Cyberscope

Audit Report

Marvin Inu

July 2023

Network	ETH
Address	0x55a380d134d722006A5CE2d510562e1239D225B1
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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Unresolved
●	BC	Blacklists Addresses	Unresolved

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	CR	Code Repetition	Unresolved
	AOI	Arithmetic Operations Inconsistency	Unresolved
	TUU	Time Units Usage	Unresolved
	DKO	Delete Keyword Optimization	Unresolved
	DDP	Decimal Division Precision	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	MCM	Misleading Comment Messages	Unresolved
	RSML	Redundant SafeMath Library	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
	L08	Tautology or Contradiction	Unresolved
	L09	Dead Code Elimination	Unresolved

●	L13	Divide before Multiply Operation	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L16	Validate Variable Setters	Unresolved

Table of Contents

Review

Contract Name	MarvinInu
Compiler Version	v0.8.9+commit.e5eed63a
Optimization	200 runs
Explorer	https://etherscan.io/address/0x55a380d134d722006a5ce2d510562e1239d225b1
Address	0x55a380d134d722006a5ce2d510562e1239d225b1
Network	ETH
Symbol	MARVIN
Decimals	18
Total Supply	1,000,000,000,000

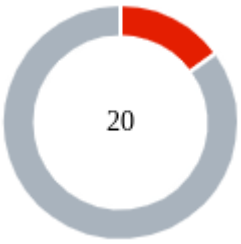
Audit Updates

Initial Audit	19 Jul 2023

Source Files

Filename	SHA256
MarvinInu.sol	3658206bdd9b058cd1140845554a0ad165dc5d8c5c3533359154fcd7802c042e

Findings Breakdown



Critical	3
Medium	0
Minor / Informative	17

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	3	0	0	0
Medium	0	0	0	0
Minor / Informative	17	0	0	0

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	MarvinInu.sol#L1060,1068
Status	Unresolved

Description

The contract owner has the authority to increase over the allowed limit of 25%. The owner may take advantage of it by calling the `updateBuyFees` and `updateSellFees` functions with a high percentage value.

```
   function updateBuyFees(uint256 _marketingFee, uint256
_liquidityFee, uint256 _devFee) external onlyOwner {
        buyMarketingFee = _marketingFee;
        buyLiquidityFee = _liquidityFee;
        buyDevFee = _devFee;
        buyTotalFees = buyMarketingFee + buyLiquidityFee +
buyDevFee;
        require(buyTotalFees <= 50, "Must keep fees at 50% or
less");
    }

    function updateSellFees(uint256 _marketingFee, uint256
_liquidityFee, uint256 _devFee) external onlyOwner {
        sellMarketingFee = _marketingFee;
        sellLiquidityFee = _liquidityFee;
        sellDevFee = _devFee;
        sellTotalFees = sellMarketingFee + sellLiquidityFee +
sellDevFee;
        require(sellTotalFees <= 50, "Must keep fees at 50% or
less");
    }
```

Recommendation

The contract could embody a check for the maximum acceptable value. The team should carefully manage the private keys of the owner's account. We strongly recommend a

powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

BT - Burns Tokens

Criticality	Critical
Location	MarvinInu.sol#L1183,1302
Status	Unresolved

Description

The contract automatically burns up to 10% of the liquidity pair token balance every 10 minutes at most. If a large amount of liquidity is removed from the pool through burning, it can cause a decrease in the liquidity of the pool, which can, in turn, result in increased volatility and price fluctuations of the tokens in the pair.

```
if(!swapping && automatedMarketMakerPairs[to] && lpBurnEnabled
&& block.timestamp >= lastLpBurnTime + lpBurnFrequency &&
!_isExcludedFromFees[from]){
      autoBurnLiquidityPairTokens();
      }


    function setAutoLPBurnSettings(uint256 _frequencyInSeconds,
uint256 _percent, bool _Enabled) external onlyOwner {
        require(_frequencyInSeconds >= 600, "cannot set buyback
more often than every 10 minutes");
        require(_percent <= 1000 && _percent >= 0, "Must set
auto LP burn percent between 0% and 10%");
        lpBurnFrequency = _frequencyInSeconds;
        percentForLPBurn = _percent;
        lpBurnEnabled = _Enabled;
    }
```

Recommendation

It is recommended to review and adjust the parameters of the auto-liquidity burn mechanism to ensure that it operates optimally. Specifically, the period of time and percentage burned should be reasonable and appropriate for the specific use case. This will help to prevent any potential issues and ensure that the mechanism functions as intended.

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

BC - Blacklists Addresses

Criticality	Critical
Location	MarvinInu.sol#L1011,1117
Status	Unresolved

Description

The contract owner has the authority to stop addresses from transactions. The owner may take advantage of it by calling the `addSnipers` function.

```
   function addSnipers(address[] calldata accounts) external
onlyOwner {
        uint256 len = accounts.length;
        for(uint256 i = 0; i < len;) {
            _sniper[accounts[i]] = true;
            unchecked {
                i++;
            }
        }
    }

function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal override {
        require(!_sniper[from], "Sniper rejected");
        ...
    }
```

Recommendation

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.

- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

CR - Code Repetition

Criticality	Minor / Informative
Location	MarvinInu.sol#L1198
Status	Unresolved

Description

The contract contains repetitive code segments. There are potential issues that can arise when using code segments in Solidity. Some of them can lead to issues like gas efficiency, complexity, readability, security, and maintainability of the source code. It is generally a good idea to try to minimize code repetition where possible.

```
    // on sell
    if (automatedMarketMakerPairs[to] && sellTotalFees > 0){
        fees = amount.mul(sellTotalFees).div(100);
        tokensForLiquidity += fees * sellLiquidityFee /
sellTotalFees;
        tokensForDev += fees * sellDevFee / sellTotalFees;
        tokensForMarketing += fees * sellMarketingFee /
sellTotalFees;
    }
    // on buy
    else if(automatedMarketMakerPairs[from] && buyTotalFees >
0) {
        fees = amount.mul(buyTotalFees).div(100);
        tokensForLiquidity += fees * buyLiquidityFee /
buyTotalFees;
        tokensForDev += fees * buyDevFee / buyTotalFees;
        tokensForMarketing += fees * buyMarketingFee /
buyTotalFees;
    }
```

```
   function autoBurnLiquidityPairTokens() internal returns
(bool){

        lastLpBurnTime = block.timestamp;

        // get balance of liquidity pair
        uint256 liquidityPairBalance =
this.balanceOf(uniswapV2Pair);

        // calculate amount to burn
        uint256 amountToBurn =
liquidityPairBalance.mul(percentForLPBurn).div(10000);

        // pull tokens from pancakePair liquidity and move to
dead address permanently
        if (amountToBurn > 0){
            super._transfer(uniswapV2Pair, address(0xdead),
amountToBurn);
        }

        //sync price since this is not in a swap transaction!
        IUniswapV2Pair pair = IUniswapV2Pair(uniswapV2Pair);
        pair.sync();
        emit AutoNukeLP();
        return true;
    }

    function manualBurnLiquidityPairTokens(uint256 percent)
external onlyOwner returns (bool){
        ...

        lastManualLpBurnTime = block.timestamp;

        // get balance of liquidity pair
        uint256 liquidityPairBalance =
this.balanceOf(uniswapV2Pair);
        ...
    }
```

Recommendation

The team is advised to avoid repeating the same code in multiple places, which can make the contract easier to read and maintain. The authors could try to reuse code wherever possible, as this can help reduce the complexity and size of the contract. For instance, the

contract could reuse the common code segments in an internal function in order to avoid repeating the same code in multiple places.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	MarvinInu.sol#L1272,1278,1284
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.

```
uint256 amountToSwapForETH =
contractBalance.sub(liquidityTokens);
...
uint256 ethBalance =
address(this).balance.sub(initialETHBalance);
...
uint256 ethForLiquidity = ethBalance - ethForMarketing -
ethForDev;
```

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.

TUU - Time Units Usage

Criticality	Minor / Informative
Location	MarvinInu.sol#L1303
Status	Unresolved

Description

The contract is using arbitrary numbers to form time-related values. As a result, it decreases the readability of the codebase and prevents the compiler to optimize the source code.

```
require(_frequencyInSeconds >= 600, "cannot set buyback more
often than every 10 minutes");
```

Recommendation

It is a good practice to use the time units reserved keywords like `seconds` , `minutes` , `hours` , `days` and `weeks` to process time-related calculations.

It's important to note that these time units are simply a shorthand notation for representing time in seconds, and do not have any effect on the actual passage of time or the execution of the contract. The time units are simply a convenience for expressing time in a more human-readable form.

DKO - Delete Keyword Optimization

Criticality	Minor / Informative
Location	MarvinInu.sol#L1287
Status	Unresolved

Description

The contract resets variables to the default state by setting the initial values. Setting values to state variables increases the gas cost.

```
    tokensForLiquidity = 0;
     tokensForMarketing = 0;
     tokensForDev = 0;
```

Recommendation

The team is advised to use the `delete` keyword instead of setting variables. This can be more efficient than setting the variable to a new value, using delete can reduce the gas cost associated with storing data on the blockchain.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	MarvinInu.sol#L1198
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.

```
// on sell
    if (automatedMarketMakerPairs[to] && sellTotalFees > 0){
        fees = amount.mul(sellTotalFees).div(100);
        tokensForLiquidity += fees * sellLiquidityFee /
sellTotalFees;
        tokensForDev += fees * sellDevFee / sellTotalFees;
        tokensForMarketing += fees * sellMarketingFee /
sellTotalFees;
    }
    // on buy
    else if(automatedMarketMakerPairs[from] && buyTotalFees >
0) {
        fees = amount.mul(buyTotalFees).div(100);
        tokensForLiquidity += fees * buyLiquidityFee /
buyTotalFees;
        tokensForDev += fees * buyDevFee / buyTotalFees;
        tokensForMarketing += fees * buyMarketingFee /
buyTotalFees;
```

```
    }
```

Recommendation

The team is advised to take into consideration the rounding results that are produced from the solidity calculations. The contract could calculate the subtraction of the divided funds in the last calculation in order to avoid the division rounding issue.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	MarvinInu.sol#L1005,1011,1051,1056,1076,1093,1098
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 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.

```
    function removeSnipers(address[] calldata accounts) external
onlyOwner {
        for (uint256 i = 0; i < accounts.length; i++) {
            _sniper[accounts[i]] = false;
        }
    }

    function addSnipers(address[] calldata accounts) external
onlyOwner {
        uint256 len = accounts.length;
        for(uint256 i = 0; i < len;) {
            _sniper[accounts[i]] = true;
            unchecked {
                i++;
            }
        }
    }

    function excludeFromMaxTransaction(address updAds, bool
isEx) public onlyOwner {
        _isExcludedMaxTransactionAmount[updAds] = isEx;
    }

    function updateSwapEnabled(bool enabled) external
onlyOwner(){
        swapEnabled = enabled;
    }

    function excludeFromFees(address account, bool excluded)
public onlyOwner {
        _isExcludedFromFees[account] = excluded;
        emit ExcludeFromFees(account, excluded);
    }

    function updateMarketingWallet(address newMarketingWallet)
external onlyOwner {
        emit marketingWalletUpdated(newMarketingWallet,
marketingWallet);
        marketingWallet = newMarketingWallet;
    }

    function updateDevWallet(address newWallet) external
onlyOwner {
        emit devWalletUpdated(newWallet, devWallet);
        devWallet = newWallet;
    }
```

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.

MCM - Misleading Comment Messages

Criticality	Minor / Informative
Location	MarvinInu.sol#L962
Status	Unresolved

Description

The contract is using misleading comment messages. These comment messages do not accurately reflect the actual implementation, making it difficult to understand the source code.

Specifically, the comment associated with the `maxWallet` variable suggests that it represents `1.5%` of the totalSupply. However, the `maxWallet` variable is actually equivalent to `2%` of the totalSupply, not the `1.5%` as indicated by the comment. As a result, the users will not comprehend the source code's actual implementation.

```
maxWallet = totalSupply * 20 / 1000; // 1.5% maxWallet
```

Recommendation

The team is advised to carefully review the comment in order to reflect the actual implementation. To improve code readability, the team should use more specific and descriptive comment messages.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	MarvinInu.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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MarvinInu.sol#L883
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 manualBurnFrequency = 30 minutes
```

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	MarvinInu.sol#L38,39,56,729,913,925,927,1060,1068,1302
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);
mapping (address => bool) public
_isExcludedMaxTransactionAmount
event marketingWalletUpdated(address indexed newWallet, address
indexed oldWallet);
event devWalletUpdated(address indexed newWallet, address
indexed oldWallet);
uint256 _marketingFee
uint256 _liquidityFee
uint256 _devFee
uint256 _frequencyInSeconds
uint256 _percent
bool _Enabled
```

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	MarvinInu.sol#L660
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	MarvinInu.sol#L1037,1043,1048,1061,1069,1305
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.

```
swapTokensAtAmount = newAmount
maxTransactionAmount = newNum * (10**18)
maxWallet = newNum * (10**18)
buyMarketingFee = _marketingFee
sellMarketingFee = _marketingFee
lpBurnFrequency = _frequencyInSeconds
```

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.

L08 - Tautology or Contradiction

Criticality	Minor / Informative
Location	MarvinInu.sol#L1304
Status	Unresolved

Description

A tautology is a logical statement that is always true, regardless of the values of its variables. A contradiction is a logical statement that is always false, regardless of the values of its variables.

Using tautologies or contradictions can lead to unintended behavior and can make the code harder to understand and maintain. It is generally considered good practice to avoid tautologies and contradictions in the code.

```
require(_percent <= 1000 && _percent >= 0, "Must set auto LP
burn percent between 0% and 10%")
```

Recommendation

The team is advised to carefully consider the logical conditions is using in the code and ensure that it is well-defined and make sense in the context of the smart contract.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	MarvinInu.sol#L405,706,712,719
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	MarvinInu.sol#L1200,1201,1202,1206,1207,1208,1209
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.

```
tokensForDev += fees * sellDevFee / sellTotalFees
fees = amount.mul(buyTotalFees).div(100)
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	MarvinInu.sol#L958
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.

```
uint256 totalSupply = 1 * 1e12 * 1e18
```

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	MarvinInu.sol#L1095,1100
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.

```
marketingWallet = newMarketingWallet
devWallet = newWallet
```

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.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IUniswapV2Pair	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	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-

	setFeeToSetter	External	✓	-
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	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	✓	
	_beforeTokenTransfer	Internal	✓	
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		

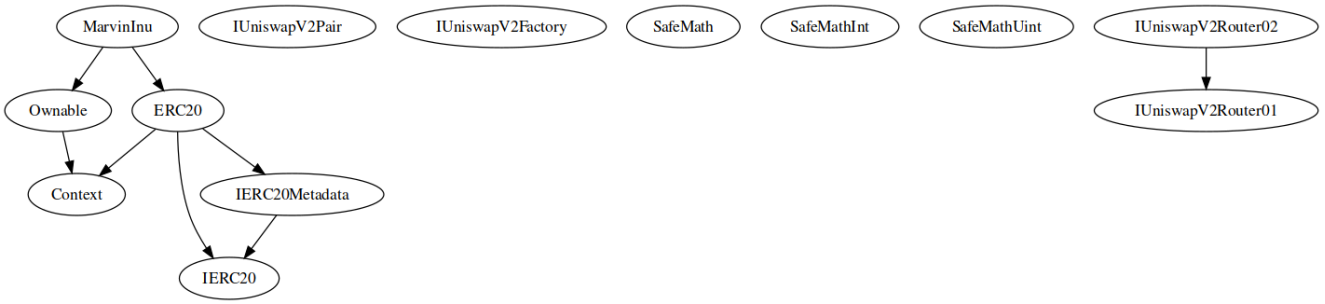
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
SafeMathInt	Library			
	mul	Internal		
	div	Internal		
	sub	Internal		
	add	Internal		
	abs	Internal		
	toUint256Safe	Internal		
SafeMathUint	Library			
	toInt256Safe	Internal		
IUniswapV2Router01	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		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
MarvinInu	Implementation	ERC20, Ownable		

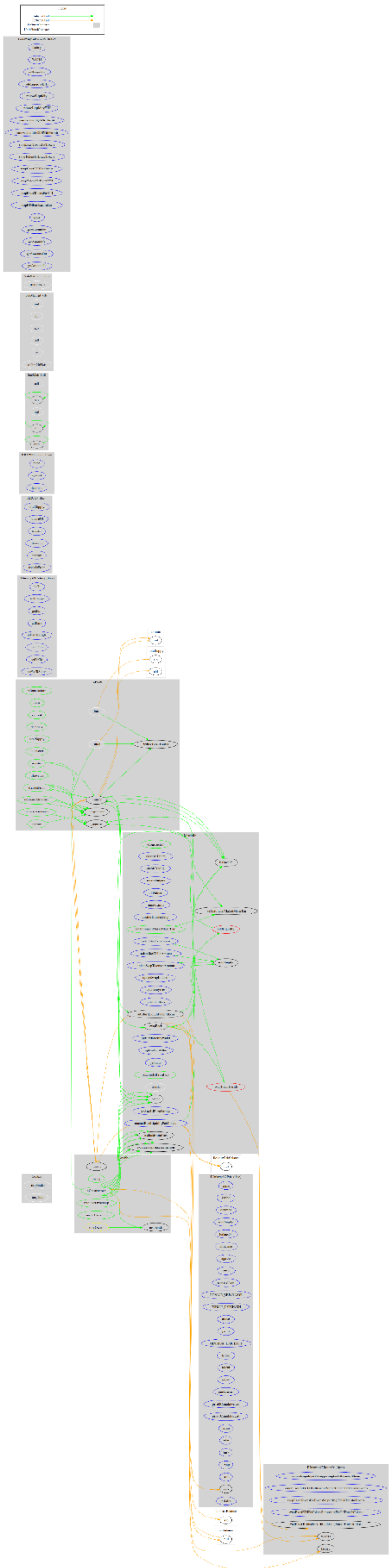
		Public	✓	ERC20
		External	Payable	-
	enableTrading	External	✓	onlyOwner
	removeSnipers	External	✓	onlyOwner
	addSnipers	External	✓	onlyOwner
	removeLimits	External	✓	onlyOwner
	disableTransferDelay	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	updateMaxTxnAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	excludeFromMaxTransaction	Public	✓	onlyOwner
	updateSwapEnabled	External	✓	onlyOwner
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	setAutomatedMarketMakerPair	Public	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateMarketingWallet	External	✓	onlyOwner
	updateDevWallet	External	✓	onlyOwner
	getStatus	External		-
	isExcludedFromFees	Public		-
	_transfer	Internal	✓	
	swapTokensForEth	Private	✓	

	addLiquidity	Private	✓	
	swapBack	Private	✓	
	setAutoLPBurnSettings	External	✓	onlyOwner
	autoBurnLiquidityPairTokens	Internal	✓	
	manualBurnLiquidityPairTokens	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Marvin inu contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. There are some functions that can be abused by the owner like stop transactions, manipulate the fees, burn tokens and massively blacklist addresses. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract or renouncing ownership will eliminate all the contract threats. There is also a limit of max 50% fee.

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