Cyberscope

Audit Report

Recordian Music

November 2024

Network BASE

Address 0x085d4444BBD7c1Df34b0baF1982df8C1F0fC49Ff

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
	AOI	Arithmetic Operations Inconsistency	Unresolved
	CCR	Contract Centralization Risk	Unresolved
	DDP	Decimal Division Precision	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	MEE	Missing Events Emission	Unresolved
	MEM	Missing Error Messages	Unresolved
	MU	Modifiers Usage	Unresolved
	PTRP	Potential Transfer Revert Propagation	Unresolved
	PVC	Price Volatility Concern	Unresolved
	RC	Repetitive Calculations	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	RSRS	Redundant SafeMath Require Statement	Unresolved
	RSD	Redundant Swap Duplication	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved

●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation**: This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation**: This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical**: Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium**: Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor**: Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative**: Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
Critical	Highly Likely / High Impact
Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
Minor / Informative	Unlikely / Low to no Impact

Review

Contract Name	RMUSIC
Compiler Version	v0.8.21+commit.d9974bed
Optimization	999 runs
Explorer	https://basescan.org/address/0x085d4444bbd7c1df34b0baf1982df8c1f0fc49ff
Address	0x085d4444bbd7c1df34b0baf1982df8c1f0fc49ff
Network	BASE
Symbol	RMUSIC
Decimals	9
Total Supply	3,000,000,000
Badge Eligibility	Yes

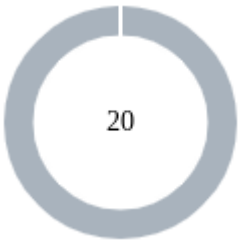
Audit Updates

Initial Audit	22 Nov 2024

Source Files

Filename	SHA256
RMUSIC.sol	aeb0577b97c03107c1afb95f06e27fe1756e5d64af2e46cd132500d902d7987f

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	20

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

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	RMUSIC.sol#L630,631,635,646,651,665,673,674,680,681,682,689,696,703,706,796,800,804,815,819,847,903,913
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
if (amount > 0) {
magnifiedDividendPerShare
magnifiedDividendPerShare.add(
(amount).mul(magnitude) / totalSupply()
);
emit DividendsDistributed(msg.sender, amount);

totalDividendsDistributed = totalDividendsDistributed.add(amount);
}
```

```
if ((treasuryFeeOnBuy + treasuryFeeOnSell) > 0) {
uint256 treasuryBNB = (newBalance *
(treasuryFeeOnBuy + treasuryFeeOnSell)) / bnbShare;
sendBNB(payable(treasuryWallet), treasuryBNB);
emit SendMarketing(treasuryBNB);
}
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	RMUSIC.sol#L81,85,626,744,749,759,766,850,914,1038,1049,1075,1086,1098,1110,1303,1308,1313,1320,1324,1349,1388,1396,1413,1420,1432
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.

```
function renounceOwnership() public virtual onlyOwner {}
function transferOwnership(address newOwner) public virtual
onlyOwner {}
function distributeDividends(uint256 amount) public onlyOwner {}
function updateMinimumTokenBalanceForDividends(
uint256 _newMinimumBalance
) external onlyOwner {}
function excludeFromDividends(address account) external onlyOwner
{}
function updateClaimWait(uint256 newClaimWait) external onlyOwner
{}
function setLastProcessedIndex(uint256 index) external onlyOwner {}
function setBalance(
address payable account,
uint256 newBalance
) external onlyOwner {}
function processAccount(
address payable account,
bool automatic
) public onlyOwner returns (bool) {}
function claimStuckTokens(address token) external onlyOwner {}
function burn(uint256 amount) external onlyOwner {}
function excludeFromFees(
address account,
bool excluded
) external onlyOwner {}
function updateBuyFees(
uint256 _liquidityFeeOnBuy,
uint256 _treasuryFeeOnBuy,
uint256 _rewardsFeeOnBuy
) external onlyOwner {}
function updateSellFees(
uint256 _liquidityFeeOnSell,
uint256 _treasuryFeeOnSell,
uint256 _rewardsFeeOnSell
) external onlyOwner {}
function setSwapEnabled(bool _enabled) external onlyOwner {}
function updateGasForProcessing(uint256 newValue) public onlyOwner
{}
function updateMinimumBalanceForDividends(
uint256 newMinimumBalance
) external onlyOwner {}
function updateClaimWait(uint256 newClaimWait) external onlyOwner
{}
function excludeFromDividends(address account) external onlyOwner
{}function claimAddress(address claimee) external onlyOwner {}
function setLastProcessedIndex(uint256 index) external onlyOwner {}
function setMaxTransactionAmounts(
uint256 _maxTransactionAmountBuy,
uint256 _maxTransactionAmountSell
```

```
) external onlyOwner {}
function excludeFromMaxTransactionLimit(
address account,
bool exclude
) external onlyOwner {}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	RMUSIC.sol#L1199
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.

```
if ((rewardsFeeOnBuy + rewardsFeeOnSell) > 0) {
    uint256 rewardBNB = (newBalance *
    (rewardsFeeOnBuy + rewardsFeeOnSell)) / bnbShare;
    swapAndSendDividends(rewardBNB);
}
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	RMUSIC.sol#L1001
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.

```
uniswapV2Pair
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	RMUSIC.sol#L1303,1308,1320,1324
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 setSwapTokensAtAmount(uint256 newAmount) external
onlyOwner{}
function setSwapEnabled(bool _enabled) external onlyOwner{}
function updateMinimumBalanceForDividends(uint256
newMinimumBalance) external onlyOwner {}
function updateClaimWait(uint256 newClaimWait) external onlyOwner
{}
```

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.

MEM - Missing Error Messages

Criticality	Minor / Informative
Location	RMUSIC.sol#L623,674,746
Status	Unresolved

Description

The contract is missing error messages. Specifically, there are no error messages to accurately reflect the problem, 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(totalSupply() > 0)
require(false)
require(!excludedFromDividends[account])
```

Recommendation

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

MU - Modifiers Usage

Criticality	Minor / Informative
Location	RMUSIC.sol#L82,86,541,570,1113,1120
Status	Unresolved

Description

The contract is using repetitive statements on some methods to validate some preconditions. In Solidity, the form of preconditions is usually represented by the modifiers. Modifiers allow you to define a piece of code that can be reused across multiple functions within a contract. This can be particularly useful when you have several functions that require the same checks to be performed before executing the logic within the function.

```
require(_treasuryWallet != address(0),
"Marketing wallet cannot be the zero address"
);
require(from != address(0),
"ERC20: transfer from the zero address");
require(to != address(0),
"ERC20: transfer to the zero address");
newOwner != address(0)
```

Recommendation

The team is advised to use modifiers since it is a useful tool for reducing code duplication and improving the readability of smart contracts. By using modifiers to perform these checks, it reduces the amount of code that is needed to write, which can make the smart contract more efficient and easier to maintain.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	RMUSIC.sol#L1195
Status	Unresolved

Description

The contract sends funds to a `treasuryWallet` 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((treasuryFeeOnBuy + treasuryFeeOnSell) > 0) {
    uint256 treasuryBNB = newBalance * (treasuryFeeOnBuy +
treasuryFeeOnSell) / bnbShare;
    sendBNB(payable(treasuryWallet), treasuryBNB);
    emit SendMarketing(treasuryBNB);
}
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	RMUSIC.sol#L1169
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` 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. Additionally, the contract swaps the entire accumulated balance at that time, not just the threshold amount, further increasing the swapped amount.

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 _transfer(address from,address to,uint256 amount) internal
override {
uint256 contractTokenBalance = balanceOf(address(this));
...
if(liquidityFeeOnBuy + liquidityFeeOnSell > 0) {
    liquidityTokens = contractTokenBalance * (liquidityFeeOnBuy +
liquidityFeeOnSell) / (totalBuyFee + totalSellFee);
    swapAndLiquify(liquidityTokens);
}
...
}
```

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

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	RMUSIC.sol#L1120
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

```
totalBuyFee + totalSellFee
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	RMUSIC.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 in cases where the explanatory error message is not used.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library in cases where the revert error message is not used. 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/stable/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSRS - Redundant SafeMath Require Statement

Criticality	Minor / Informative
Location	RMUSIC.sol#L98
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns (uint256)
{
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSD - Redundant Swap Duplication

Criticality	Minor / Informative
Location	RMUSIC.sol#L1120
Status	Unresolved

Description

The contract contains multiple swap methods that individually perform token swaps and transfer promotional amounts to specific addresses and features. This redundant duplication of code introduces unnecessary complexity and increases dramatically the gas consumption. By consolidating these operations into a single swap method, the contract can achieve better code readability, reduce gas costs, and improve overall efficiency.

```
function _transfer(
    address from,
    address to,
    uint256 amount) internal override {
    ...
    swapAndLiquify(liquidityTokens);
    ...
    try
        uniswapV2Router
        .swapExactTokensForETHSupportingFeeOnTransferTokens(
            contractTokenBalance,
            0,
            path,
            address(this),
            block.timestamp
            )
            {} catch {}
    ...
    swapAndSendDividends(rewardBNB);
    ...
}
```

Recommendation

A more optimized approach could be adopted to perform the token swap operation once for the total amount of tokens and distribute the proportional amounts to the corresponding addresses, eliminating the need for separate swaps.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	RMUSIC.sol#L278,279,296,316,609,656,660,664,668,740,774,1082,1094,1106,1304,1416
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 DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
uint256 constant internal magnitude = 2**128
address _owner
uint256 _newMinimumBalance
address _account
uint256 _liquidityFeeOnBuy
uint256 _treasuryFeeOnBuy
uint256 _rewardsFeeOnBuy
uint256 _liquidityFeeOnSell
uint256 _rewardsFeeOnSell
uint256 _treasuryFeeOnSell

...
```

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/stable/style-guide.html#naming-conventions.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	RMUSIC.sol#L763,1301
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.

```
lastProcessedIndex = index
swapTokensAtAmount = newAmount
```

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	RMUSIC.sol#L673
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 _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.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	RMUSIC.sol#L1161
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 liquidityTokens
```

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	RMUSIC.sol#L618,656,660,664,668
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	RMUSIC.sol#L619
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.

```
rewardToken = _rewardToken
```

Recommendation

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

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	RMUSIC.sol#L1042
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.

```
ERC20token.transfer(msg.sender, balance)
```

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

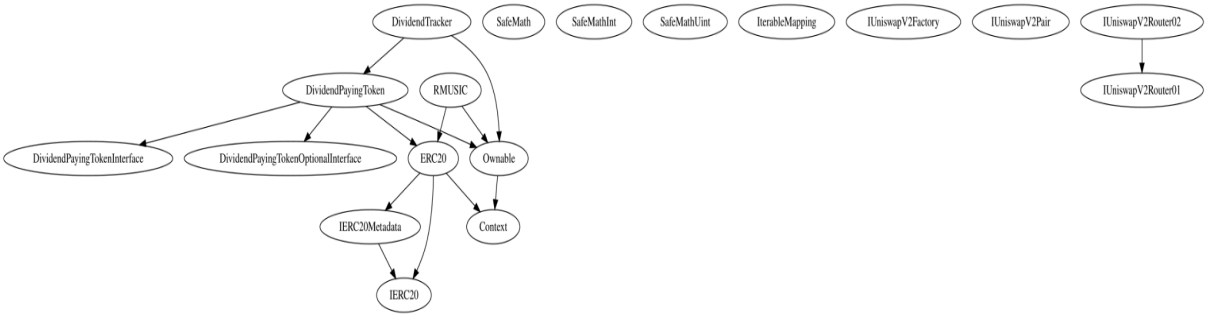
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
DividendPayingToken	Implementation	ERC20, Ownable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
		Public	✓	ERC20
	distributeDividends	Public	✓	onlyOwner
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_setBalance	Internal	✓	
DividendTracker	Implementation	Ownable, DividendPayingToken		
		Public	✓	DividendPayingToken

	_transfer	Internal		
	withdrawDividend	Public		-
	updateMinimumTokenBalanceForDividends	External	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	setLastProcessedIndex	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfTokenHolders	External		-
	getAccount	Public		-
	getAccountAtIndex	Public		-
	canAutoClaim	Private		
	setBalance	External	✓	onlyOwner
	process	Public	✓	-
	processAccount	Public	✓	onlyOwner
RMUSIC	Implementation	ERC20, Ownable		
		Public	Payable	ERC20
		External	Payable	-
	claimStuckTokens	External	✓	onlyOwner
	burn	External	✓	onlyOwner
	isContract	Internal		
	sendBNB	Internal	✓	
	_setAutomatedMarketMakerPair	Private	✓	
	excludeFromFees	External	✓	onlyOwner

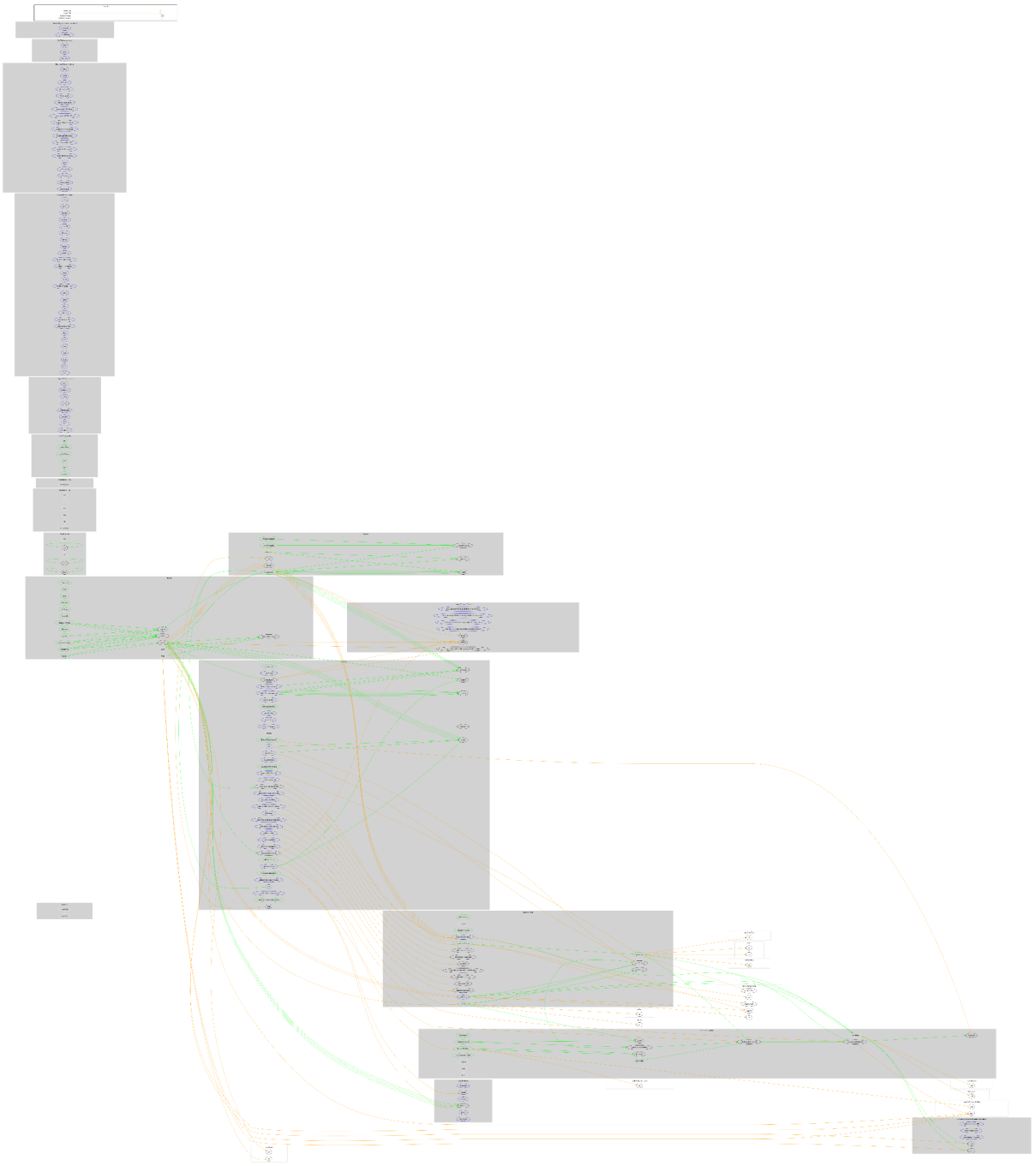
	isExcludedFromFees	Public		-
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	changeTreasuryWallet	External	✓	onlyOwner
	_transfer	Internal	✓	
	swapAndLiquify	Private	✓	
	swapAndSendDividends	Private	✓	
	setSwapTokensAtAmount	External	✓	onlyOwner
	setSwapEnabled	External	✓	onlyOwner
	updateGasForProcessing	Public	✓	onlyOwner
	updateMinimumBalanceForDividends	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getClaimWait	External		-
	getTotalDividendsDistributed	External		-
	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-
	totalRewardsEarned	Public		-
	excludeFromDividends	External	✓	onlyOwner
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	claimAddress	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	setLastProcessedIndex	External	✓	onlyOwner

	getNumberOfDividendTokenHolders	External		-
	setEnableMaxTransactionLimit	External	✓	onlyOwner
	setMaxTransactionAmounts	External	✓	onlyOwner
	excludeFromMaxTransactionLimit	External	✓	onlyOwner
	isExcludedFromMaxTransaction	Public		-

Inheritance Graph



Flow Graph



Summary

Recordian Music contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Recordian Music 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 5% 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

cyberscope.io