Cyberscope

Audit Report

Altmoon

April 2023

Network	BSC
Address	0x83a529ce010b4fa745c7585a1e9acd94586554a2
Audited by	© cyberscope

Table of Contents

Review

Contract Name	Altmoon
Compiler Version	v0.8.9+commit.e5eed63a
Optimization	200 runs
Explorer	https://bscscan.com/address/0x83a529ce010b4fa745c7585a1e9acd94586554a2
Address	0x83a529ce010b4fa745c7585a1e9acd94586554a2
Network	BSC
Symbol	ALTM
Decimals	18
Total Supply	10,000,000

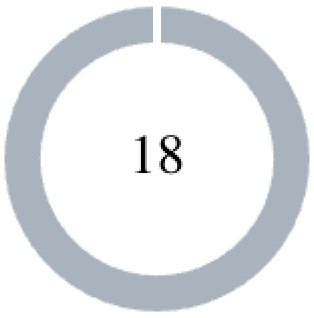
Audit Updates

Initial Audit	27 Apr 2023
Corrected Phase 2	29 Apr 2023

Source Files

Filename	SHA256
Altmoon.sol	837b6993797fd7ced9457b84b8eeb68937439e38f7723393be75680afe2e2fdf

Findings Breakdown



🟥	Critical	0
🟧	Medium	0
●	Minor / Informative	18

Severity		Unresolved	Acknowledged	Resolved	Other
🟥	Critical	0	0	0	0
🟧	Medium	0	0	0	0
●	Minor / Informative	18	0	0	0

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OCTD	Transfers Contract's Tokens	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	ULTW	Transfers Liquidity to Team Wallet	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	EFD	Exclude From Dividend	Unresolved
●	DDP	Decimal Division Precision	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	MRM	Missing Revert Messages	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSK	Redundant Storage Keyword	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
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved

●	L16	Validate Variable Setters	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L22	Potential Locked Ether	Unresolved

EFD - Exclude From Dividend

Criticality	Minor / Informative
Location	Altmoon.sol#L1688
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 changes the exclude from dividends state for the corresponding account to true even if the state is already true.

```
function excludeFromDividends(address account) external
onlyOwner {
    excludedFromDividends[account] = true;

    _setBalance(account, 0);
    tokenHoldersMap.remove(account);

    emit ExcludeFromDividends(account);
}
```

Recommendation

The team is advised to take these segments into consideration and rewrite them so the runtime will be more performant. That way it will improve the efficiency and performance of the source code and reduce the cost of executing it.

The contract should update the state only when it changes.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	Altmoon.sol#L1500
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 `fees` might not be splitted as expected.

```
if (automatedMarketMakerPairs[to] && totalSellFees > 0){
    fees = amount.mul(totalSellFees).div(100);
    tokensForRewards += fees * rewardsSellFee / totalSellFees;
    tokensForLiquidity += fees * liquiditySellFee /
totalSellFees;
    tokensForGrowth += fees * growthSellFee / totalSellFees;
}
// on buy
else if(automatedMarketMakerPairs[from] && totalBuyFees > 0) {
    fees = amount.mul(totalBuyFees).div(100);
    tokensForRewards += fees * rewardsBuyFee / totalBuyFees;
    tokensForLiquidity += fees * liquidityBuyFee /
totalBuyFees;
    tokensForGrowth += fees * growthBuyFee / totalBuyFees;
} else if (totalTransferFees > 0) {
    fees = amount.mul(totalTransferFees).div(100);
    tokensForRewards += fees * rewardsTransferFee /
totalTransferFees;
    tokensForLiquidity += fees * liquidityTransferFee /
totalTransferFees;
    tokensForGrowth += fees * growthTransferFee /
totalTransferFees;
}
```

Recommendation

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

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Altmoon.sol#L1380
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `minimumAmountSwap` sets a threshold where the contract will trigger the swap functionality. If the variable is set to a big number, then the contract will swap a huge amount of tokens for ETH.

It is important to note that the price of the token representing it, can be highly volatile. This means that the value of a price volatility swap involving Ether could fluctuate significantly at the triggered point, potentially leading to significant price volatility for the parties involved.

```
function updateMinimumAmountSwap (uint256 _minimumAmountSwap)
external onlyOwner{
    minimumAmountSwap = _minimumAmountSwap;
}
```

Recommendation

The contract could ensure that it will not sell more than a reasonable amount of tokens in a single transaction. A suggested implementation could check that the maximum amount should be less than a fixed percentage of the total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

MRM - Missing Revert Messages

Criticality	Minor / Informative
Location	Altmoon.sol#L1004,1698
Status	Unresolved

Description

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

```
require(totalBalance > 0);
require(excludedFromDividends[account]);
```

Recommendation

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

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Altmoon.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.

RSK - Redundant Storage Keyword

Criticality	Minor / Informative
Location	Altmoon.sol#L92,96,103,109
Status	Unresolved

Description

The contract uses the `storage` keyword in a view function. The `storage` keyword is used to persist data on the contract's storage. View functions are functions that do not modify the state of the contract and do not perform any actions that cost gas (such as sending a transaction). As a result, the use of the `storage` keyword in view functions is redundant.

```
Map storage map
```

Recommendation

It is generally considered good practice to avoid using the `storage` keyword in view functions because it is unnecessary and can make the code less readable.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Altmoon.sol#L1146,1149
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 liquidityActiveBlock = 0
bool public tradingActive = false
```

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	Altmoon.sol#L31,32,49,835,975,1066,1073,1080,1090,1144,1196,1298,1307,1316,1380,1722
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
uint8 public _decimals
event growthWalletUpdated(address indexed newWallet, address
indexed oldWallet);
bool _flag
address _presaleRouterAddress
address _presaleAddress
uint256 _minimumAmountSwap
address _account
```

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	Altmoon.sol#L766
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	Altmoon.sol#L1381
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.

```
minimumAmountSwap = _minimumAmountSwap
```

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	Altmoon.sol#L440,812
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 _burn(address account, uint256 amount) internal
virtual {
        require(account != address(0), "ERC20: burn from the
zero address");

        _beforeTokenTransfer(account, address(0), amount);

        _balances[account] = _balances[account].sub(amount,
"ERC20: burn amount exceeds balance");
        _totalSupply = _totalSupply.sub(amount);
        emit Transfer(account, address(0), amount);
    }

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

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Altmoon.sol#L1501,1502,1503,1504,1508,1509,1510,1511,1513,1514,1515,1516
Status	Unresolved

Description

It is important to be aware of the order of operations when performing arithmetic calculations. This is especially important when working with large numbers, as the order of operations can affect the final result of the calculation. Performing divisions before multiplications may cause loss of prediction.

```
fees = amount.mul(totalTransferFees).div(100)
tokensForGrowth += fees * growthTransferFee / totalTransferFees
```

Recommendation

To avoid this issue, it is recommended to carefully consider the order of operations when performing arithmetic calculations in Solidity. It's generally a good idea to use parentheses to specify the order of operations. The basic rule is that the multiplications should be prior to the divisions.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	Altmoon.sol#L1534
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 claims
uint256 iterations
uint256 lastProcessedIndex
```

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	Altmoon.sol#L1066,1073,1080,1090,1227
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.

```
address _owner
uint256 totalSupply = 1e7 * (10**_decimals)
```

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	Altmoon.sol#L1308,1311,1317,1318,1370
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.

```
presaleAddress = _presaleAddress
presaleRouterAddress = _presaleRouterAddress
growthWallet = newGrowthWallet
```

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	Altmoon.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 ^0.8.9;
```

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.

L22 - Potential Locked Ether

Criticality	Minor / Informative
Location	Altmoon.sol#L1001
Status	Unresolved

Description

The contract contains Ether that has been placed into a Solidity contract and is unable to be transferred. Thus, it is impossible to access the locked Ether. This may produce a financial loss for the users that have called the payable method.

```
receive() external payable {
    distributeDividends();
  }
```

Recommendation

The team is advised to either remove the payable method or add a withdraw functionality. it is important to carefully consider the risks and potential issues associated with locked Ether.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	allowance	Public		-
	transfer	Public	✓	-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	_transfer	Internal	✓	

	_burn	Internal	✓	
	_mint	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPayingTokenInterface	Interface			
	dividendOf	External		-
	distributeDividends	External	Payable	-
	withdrawDividend	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		

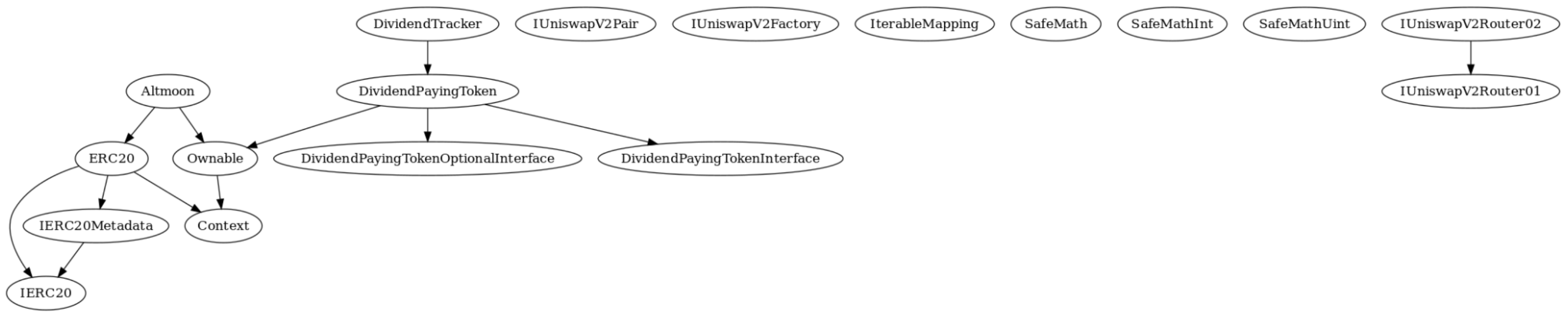
	mod	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	transferOwnership	Public	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner
DividendPaying Token	Implementation	DividendPayingTokenInterface, DividendPayingTokenOptionalInterface, Ownable		
		External	Payable	-
	distributeDividends	Public	Payable	-
	distributeTokenDividends	Public	✓	onlyOwner
	withdrawDividend	Public	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	Public		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	Public		-
	accumulativeDividendOf	Public		-
	_increase	Internal	✓	
	_reduce	Internal	✓	
	_setBalance	Internal	✓	

Altmoon	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	setSwapEnabled	Public	✓	onlyOwner
	decimals	Public		-
	addPresaleAddressForExclusions	External	✓	onlyOwner
	emergencyPresaleAddressUpdate	External	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	includeInDividends	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	excludeMultipleAccountsFromFees	External	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	setMinimumTokenBalanceForDividends	Public	✓	onlyOwner
	updateGrowthWallet	External	✓	onlyOwner
	updateGasForProcessing	External	✓	onlyOwner
	updateMinimumAmountSwap	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getClaimWait	External		-
	getTotalDividendsDistributed	External		-
	withdrawableDividendOf	Public		-
	dividendTokenBalanceOf	Public		-

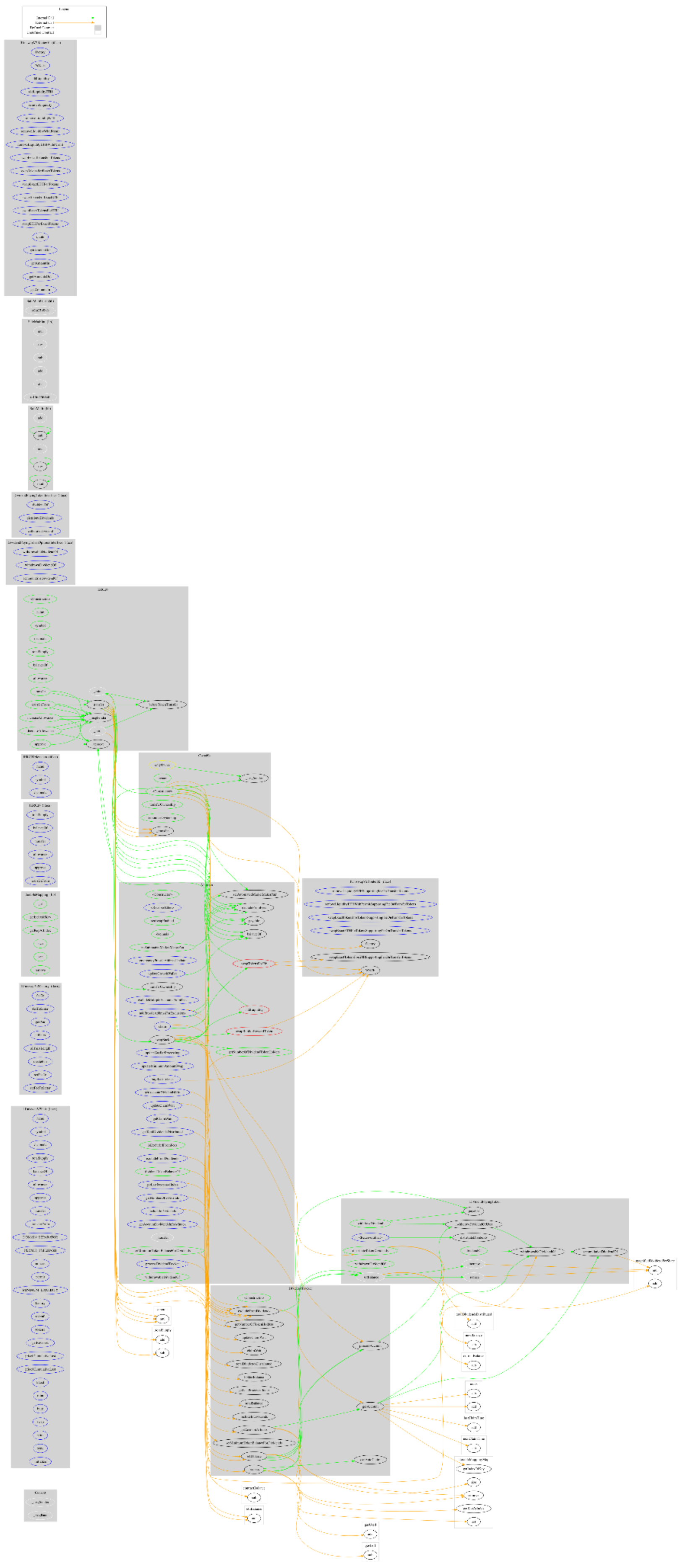
	isExcludedFromFees	Public		-
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	getLastProcessedIndex	External		-
	getNumberOfDividendTokenHolders	Public		-
	getNumberOfDividends	External		-
	_transfer	Internal	✓	
	swapBnbForRewardToken	Private	✓	
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	swapBack	Private	✓	
	buyBackTokens	External	✓	onlyOwner
DividendTracker	Implementation	DividendPayingToken		
		Public	✓	-
	setMinimumTokenBalanceForDividends	External	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	includeInDividends	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getLastProcessedIndex	External		-
	getNumberOfTokenHolders	Public		-

	getAccount	Public		-
	getAccountAtIndex	Public		-
	canAutoClaim	Private		
	setBalance	External	✓	onlyOwner
	process	Public	✓	-
	processAccount	Public	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Altmoon contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. Altmoon 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 9% fee.

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