Cyberscope

Audit Report

Tomiwagmi

August 2023

SHA256 2481c3300772defa4f81a3f4e6dba69e5284482d67b6e1be04913441c707ed97

Audited by © cyberscope

Analysis

● Critical ● Medium ● Minor / Informative ● Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Unresolved
●	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
●	RSW	Redundant Storage Writes	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	DPI	Decimals Precision Inconsistency	Unresolved
●	GO	Gas Optimization	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L05	Unused State Variable	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L13	Divide before Multiply Operation	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved

●	L15	Local Scope Variable Shadowing	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Testing Deploy	https://testnet.bscscan.com/address/0x7422b15b554f997e4ab86cac6d938a45717c8271

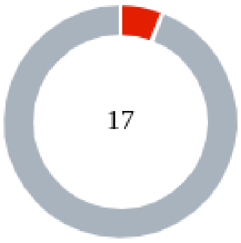
Audit Updates

Initial Audit	14 Aug 2023

Source Files

Filename	SHA256
contracts/Tomwagmi.sol	2481c3300772defa4f81a3f4e6dba69e5284482d67b6e1be04913441c707ed97

Findings Breakdown



●	Critical	1
●	Medium	0
●	Minor / Informative	16

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	1	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	16	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	contracts/Tomiwagmi.sol#L1534
Status	Unresolved

Description

The transactions are initially disabled for all users excluding the authorized addresses. The owner can enable the transactions for all users. Once the transactions are enable the owner will not be able to disable them again.

```
        if(!tradingActive){
            require(_isExcludedFromFees[from] ||
_isExcludedFromFees[to], "Trading is not active yet.");
        }
```

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 multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.sol#L1333,1338,1355,1359,1364
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 updates the state of `excludeFromDividends` addresses even if their current state is the same as the the one passed as an argument. As a result, the contract performs redundant storage writes.

Additionally, the functions `updateSwapEnabled` , `updateMaxAmount` , and `updateMaxWalletAmount` update the state of `swapEnabled` , `maxTransactionAmount` , and `maxWallet` respectively, even if their current state is the same as the value passed as an argument.

```
   function excludeFromDividends(address account) external
onlyOwner {
        dividendTracker.excludeFromDividends(account);
    }

    function includeInDividends(address account) external
onlyOwner {
        dividendTracker.includeInDividends(account);
    }

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

    function updateMaxAmount(uint256 newNum) external onlyOwner
{
        require(newNum > (totalSupply() * 1 / 1000)/1e18,
"Cannot set maxTransactionAmount lower than 0.1%");
        maxTransactionAmount = newNum * (10**18);
    }

    function updateMaxWalletAmount(uint256 newNum) external
onlyOwner {
        require(newNum > (totalSupply() * 1 / 100)/1e18,
"Cannot set maxWallet lower than 1%");
        maxWallet = newNum * (10**18);
    }
```

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.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.sol#L1516
Status	Unresolved

Description

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

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

```
    function updateSwapTokensAtAmount(uint256 newAmount)
external onlyOwner returns (bool){
        swapTokensAtAmount = newAmount;
        return true;
    }
```

Recommendation

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

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.sol#L1686
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 totalTokensToSwap = tokensForLiquidity +
tokensForMarketing + tokensForRewards;
        uint256 liquidityTokens = contractBalance *
tokensForLiquidity / totalTokensToSwap / 2;
        uint256 amountToSwapForETH =
contractBalance.sub(liquidityTokens);
        uint256 ethForMarketing =
ethBalance.mul(tokensForMarketing).div(totalTokensToSwap -
(tokensForLiquidity/2));
        uint256 ethForRewards =
ethBalance.mul(tokensForRewards).div(totalTokensToSwap -
(tokensForLiquidity/2));
        uint256 ethForLiquidity = ethBalance - ethForMarketing
- ethForRewards;
```

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.

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.sol#L1556
Status	Unresolved

Description

The decimals field of a contract's ERC20 token can be used to specify the number of decimal places that the token uses. For example, if decimals are set to `8`, it means that the smallest unit of the token is `0.00000001`, and if decimals are set to `18`, it means that the smallest unit of the token is `0.000000000000000001`.

However, there is an inconsistency in the way that the decimals field is handled in some ERC20 contracts. The ERC20 specification does not specify how the decimals field should be implemented, and as a result, some contracts use different precision numbers.

This inconsistency can cause problems when interacting with these contracts, as it is not always clear how the decimals field should be interpreted. For example, if a contract expects the decimals field to be 18 digits, but the contract being interacted with uses 8 digits, the result of the interaction may not be what was expected.

```
            if (automatedMarketMakerPairs[to] && totalSellFees
> 0){
                fees =
amount.mul(totalSellFees).div(feeDivisor);
                tokensForRewards += fees * rewardsSellFee /
totalSellFees;
                tokensForLiquidity += fees * liquiditySellFee /
totalSellFees;
                tokensForMarketing += fees * marketingSellFee /
totalSellFees;
                 tokensForBurn += fees * burnSellFee /
totalSellFees;
            }

            else if(automatedMarketMakerPairs[from] &&
totalBuyFees > 0) {
                fees =
amount.mul(totalBuyFees).div(feeDivisor);
                tokensForRewards += fees * rewardsBuyFee /
totalBuyFees;
                tokensForLiquidity += fees * liquidityBuyFee /
totalBuyFees;
                tokensForMarketing += fees * marketingBuyFee /
totalBuyFees;
                  tokensForBurn += fees * burnBuyFee /
totalBuyFees;
            }
```

Recommendation

To avoid these issues, it is important to carefully review the implementation of the decimals field of the underlying tokens. The team is advised to normalize each decimal to one single source of truth. A recommended way is to scale all the decimals to the greatest token's decimal. Hence, the contract will not lose precision in the calculations.

The following example depicts 3 tokens with different decimals precision.

ERC20	Decimals
Token 1	6
Token 2	9
Token 3	18

All the decimals could be normalized to 18 since it represents the ERC20 token with the greatest digits.

GO - Gas Optimization

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.sol#L1516,1570
Status	Unresolved

Description

Gas optimization refers to the process of reducing the amount of gas required to execute a transaction. Gas is the unit of measurement used to calculate the fees paid to miners for including a transaction in a block on the blockchain.

The contract variable `swapTokensAtAmount` is used to set a threshold where the contract will trigger the swap functionality. Since there is no input validation at the `updateSwapTokensAtAmount` function, the `swapTokensAtAmount` could be set to any amount such as zero. As a result, the contract will execute the swap functionality every time a transaction is taking place and consume more gas.

```
    function updateSwapTokensAtAmount(uint256 newAmount)
external onlyOwner returns (bool){
        swapTokensAtAmount = newAmount;
        return true;
    }

        uint256 contractTokenBalance =
balanceOf(address(this));

        bool canSwap = contractTokenBalance >=
swapTokensAtAmount;

        if(
            canSwap &&
            swapEnabled &&
            !swapping &&
            !automatedMarketMakerPairs[from] &&
            !_isExcludedFromFees[from] &&
            !_isExcludedFromFees[to]
        ) {
            swapping = true;
            swapBack();
            swapping = false;
        }
```

Recommendation

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

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.sol#L1333,1338,1359,1364,1369
Status	Unresolved

Description

The contract performs actions and state mutations from external methods that do not result in the emission of events. Emitting events for significant actions is important as it allows external parties, such as wallets or dApps, to track and monitor the activity on the contract. Without these events, it may be difficult for external parties to accurately determine the current state of the contract.

```
   function excludeFromDividends(address account) external
onlyOwner {
        dividendTracker.excludeFromDividends(account);
    }

    function includeInDividends(address account) external
onlyOwner {
        dividendTracker.includeInDividends(account);
    }

    function updateMaxAmount(uint256 newNum) external onlyOwner
{
        require(newNum > (totalSupply() * 1 / 1000)/1e18,
"Cannot set maxTransactionAmount lower than 0.1%");
        maxTransactionAmount = newNum * (10**18);
    }

    function updateMaxWalletAmount(uint256 newNum) external
onlyOwner {
        require(newNum > (totalSupply() * 1 / 100)/1e18,
"Cannot set maxWallet lower than 1%");
        maxWallet = newNum * (10**18);
    }
```

Recommendation

It is recommended to include events in the code that are triggered each time a significant action is taking place within the contract. These events should include relevant details such as the user's address and the nature of the action taken. By doing so, the contract will be more transparent and easily auditable by external parties. It will also help prevent potential issues or disputes that may arise in the future.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.sol#L1285
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
dividendTracker
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.sol#L564,705,801,823,830,837,847,1006,1051,1078,1224,1369,1378
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 WETH() external pure returns (address);
uint256 constant internal magnitude = 2**128
address _rewardToken
address _owner
address _account
uint256 _newMinimumBalance
mapping (address => bool) public
_isExcludedMaxTransactionAmount
uint256 _liquidityFee
uint256 _burnFee
uint256 _marketingFee
uint256 _rewardsFee
```

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	contracts/Tomiwagmi.sol#L495
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	contracts/Tomiwagmi.sol#L1361,1366,1370,1379,1517
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.

```
maxTransactionAmount = newNum * (10**18)
maxWallet = newNum * (10**18)
marketingBuyFee = _marketingFee
marketingSellFee = _marketingFee
swapTokensAtAmount = newAmount
```

Recommendation

By including all required events in the contract and thoroughly testing the contract's functionality, the contract ensures that it performs as intended and does not have any missing events that could cause issues with its arithmetic.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.sol#L541,900
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 abs(int256 a) internal pure returns (int256) {
        require(a != MIN_INT256);
        return a < 0 ? -a : a;
    }

function get(address key) private view returns (uint) {
        return tokenHoldersMap.values[key];
    }
```

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	contracts/Tomiwagmi.sol#L1602,1603,1604,1605,1606,1612,1613,1614,1615
Status	Unresolved

Description

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

```
fees = amount.mul(totalSellFees).div(feeDivisor)
tokensForRewards += fees * rewardsSellFee / totalSellFees
```

Recommendation

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

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.sol#L857,868,1148,1149,1639
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
uint256 i
bool paid
uint256 lastProcessedIndex
uint256 iterations
uint256 claims
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.sol#L823,830,837,847,1266
Status	Unresolved

Description

Local scope variable shadowing occurs when a local variable with the same name as a variable in an outer scope is declared within a function or code block. When this happens, the local variable "shadows" the outer variable, meaning that it takes precedence over the outer variable within the scope in which it is declared.

```
address _owner
uint256 totalSupply = 10000000000 * 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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	contracts/Tomiwagmi.sol#L812
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.

```
IERC20(_rewardToken).transfer(user, _withdrawableDividend)
```

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		
IUniswapV2Factory	Interface			
	createPair	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	✓	
	_createInitialSupply	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External		-

	withdrawnDividendOf	External		-
	accumulativeDividendOf	External		-
DividendPaying TokenInterface	Interface			
	dividendOf	External		-
	distributeDividends	External	Payable	-
	withdrawDividend	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	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	✓	-
DividendPayingToken	Implementation	DividendPayingTokenInterface, DividendPayingTokenOptionalInterface, Ownable		
		Public	✓	-
		External	Payable	-

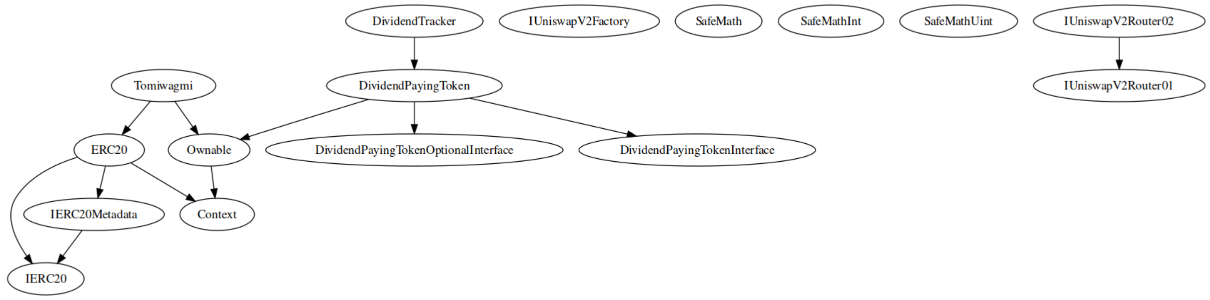
	distributeDividends	Public	Payable	-
	buyTokens	Internal	✓	
	withdrawDividend	External	✓	-
	_withdrawDividendOfUser	Internal	✓	
	dividendOf	External		-
	withdrawableDividendOf	Public		-
	withdrawnDividendOf	External		-
	accumulativeDividendOf	Public		-
	_increase	Internal	✓	
	_reduce	Internal	✓	
	_setBalance	Internal	✓	
DividendTracker	Implementation	DividendPayingToken		
	get	Private		
	getIndexOfKey	Private		
	getKeyAtIndex	Private		
	size	Private		
	set	Private	✓	
	remove	Private	✓	
		Public	✓	-
	excludeFromDividends	External	✓	onlyOwner
	includeInDividends	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner

	getLastProcessedIndex	External		-
	getNumberOfTokenHolders	External		-
	getAccount	Public		-
	getAccountAtIndex	External		-
	canAutoClaim	Private		
	updateMinimumTokenBalanceForDividends	External	✓	onlyOwner
	setBalance	External	✓	onlyOwner
	process	External	✓	-
	processAccount	Public	✓	onlyOwner
Tomiwagmi	Implementation	ERC20, Ownable		
		Public	✓	ERC20
		External	Payable	-
	disableTransferDelay	External	✓	onlyOwner
	excludeFromDividends	External	✓	onlyOwner
	includeInDividends	External	✓	onlyOwner
	enableTrading	External	✓	onlyOwner
	burn	Public	✓	-
	updateSwapEnabled	External	✓	onlyOwner
	updateMaxAmount	External	✓	onlyOwner
	updateMaxWalletAmount	External	✓	onlyOwner
	updateBuyFees	External	✓	onlyOwner
	updateSellFees	External	✓	onlyOwner

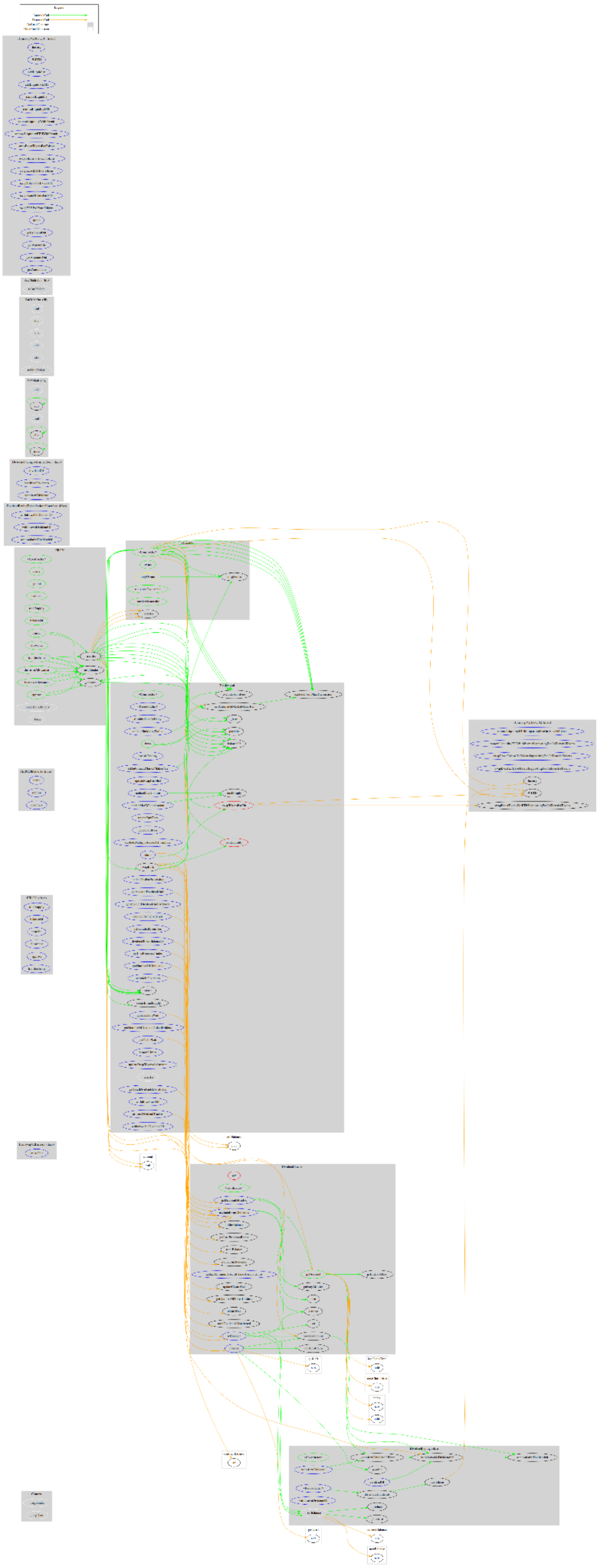
	excludeFromMaxTransaction	Public	✓	onlyOwner
	excludeFromFees	Public	✓	onlyOwner
	excludeMultipleAccountsFromFees	External	✓	onlyOwner
	setAutomatedMarketMakerPair	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Private	✓	
	updateMarketingWallet	External	✓	onlyOwner
	updateGasForProcessing	External	✓	onlyOwner
	updateClaimWait	External	✓	onlyOwner
	getClaimWait	External		-
	getTotalDividendsDistributed	External		-
	isExcludedFromFees	External		-
	withdrawableDividendOf	External		-
	dividendTokenBalanceOf	External		-
	getAccountDividendsInfo	External		-
	getAccountDividendsInfoAtIndex	External		-
	processDividendTracker	External	✓	-
	claim	External	✓	-
	getLastProcessedIndex	External		-
	getNumberOfDividendTokenHolders	External		-
	getNumberOfDividends	External		-
	removeLimits	External	✓	onlyOwner
	updateSwapTokensAtAmount	External	✓	onlyOwner
	_transfer	Internal	✓	

	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	swapBack	Private	✓	
	withdrawStuckEth	External	✓	onlyOwner

Inheritance Graph



Flow Graph



Summary

Tomiwagmi 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 and manipulate the fees. A multi-wallet signing pattern will provide security against potential hacks. Temporarily locking the contract will eliminate all the contract threats. There is also a limit of max 10% 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