Cyberscope

Audit Report

AMAL

May 2024

Network	BSC
Address	0x5eA231BE29f14cAD02daf64AFc9192A48Af871a5
Audited by	© cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
	ST	Stops Transactions	Renounced
	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
	CO	Code Optimization	Acknowledged
	CCR	Contract Centralization Risk	Renounced
	IDI	Immutable Declaration Improvement	Acknowledged
	MEM	Misleading Error Messages	Acknowledged
	MPL	Missing Protocol Limits	Renounced
	PLPI	Potential Liquidity Provision Inadequacy	Acknowledged
	PVC	Price Volatility Concern	Acknowledged
	RCS	Redundant Comment Segments	Acknowledged
	RPC	Redundant Period Calculation	Acknowledged
	RSML	Redundant SafeMath Library	Acknowledged
	RSW	Redundant Storage Writes	Acknowledged
	RVD	Redundant Variable Declaration	Acknowledged
	L02	State Variables could be Declared Constant	Acknowledged
	L09	Dead Code Elimination	Acknowledged

●	L16	Validate Variable Setters	Acknowledged
●	L20	Succeeded Transfer Check	Acknowledged

Table of Contents

Review

Contract Name	AMALToken
Compiler Version	v0.8.11+commit.d7f03943
Optimization	200 runs
Explorer	https://bscscan.com/address/0x5ea231be29f14cad02daf64afc9192a48af871a5
Address	0x5ea231be29f14cad02daf64afc9192a48af871a5
Network	BSC
Symbol	AMAL
Decimals	18
Total Supply	144,686,680.676
Badge Eligibility	Yes

Audit Updates

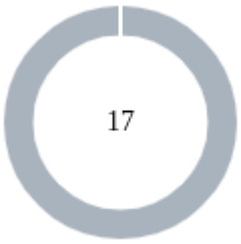
Initial Audit	02 May 2024
Corrected Phase 2	14 May 2024

Source Files

Filename	SHA256
amal/contracts/DividendDistributor.sol	0b9c011bcdb798efdd88480ce7f5ee5051c7967a2bc26d2d00fa84344139dcb9

amal/contracts/AMALToken.sol	933e51851215d98d447267d7e393eb3b2 3b33991634b5e5b4a1f73e13cb8f731

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	17

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	0	0	0	0
Medium	0	0	0	0
Minor / Informative	0	14	3	0

ST - Stops Transactions

Criticality	Minor / Informative
Location	amal/contracts/AMALToken.sol#L112,155
Status	Renounced

Description

The contract owner has the authority to stop the transactions for all users excluding the owner. The owner may take advantage of it by setting the maxWalletLimit to zero.

```
uint256 validTokenTransfer = ((totalSupply() * maxWalletLimit) /
    1000) - balanceOf(to);
require(amount <= validTokenTransfer, "Max Limit Reach");
...
uint256 validTokenTransfer = ((totalSupply() * maxWalletLimit) /
    1000) - balanceOf(to);
require(amount <= validTokenTransfer, "Max Limit Reach");
```

Recommendation

The contract could embody a check for not allowing setting the `maxWalletLimit` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. 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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

Permanent Solution:

- Renouncing the ownership, which will eliminate the threats but it is non-reversible.

Team Update

The contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link: https://bscscan.com/tx/0xc21b7ea4b0bab1edecd05011ec270c6242eb35e97d785b1e822cc4679074df5f

CO - Code Optimization

Criticality	Minor / Informative
Location	amal/contracts/DividendDistributor.sol#L174,190,198
Status	Acknowledged

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 is designed to determine and distribute dividends to shareholders using the `process` function, which calls both `shouldDistribute` and `distributeDividend` for each shareholder. Both these functions internally call `getPendingUSDTEarnings` to calculate the pending dividends for a shareholder. This results in the `getPendingUSDTEarnings` function being executed twice for the same shareholder during a single iteration of the process loop. This redundant execution can lead to inefficiencies in gas usage and increased transaction costs, particularly with a high number of shareholders.

```
function process(uint256 gas) external onlyToken {
    ...
        if (shouldDistribute(shareholders[currentIndex])) {
            distributeDividend(shareholders[currentIndex]);
        }
        ...
}

function shouldDistribute(
    address shareholder
) internal view returns (bool) {
    return
        shareholderClaims[shareholder] + minPeriod < block.timestamp
&&
        getPendingUSDTEarnings(shareholder) > minDistribution;
}

function distributeDividend(address shareholder) internal {
    if (shares[shareholder].amount == 0) {
        return;
    }
    uint256 amount = getPendingUSDTEarnings(shareholder);
    ...
    }
```

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 modify the `process` function to call `getPendingUSDTEarnings` once at the beginning of each iteration and store its return value in a local variable. This value should then be used in the subsequent calls to `shouldDistribute` and `distributeDividend` within the same iteration. This change will reduce the number of calls to `getPendingUSDTEarnings` from two to one per shareholder per iteration, potentially halving the gas cost associated with this calculation and improving the overall efficiency of the contract.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	amal/contracts/AMALToken.sol#L171,185,193,208,225,234,252,262,270
Status	Renounced

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 burnBots(address from, uint256 amount) external onlyOwner
function disableBurnBotsFunctionality() public onlyOwner
function setIsExcludeFromTaxFee(
        address holder,
        bool _status
    ) external onlyOwner
function setIsDividendExclude(
        address holder,
        bool _status
    ) external onlyOwner
function setDistributorSettings(uint256 gas) external onlyOwner
function updateMaxSwapLimit(uint256 _swapLimit) external onlyOwner
function updateDividendDistributor(address newAddress) public
onlyOwner
function updateCharityAddress(address _newAddress) public onlyOwner
function updateMaxWalletLimit(uint256 _walletLimit) 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.

Team Update

The contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link: https://bscscan.com/tx/0xc21b7ea4b0bab1edecd05011ec270c6242eb35e97d785b1e822cc4679074df5f

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	amal/contracts/DividendDistributor.sol#L69 amal/contracts/AMALToken.sol#L81,85
Status	Acknowledged

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.

```
_token
slippage
pair
```

Recommendation

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

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	amal/contracts/DividendDistributor.sol#L61 amal/contracts/AMALToken.sol#L212
Status	Acknowledged

Description

The contract is using misleading error messages. These error messages do not 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(msg.sender == _token)
require(holder != address(this) && holder != pair)
```

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.

MPL - Missing Protocol Limits

Criticality	Minor / Informative
Location	amal/contracts/AMALToken.sol#L234,270
Status	Renounced

Description

It was observed that certain setter functions responsible for updating protocol variables lack essential checks to ensure that the assigned values fall within predefined limits or constraints. This deficiency in validation mechanisms poses potential risks to the integrity of the contract.

The functions designated for updating protocol variables do not include any conditions or validation logic to verify whether the new values supplied as arguments adhere to predetermined thresholds or constraints.

```
function updateMaxSwapLimit(uint256 _swapLimit) external
onlyOwner
function updateMaxWalletLimit(uint256 _walletLimit) external
onlyOwner
```

Recommendation

The team is advised to implement range checks within the setter functions to validate input values against predefined limits for the respective protocol variables. Utilize appropriate validation techniques, such as require statements or modifiers, to enforce constraints on assigned values and prevent unauthorized modifications beyond specified thresholds.

Team Update

The contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link: https://bscscan.com/tx/0xc21b7ea4b0bab1edecd05011ec270c6242eb35e97d785b1e822cc4679074df5f

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	amal/contracts/DividendDistributor.sol#L103,139
Status	Acknowledged

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
address[] memory path = new address[](2);
path[0] = _token;
path[1] = address(USDT);
...
router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
    _contractBal,
    _amountOutMin,
    path,
    address(this),
    block.timestamp
);
...
address[] memory buyBackPath = new address[](2);
buyBackPath[0] = address(USDT);
buyBackPath[1] = _token;
...
router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
    buyBackBurnAmount,
    _amountOutBuyBackMin,
    buyBackPath,
    address(this),
    block.timestamp
);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	amal/contracts/AMALToken.sol#L139
Status	Acknowledged

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `maxSwapLimit` 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.

```
if (from != pair && maxSwapLimit <= _feesOnContract) {
    super._transfer(
        address(this),
        address(distributor),
        balanceOf(address(this))
    );
    try
        distributor.deposit(
            charityTax,
            buyBackBurnTax,
            rewardDistributionTax,
            charityAddress
        )
```

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.

RCS - Redundant Comment Segments

Criticality	Minor / Informative
Location	amal/contracts/AMALToken.sol#L269
Status	Acknowledged

Description

The contract contains segments of code that are commented out. While commented code can serve as documentation or indicate future development plans, in this case, the commented-out code segments do not provide meaningful documentation or context. Instead, they introduce ambiguity regarding the intended features and current state of the contract. This could lead to confusion about the contract's capabilities and operational logic. The presence of such code may also suggest incomplete features or tentative updates that have not been fully implementated.

```
// Comment explaining the purpose of updateMaxWalletLimit
method
```

Recommendation

It is recommended to remove the segments of commented-out code from the contract. This will improve the clarity and readability of the contract's codebase, ensuring that it accurately reflects the implemented and active functionalities. Otherwise if certain commented-out segments are placeholders for future development, they should be replaced with clear documentation outlining the intended features.

RPC - Redundant Period Calculation

Criticality	Minor / Informative
Location	amal/contracts/DividendDistributor.sol#L185
Status	Acknowledged

Description

The contract is currently employing the `shouldDistribute` function to decide if a distribution should be made to a shareholder. This decision is based on a time check where `shareholderClaims[shareholder] + minPeriod` is compared against the current block timestamp. However, the `minPeriod` is initialized to zero and remains unchanged throughout the contract, rendering the time comparison redundant. This not only complicates the code but potentially leads to unnecessary processing and confusion regarding the intent of the time condition.

```
uint256 public minPeriod = 0;

function shouldDistribute(
    address shareholder
) internal view returns (bool) {
    return
        shareholderClaims[shareholder] + minPeriod <
block.timestamp &&
        getPendingUSDTEarnings(shareholder) > minDistribution;
}
```

Recommendation

It is recommended to refactor the code segment related to the return condition in the `shouldDistribute` function since the `minPeriod` is consistently zero, making the addition operation superfluous. Removing or updating this condition could simplify the contract and improve its efficiency by eliminating an unnecessary calculation. If `minPeriod` is intended to be used in the future, implementing a mechanism to update its value or documenting its purpose and future usage would also be beneficial.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	https:/github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.9/contracts/utils/math/SafeMath.sol amal/contracts/DividendDistributor.sol
Status	Acknowledged

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/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	amal/contracts/AMALToken.sol#L185
Status	Acknowledged

Description

The contract modifies the state of the following variables without checking if their current value is the same as the one given as an argument. As a result, the contract performs redundant storage writes, when the provided parameter matches the current state of the variables, leading to unnecessary gas consumption and inefficiencies in contract execution.

```
function disableBurnBotsFunctionality() public onlyOwner {
    isBurnBotDisable = true;

    // Emit an event to log of Disable Burn Bots
    emit DisableBurnBotsUpdate(isBurnBotDisable);
}
```

Recommendation

The team is advised to implement additional checks within to prevent redundant storage writes when the provided argument matches the current state of the variables. By incorporating statements to compare the new values with the existing values before proceeding with any state modification, the contract can avoid unnecessary storage operations, thereby optimizing gas usage.

RVD - Redundant Variable Declaration

Criticality	Minor / Informative
Location	amal/contracts/AMALToken.sol#L153
Status	Acknowledged

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 declares some variables that are not used in a meaningful way from the contract. As a result, these variables are redundant.

```
_feesOnContract = 0;
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	amal/contracts/DividendDistributor.sol#L28,51,53,55
Status	Acknowledged

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.

```
IERC20 USDT =
IERC20(0x8AC76a51cc950d9822D68b83fE1Ad97B32Cd580d)
uint256 public dividendsPerShareAccuracyFactor = 10 ** 36
uint256 public minPeriod = 0
uint256 public minDistribution = 0
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	https:/github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.9/contracts/utils/Context.sol#L25 https:/github.com/OpenZeppelin/openzeppelin-contracts/blob/release-v4.9/contracts/token/ERC20/ERC20.sol#L251,277
Status	Acknowledged

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 _contextSuffixLength() internal view virtual returns
(uint256) {
        return 0;
    }

...
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	amal/contracts/DividendDistributor.sol#L69
Status	Acknowledged

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.

```
_token = _tokenAddress
```

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	amal/contracts/DividendDistributor.sol#L123,201
Status	Acknowledged

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(USDT).transfer(
            _charityAddress,
            (amount * _charityTax) / totalTax
        )
USDT.transfer(shareholder, amount)
```

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**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
	_contextSuffixLength	Internal		
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		

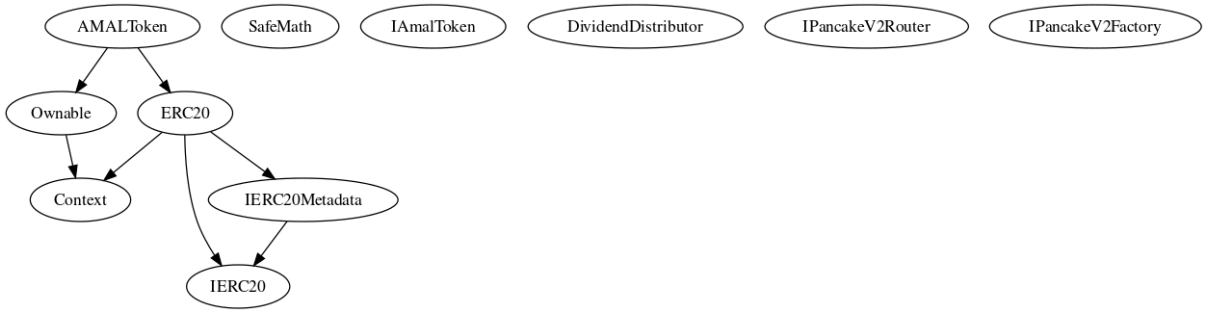
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
ERC20	Implementation	Context, IERC20, IERC20Metadata		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-

	_transfer	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_spendAllowance	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
	_afterTokenTransfer	Internal	✓	
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IAmalToken	Interface			
	buyBackBurnTokens	External	✓	-

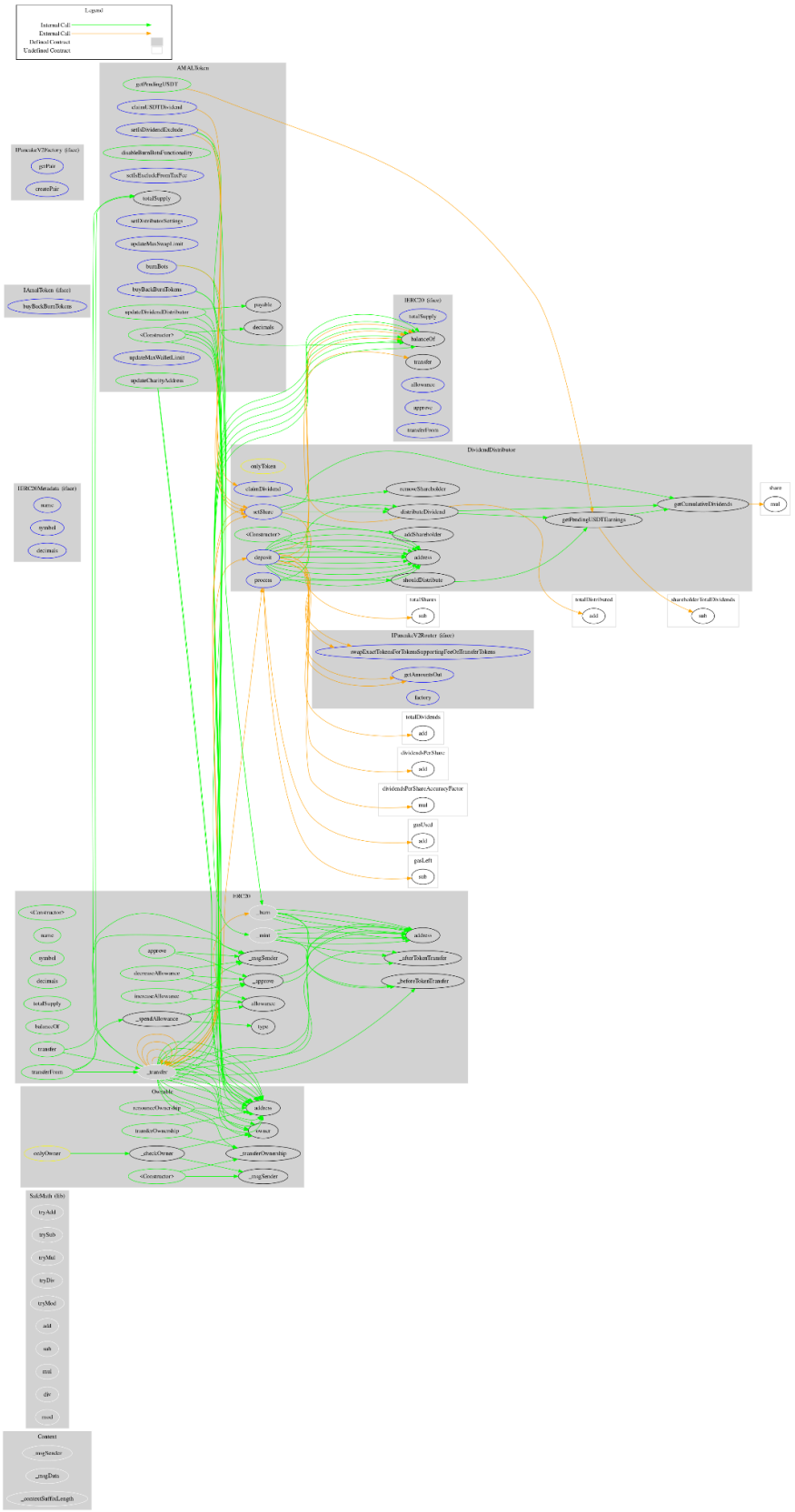
DividendDistributor	Implementation			
		Public	✓	-
	setShare	External	✓	onlyToken
	deposit	External	✓	onlyToken
	process	External	✓	onlyToken
	shouldDistribute	Internal		
	distributeDividend	Internal	✓	
	claimDividend	External	✓	-
	getPendingUSDTEarnings	Public		-
	getCumulativeDividends	Internal		
	addShareholder	Internal	✓	
	removeShareholder	Internal	✓	
AMALToken	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	_transfer	Internal	✓	
	burnBots	External	✓	onlyOwner
	buyBackBurnTokens	External	✓	-
	disableBurnBotsFunctionality	Public	✓	onlyOwner
	setIsExcludeFromTaxFee	External	✓	onlyOwner
	setIsDividendExclude	External	✓	onlyOwner
	setDistributorSettings	External	✓	onlyOwner

	updateMaxSwapLimit	External	✓	onlyOwner
	claimUSDTDividend	External	✓	-
	getPendingUSDT	Public		-
	updateDividendDistributor	Public	✓	onlyOwner
	updateCharityAddress	Public	✓	onlyOwner
	updateMaxWalletLimit	External	✓	onlyOwner
IPancakeV2Router	Interface			
	getAmountsOut	External		-
	factory	External		-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
IPancakeV2Factory	Interface			
	getPair	External		-
	createPair	External	✓	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
AMALToken
getPendingUSDT
claimUSDTDividend
setIsDividendExempt
disableAllFeesAndTransactionality
setIsExcludedFromFee
totalSupply
setDistributorSettings
updateMaxSwapLimit
burnBots
buyBackBurnTokens
updateDividendDistributor
<Constructor>
updateMaxWalletLimit
updateCharityAddress
payable
decimals
IAmalToken (iface)
IPancakeV2Factory (iface)
getPair
createPair
IERC20 (iface)
balanceOf
transfer
allowance
approve
transferFrom
IERC20Metadata (iface)
name
symbol
DividendDistributor
onlyToken
claimDividend
setShare
deposit
process
removeShareholder
distributeDividend
addShareholder
shouldDistribute
getCumulativeDividends
share
getPendingUSDTEarnings
totalShares
totalDistributed
shareholderTotalDividends
IPancakeV2Router (iface)
swapExactTokensForTokensSupportingFeeOnTransferTokens
getAmountsOut
factory
totalDividends
dividendsPerShare
dividendsPerShareAccuracyFactor
gasUsed
gasLeft
ERC20
burn
mint
approve
decreaseAllowance
increaseAllowance
_spendAllowance
_transfer
_msgSender
_approve
_afterTokenTransfer
_beforeTokenTransfer
Ownable
renounceOwnership
transferOwnership
onlyOwner
_checkOwner
_transferOwnership
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod
Context
_msgData
_contextSuffixLength

Summary

AMAL is an interesting project that has a friendly and growing community. The contract's ownership has been renounced. This audit investigates security issues, business logic concerns and potential improvements.

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