBuy
excludeFromDividends
setDiv_Token
setSwapEnabled
trackerRescueETH20Tokens
rescueETH20Tokens
dividendTokenBalanceOf
AddAirdropRewardsFromContract
withdrawableDividendOf
updateRouter
activateTrading
setClaimEnabled
AddAirdropRewardsFromOwner
claim
isExcludedFromFees
forceSend
payable
swapAndLiquify
setAutomatedMarketMakerPair
setAutomatedMarketMakerPair
swapTokensForETH
burn
balanceOf
ERC20
<Constructor>
_mint
name
_burn
symbol
_beforeTokenTransfer
decimals
_transfer
afterTokenTransfer
totalSupply
approve
_msgSender
balanceOf
increaseAllowance
approve
transfer
decreaseAllowance
allowance
transferFrom
spendAllowance
type
IRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
addLiquidityETH
swapExactTokensForTokensSupportingFeeOnTransferTokens
swapExactETHForTokens
factory

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 stop transactions, manipulate the fees and massively blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats.

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