Cyberscope

# Audit Report

# MegaCoin

February 2023

|  |  |
|---|---|
| Type | BEP20 |
| Network | BSC |
| Address | 0x740293131c635b401b3C1aD5568306334b717729 |
| Audited by | © cyberscope |

# Table of Contents

# Review

| | |
|---|---|
| **Contract Name** | MegaCoin |
| **Compiler Version** | v0.6.12+commit.27d51765 |
| **Optimization** | 200 runs |
| **Explorer** | https://bscscan.com/address/0x740293131c635b401b3c1ad5568306334b717729 |
| **Address** | 0x740293131c635b401b3c1ad5568306334b717729 |
| **Network** | BSC |
| **Symbol** | MTG |
| **Decimals** | 3 |
| **Total Supply** | 1,000,000,000 |

# Audit Updates

| | |
|---|---|
| **Initial Audit** | 22 Feb 2023 |

# Source Files

| **Filename** | SHA256 |
|---|---|
| **MegaCoin.sol** | 519c27a7c9328e6610abd5c6eb5f10b12a701ae4cbc9d753a9b2eebae2e865f3 |

# 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 |
|---|---|---|---|
| ● | MFN | Misleading Function Naming | Unresolved |
| 🔴 | TSD | Total Supply Diversion | Unresolved |
| 🔴 | IRF | Insufficient Reward Funds | Unresolved |
| ● | PTRP | Potential Transfer Revert Propagation | Unresolved |
| ● | DDP | Decimal Division Precision | Unresolved |
| ● | CO | Code Optimization | Unresolved |
| ● | L02 | State Variables could be Declared Constant | Unresolved |
| ● | L04 | Conformance to Solidity Naming Conventions | Unresolved |
| ● | L05 | Unused State Variable | Unresolved |
| ● | L09 | Dead Code Elimination | Unresolved |
| ● | L13 | Divide before Multiply Operation | Unresolved |
| ● | L16 | Validate Variable Setters | Unresolved |
| ● | L17 | Usage of Solidity Assembly | Unresolved |

| | | | |
|---|---|---|---|
| ● | L20 | Succeeded Transfer Check | Unresolved |

# MFN - Misleading Function Naming

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Status** | Unresolved |

## Description

Functions can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some function names that do not clearly reflect the functions purpose and functionality. Misleading function names can lead to confusion, making the code more difficult to read and understand.

The function `_reflectFee` adds the fee amount of a transaction to the `_tFeeTotal` variable, which is the accumulated amount of fees collected by the contract. This functions name does not reflect its functionality in any way.

```
function _reflectFee(uint256 tFee) private {
    _tFeeTotal = _tFeeTotal.add(tFee);
}
```

## Recommendation

It's always a good practice for the contract to contain function names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

# TSD - Total Supply Diversion

| | |
|---|---|
| **Criticality** | Critical |
| **Location** | MegaCoin.sol#L653 |
| **Status** | Unresolved |

## Description

The total supply of a token is the total number of tokens that have been created, while the balances of individual accounts represent the number of tokens that an account owns. The total supply and the balances of individual accounts are two separate concepts that are managed by different variables in a smart contract. These two entities should be equal to each other.

Within the contract, fees are partitioned into 5 distinct portions, with each portion being allocated to a corresponding wallet's balance. The precise amount of each portion is calculated using a percentage value. However, the sum of all percentage values is 9, with a divisor of 10. This arrangement results in a small fraction of the fee being excluded from the balance calculations. Consequently, the sum of all balances will not match the total supply.

```
MarketingShare=tFee.mul(_marketingPer).div(10);
RewardShare=tFee.mul(_RewardPer).div(10);
Buyback=tFee.mul(BuybackPer).div(10);
devShare=tFee.mul(_devPer).div(10);
lotShare=tFee.mul(_lotPer).div(10);
```

## Recommendation

The total supply and the balance variables are separate and independent from each other. The total supply represents the total number of tokens that have been created, while the balance mapping stores the number of tokens that each account owns. The sum of balances should always equal the total supply.

# IRF - Insufficient Reward Funds

| | |
|---|---|
| **Criticality** | Critical |
| **Location** | MegaCoin.sol#L555 |
| **Status** | Unresolved |

## Description

The reward pool in the contract can be used by users to claim rewards. Unfortunately, due to bugs in the reward formula, some users are unable to claim their rewards. To illustrate this issue, consider the following scenario:

- User A initiates a transfer, resulting in a reward pool of x.
- User B then transfers, increasing the pool to x + y.
- Finally, User C transfers, bringing the reward pool to x + y + z.
- When User B claims their reward, the user is able to take z, leaving x + y in the pool.
- Subsequently, when User A claims reward, they user can take y + z, which will not be sufficient.

```
function claimReward() public {
    if(msg.sender!=pancakePair)
    {
        uint256 rewardPool=UserrewardPoolOnLastClaim[msg.sender];
        emit comments ("URP",  UserrewardPoolOnLastClaim[msg.sender]);
        emit comments ("TRP",_rewardPool);
        emit comments ("AP",rewardPool);
        uint256 remainPool=_rewardPool-rewardPool;
        emit comments ("RP" , remainPool);

        if(remainPool>0 && balanceOf(msg.sender)>0 && exist[msg.sender]){
            uint256 userShare =
(balanceOf(msg.sender).mul(remainPool)).div(totalSupply());
            emit comments("balanceOf",balanceOf(msg.sender));
            emit comments ("rP",remainPool);
            emit comments ("TS",totalSupply());
            emit comments("uS line 595S", userShare);
            payable(msg.sender).transfer(userShare);
            _claimedRewardPool+=userShare;
            emit comments("_clad 597", _claimedRewardPool);
        }
        UserrewardPoolOnLastClaim[msg.sender]=_rewardPool;
        emit comments ("Uclaimed", UserrewardPoolOnLastClaim[msg.sender]);
        emit comments ("rP",_rewardPool);
    }
}
```

## Recommendation

The team is advised to carefully check the implementation of the `claimReward` function to ensure that all users are able to claim their share.

# PTRP - Potential Transfer Revert Propagation

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | MegaCoin.sol#L527 |
| **Status** | Unresolved |

## Description

The contract sends funds to the following address

- `markWallet`
- `buybackWallet`
- `devWallet`
- `lotteryWallet`

as part of the transfer flow. These addresses 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.

```
markWallet.transfer(MarketingBNB);
buybackWallet.transfer(BuybackBNB);
devWallet.transfer(devBNB);
lotteryWallet.transfer(lotBNB);
```

## 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 archived by not allowing set contract addresses or by sending the funds in a non-revertable way.

# DDP - Decimal Division Precision

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | MegaCoin.sol#L521 |
| **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.

```
uint256 SplitBNBBalance =
Balance.div(_marketingPer.add(_RewardPer).add(BuybackPer).add(_devPer).add(_lotPer));
uint256  MarketingBNB=SplitBNBBalance.mul(_marketingPer);
uint256  BuybackBNB=SplitBNBBalance.mul(BuybackPer);
uint256  devBNB=SplitBNBBalance.mul(_devPer);
uint256  lotBNB=SplitBNBBalance.mul(_lotPer);
uint256  RewardBNB=SplitBNBBalance.mul(_RewardPer);
```

## Recommendation

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

# CO - Code Optimization

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | MegaCoin.sol#L485 |
| **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.

During a transaction the contract checks the sender's and recipient's address if they match `pancakePair` address. Since the previous checks at the `if-else if` block already check equality between these addresses, the last check is redundant as it will always be true. Hence, an `else` block will be sufficient.

```
else if(from != pancakePair && to != pancakePair)
{
    takeFee = false;
    TaxType=0;
}
```

## Recommendation

The team is advised to take into consideration these segments 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** | MegaCoin.sol#L117,344,345,346,354,355,356,357,358,368 |
| **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.

```
address private _previousOwner
string  private _name = "Mega Coin"
string  private _symbol = "MTG"
uint8  private _decimals = 3
uint256 public  _marketingPer = 3
uint256 public  _RewardPer = 2
uint256 public  BuybackPer=1
uint256 public _devPer=2
uint256 public _lotPer=1
uint256 private minTokensBeforeSwap = 100
```

## 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** | MegaCoin.sol#L172,173,187,205,341,348,349,350,351,352,354,355,356,357,358,370,372,543,580,599,643,713,718,722,727,735,741,748,752,756 |
| **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);
address[] private _ExcludedFromReward
mapping (address => uint) public UserLastSellTimeStamp
mapping (address => uint256) public UserrewardPoolOnLastClaim
uint256 public  _rewardPool
uint256 public  _claimedRewardPool
uint256 public  _TaxFee = 9
uint256 public  _marketingPer = 3
uint256 public  _RewardPer = 2
uint256 public  BuybackPer=1
uint256 public _devPer=2

...
```

## 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** | MegaCoin.sol#L117,341 |
| **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.

```
address private _previousOwner
address[] private _ExcludedFromReward
```

## 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.

# L09 - Dead Code Elimination

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | MegaCoin.sol#L70,77,82,85,88,91,95 |
| **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 isContract(address account) internal view returns (bool) {
        bytes32 codehash;
        bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
        //solhint-disable-next-line no-inline-assembly
        assembly { codehash := extcodehash(account) }
        return (codehash != accountHash && codehash != 0x0);
    }

function sendValue(address payable recipient, uint256 amount) internal {
        require(address(this).balance >= amount, "Address: insufficient balance");
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value, recipient may have
reverted");
    }

...
```

## 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** | MegaCoin.sol#L521,522,523,524,525,526 |
| **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.

```
uint256 SplitBNBBalance =
Balance.div(_marketingPer.add(_RewardPer).add(BuybackPer).add(_devPer).add(_lotPer))
uint256  lotBNB=SplitBNBBalance.mul(_lotPer)
```

## 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.

# L16 - Validate Variable Setters

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | MegaCoin.sol#L687,714,719,723,728,737,749,753,757 |
| **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.

```
takeMain=payable(addre)
markWallet = wallet
buybackWallet = wallt
devWallet = wallet
lotteryWallet = wallt
_toAddtress.transfer(address(this).balance)
RewardHolder.transfer(RewardAmnt)
DivRewardHolder.transfer(RewardAmnt)
holder.transfer(RewardAmnt)
```

## 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.

# L17 - Usage of Solidity Assembly

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | MegaCoin.sol#L74,102 |
| **Status** | Unresolved |

## Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { codehash := extcodehash(account) }

assembly {
                    let returndata_size := mload(returndata)
                    revert(add(32, returndata), returndata_size)
                }
```

## Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

# L20 - Succeeded Transfer Check

| | |
|---|---|
| **Criticality** | Minor / Informative |
| **Location** | MegaCoin.sol#L745 |
| **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.

```
tokenContract.transferFrom(address(this), buybackWallet, _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 | | |
| | | | | |
| **IERC20** | Interface | | | |
| | totalSupply | External | | - |
| | balanceOf | External | | - |
| | transfer | External | ✓ | - |
| | allowance | External | | - |
| | approve | External | ✓ | - |
| | transferFrom | External | ✓ | - |
| | | | | |
| **SafeMath** | Library | | | |
| | add | Internal | | |
| | sub | Internal | | |
| | sub | Internal | | |
| | mul | Internal | | |
| | div | Internal | | |
| | div | Internal | | |
| | mod | Internal | | |
| | mod | Internal | | |
| | | | | |
| **Address** | Library | | | |
| | isContract | Internal | | |

| | | | | |
|---|---|---|---|---|
| | sendValue | Internal | ✓ | |
| | functionCall | Internal | ✓ | |
| | functionCall | Internal | ✓ | |
| | functionCallWithValue | Internal | ✓ | |
| | functionCallWithValue | Internal | ✓ | |
| | _functionCallWithValue | Private | ✓ | |
| | | | | |
| **Ownable** | Implementation | Context | | |
| | | Internal | ✓ | |
| | owner | Public | | - |
| | renounceOwnership | Public | ✓ | onlyOwner |
| | transferOwnership | Public | ✓ | onlyOwner |
| | | | | |
| **IPancakeFactory** | Interface | | | |
| | feeTo | External | | - |
| | feeToSetter | External | | - |
| | getPair | External | | - |
| | allPairs | External | | - |
| | allPairsLength | External | | - |
| | createPair | External | ✓ | - |
| | setFeeTo | External | ✓ | - |
| | setFeeToSetter | External | ✓ | - |
| | | | | |
| **IPancakePair** | Interface | | | |
| | name | External | | - |
| | symbol | External | | - |
| | decimals | External | | - |
| | totalSupply | External | | - |
| | balanceOf | External | | - |

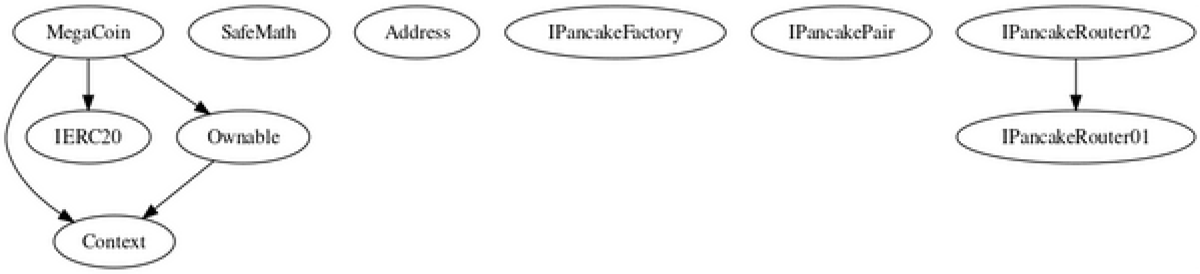
| | | | | |
|---|---|---|---|---|
| | allowance | External | | - |
| | approve | External | ✓ | - |
| | transfer | External | ✓ | - |
| | transferFrom | External | ✓ | - |
| | DOMAIN_SEPARATOR | External | | - |
| | PERMIT_TYPEHASH | External | | - |
| | nonces | External | | - |
| | permit | External | ✓ | - |
| | MINIMUM_LIQUIDITY | External | | - |
| | factory | External | | - |
| | token0 | External | | - |
| | token1 | External | | - |
| | getReserves | External | | - |
| | price0CumulativeLast | External | | - |
| | price1CumulativeLast | External | | - |
| | kLast | External | | - |
| | mint | External | ✓ | - |
| | burn | External | ✓ | - |
| | swap | External | ✓ | - |
| | skim | External | ✓ | - |
| | sync | External | ✓ | - |
| | initialize | External | ✓ | - |
| | | | | |
| **IPancakeRouter01** | Interface | | | |
| | factory | External | | - |
| | WETH | External | | - |
| | addLiquidity | External | ✓ | - |
| | addLiquidityETH | External | Payable | - |
| | removeLiquidity | External | ✓ | - |

| | | | | |
|---|---|---|---|---|
| | removeLiquidityETH | External | ✓ | - |
| | removeLiquidityWithPermit | External | ✓ | - |
| | removeLiquidityETHWithPermit | External | ✓ | - |
| | swapExactTokensForTokens | External | ✓ | - |
| | swapTokensForExactTokens | External | ✓ | - |
| | swapExactETHForTokens | External | Payable | - |
| | swapTokensForExactETH | External | ✓ | - |
| | swapExactTokensForETH | External | ✓ | - |
| | swapETHForExactTokens | External | Payable | - |
| | quote | External | | - |
| | getAmountOut | External | | - |
| | getAmountIn | External | | - |
| | getAmountsOut | External | | - |
| | getAmountsIn | External | | - |
| | | | | |
| **IPancakeRouter02** | Interface | IPancakeRouter01 | | |
| | removeLiquidityETHSupportingFeeOnTransferTokens | External | ✓ | - |
| | removeLiquidityETHWithPermitSupportingFeeOnTransferTokens | External | ✓ | - |
| | swapExactTokensForTokensSupportingFeeOnTransferTokens | External | ✓ | - |
| | swapExactETHForTokensSupportingFeeOnTransferTokens | External | Payable | - |
| | swapExactTokensForETHSupportingFeeOnTransferTokens | External | ✓ | - |
| | | | | |
| **MegaCoin** | Implementation | Context, IERC20, Ownable | | |
| | | Public | ✓ | - |
| | name | Public | | - |
| | symbol | Public | | - |

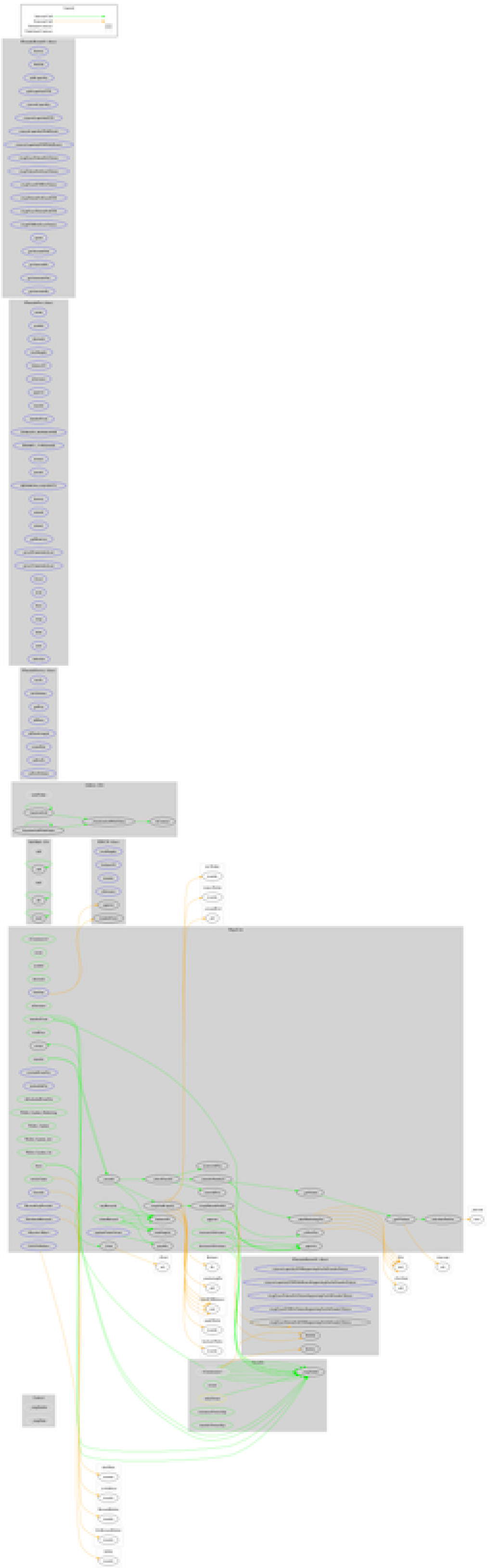
| | | | | |
|---|---|---|---|---|
| | decimals | Public | | - |
| | totalSupply | Public | | - |
| | balanceOf | Public | | - |
| | transfer | Public | ✓ | - |
| | allowance | Public | | - |
| | approve | Public | ✓ | - |
| | transferFrom | Public | ✓ | - |
| | increaseAllowance | Public | ✓ | - |
| | decreaseAllowance | Public | ✓ | - |
| | totalFees | Public | | - |
| | _approve | Private | ✓ | |
| | _transfer | Private | ✓ | |
| | swapAndLiquify | Private | ✓ | |
| | myRewards | Public | | - |
| | claimReward | Public | ✓ | - |
| | swapTokensForEth | Private | ✓ | |
| | _tokenTransfer | Private | ✓ | |
| | _transferStandard | Private | ✓ | |
| | _getValues | Private | | |
| | _getTValues | Private | | |
| | calculateTaxFee | Private | | |
| | _takeMarketingFee | Private | ✓ | |
| | _reflectFee | Private | ✓ | |
| | resetAllFee | Private | ✓ | |
| | restoreAllFee | Private | ✓ | |
| | verifyClaim | Public | ✓ | - |
| | updateClaimOwner | External | ✓ | onlyOwner |
| | burn | Public | ✓ | - |
| | excludeFromFee | External | ✓ | onlyOwner |

| | | | | |
|---|---|---|---|---|
| | includeInFee | External | ✓ | onlyOwner |
| | isExcludedFromFee | Public | | - |
| | _burn | Internal | ✓ | |
| | Wallet_Update_Marketing | Public | ✓ | onlyOwner |
| | Wallet_Update | Public | ✓ | onlyOwner |
| | Wallet_Update_dev | Public | ✓ | onlyOwner |
| | Wallet_Update_lot | Public | ✓ | onlyOwner |
| | GetAlls | External | ✓ | onlyOwner |
| | GetOne | External | ✓ | onlyOwner |
| | RewardLiqProvider | External | ✓ | onlyOwner |
| | DividendsRewards | External | ✓ | onlyOwner |
| | GiftsToHolders | External | ✓ | onlyOwner |
| | | External | Payable | - |

# Inheritance Graph



# Flow Graph



# Summary

MegaCoin is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler errors. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions. The analysis reported two critical issues, regarding the total supply and balances diversion, as described in detail at the TSD section, and the rewards calculation formula, as described in detail at the IRF section. There is also a limit of max 9% 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

https://www.cyberscope.io