Audit Report

Mx Million Metaverse DAO

December 2023

Network	MATIC
Address	0xa834ed56a63d1801c365f0ceacc4060283ebb5f5
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	Unresolved
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	UFV	Unchecked Fee Value	Unresolved
●	MC	Missing Check	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	MAU	Misleading Address Usage	Unresolved
●	FRV	Fee Restoration Vulnerability	Unresolved
●	FSA	Fixed Swap Address	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	RED	Redudant Event Declaration	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

●	L19	Stable Compiler Version	Unresolved

Table of Contents

Analysis	**1**
Diagnostics	**2**
Table of Contents	**4**
Review	**6**
Audit Updates	6
Source Files	6
Findings Breakdown	**7**
ST - Stops Transactions	8
Description	8
Recommendation	8
ELFM - Exceeds Fees Limit	10
Description	10
Recommendation	10
UFV - Unchecked Fee Value	12
Description	12
Recommendation	12
MC - Missing Check	14
Description	14
Recommendation	14
PVC - Price Volatility Concern	15
Description	15
Recommendation	16
MAU - Misleading Address Usage	17
Description	17
Recommendation	17
FRV - Fee Restoration Vulnerability	19
Description	19
Recommendation	20
FSA - Fixed Swap Address	21
Description	21
Recommendation	21
RSW - Redundant Storage Writes	22
Description	22
Recommendation	22
RED - Redudant Event Declaration	23
Description	23
Recommendation	23
RSML - Redundant SafeMath Library	24
Description	24

Recommendation 24
L02 - State Variables could be Declared Constant 25
Description 25
Recommendation 25
L04 - Conformance to Solidity Naming Conventions 26
Description 26
Recommendation 26
L07 - Missing Events Arithmetic 28
Description 28
Recommendation 28
L09 - Dead Code Elimination 29
Description 29
Recommendation 31
L17 - Usage of Solidity Assembly 32
Description 32
Recommendation 32
L19 - Stable Compiler Version 33
Description 33
Recommendation 33
Functions Analysis 34
Inheritance Graph 42
Flow Graph 43
Summary 44
Disclaimer 45
About Cyberscope 46

Review

Contract Name	MXMDAO
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	800 runs
Explorer	https://polygonscan.com/address/0xa834ed56a63d1801c365f0ceacc4060283ebb5f5
Address	0xa834ed56a63d1801c365f0ceacc4060283ebb5f5
Network	MATIC
Symbol	MXMDAO
Decimals	18
Total Supply	10,000,000

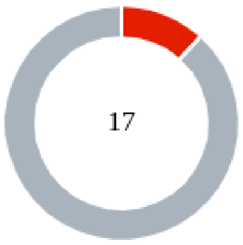
Audit Updates

Initial Audit	09 Dec 2023

Source Files

Filename	SHA256
Token.sol	f00df8b96bab9c6b0e100ff4045747ee63fb00a86833b14329ce73178825c00c

Findings Breakdown



●	Critical	2
●	Medium	0
●	Minor / Informative	15

Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	2	0	0	0
●	Medium	0	0	0	0
●	Minor / Informative	15	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	Token.sol#L540,649
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `_maxTxAmount` to zero. As a result, the contract may operate as a honeypot.

```
    function setMaxTxPercent(uint256 maxTxPercent) external
onlyOwner() {
        _maxTxAmount = _tTotal.mul(maxTxPercent).div(
            10**3
        );
    }

if(from != owner() && to != owner())
  require(amount <= _maxTxAmount, "Transfer amount exceeds the
maxTxAmount.");
```

Recommendation

The contract could embody a check for not allowing setting the `_maxTxAmount` 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 should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions.

Temporary Solutions:

These measurements do not decrease the severity of the finding

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

- Introduce a governance model where users will vote about the actions.

Permanent Solution:

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

ELFM - Exceeds Fees Limit

Criticality	Critical
Location	Token.sol#L531,534,537
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 `setTaxFeePercent` function with a high percentage value.

```
   function setTaxFeePercent(uint256 taxFee) external
onlyOwner() {
        _taxFee = taxFee;
    }
    function setBurnFeePercent(uint256 BurnFee) external
onlyOwner() {
        _BurnFee = BurnFee;
    }
    function setLiquidityFeePercent(uint256 liquidityFee)
external onlyOwner() {
        _liquidityFee = liquidityFee;
    }
```

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.

Temporary Solutions:

These measurements do not decrease the severity of the finding

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

- Introduce a governance model where users will vote about the actions.

Permanent Solution:

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

UFV - Unchecked Fee Value

Criticality	Minor / Informative
Location	Token.sol#L618,705
Status	Unresolved

Description

The contract is designed to use the `takeFee` variable to determine whether a fee should be deducted during token transfers. In scenarios where `takeFee` is set to `false`, the contract invokes the `removeAllFee` function. This function checks if both `_taxFee` and `_liquidityFee` are zero but does not consider the value of `_BurnFee`. Consequently, if `_BurnFee` is not zero, users will still incur a fee. This oversight in the fee logic can lead to users paying a burn fee even when no fees are intended to be applied, contradicting the expected functionality of the contract when `takeFee` is `false`.

```
   function removeAllFee() private {
        if(_taxFee == 0 && _liquidityFee == 0) return;
        _previousTaxFee = _taxFee;
        _previousBurnFee = _BurnFee;
        _previousLiquidityFee = _liquidityFee;
        _taxFee = 0;
        _BurnFee = 0;
        _liquidityFee = 0;
    }
    function _tokenTransfer(address sender, address recipient,
uint256 amount,bool takeFee) private {
        if(!takeFee)
            removeAllFee();
        ...
        }
```

Recommendation

It is recommended to modify the `removeAllFee` function to include a check for the `_BurnFee` variable. Specifically, the function should verify if `_BurnFee` is also zero before returning from the function. This change will ensure that all relevant fee components are considered when deciding to remove fees, aligning the contract's behavior with its intended functionality. By doing so, the contract will accurately reflect the fee structure as intended, preventing users from being charged fees unexpectedly.

MC - Missing Check

Criticality	Minor / Informative
Location	Token.sol#L596
Status	Unresolved

Description

The contract is processing variables that have not been properly sanitized and checked that they form the proper shape. These variables may produce vulnerability issues.

Specifically, the contract is currently structured to execute the `_takeBurn` function without initially checking if the `tBurn` variable is greater than zero. This oversight leads to the function performing redundant calculations when the `tBurn` value is zero. Such unnecessary computations can result in inefficiencies in contract execution, potentially consuming more gas than needed.

```
    function _takeBurn(uint256 tBurn) private {
        uint256 currentRate =  _getRate();
        uint256 rBurn = tBurn.mul(currentRate);
        _rOwned[_burnWalletAddress] = 
_rOwned[_burnWalletAddress].add(rBurn);
        if(_isExcluded[_burnWalletAddress])
            _tOwned[_burnWalletAddress] = 
_tOwned[_burnWalletAddress].add(tBurn);
    }
```

Recommendation

The team is advised to properly check the variables according to the required specifications. It is recommended to add a conditional check at the beginning of the `_takeBurn` function to determine if the `tBurn` variable is greater than zero. This check should be implemented before proceeding with any calculations or state changes within the function. By incorporating this check, the contract can avoid needless calculations and state modifications when `tBurn` is zero, optimizing the contract's performance and potentially reducing gas costs for transactions involving this function.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	Token.sol#L384,402,651
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `numTokensSellToAddToLiquidity` sets a threshold where the contract will trigger the swap functionality. This variable is set to the value of `10000000 * 10**18` which is equal to the total supply.

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.

```
   uint256 private _tTotal = 10000000 * 10**18;

    uint256 private numTokensSellToAddToLiquidity = 10000000 *
10**18;

        uint256 contractTokenBalance =
balanceOf(address(this));
        if(contractTokenBalance >= _maxTxAmount)
        {
            contractTokenBalance = _maxTxAmount;
        }
        bool overMinTokenBalance = contractTokenBalance >=
numTokensSellToAddToLiquidity;
        if (
            overMinTokenBalance &&
            !inSwapAndLiquify &&
            from != uniswapV2Pair &&
            swapAndLiquifyEnabled
        ) {
            contractTokenBalance =
numTokensSellToAddToLiquidity;
            swapAndLiquify(contractTokenBalance);
        }
```

Recommendation

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

MAU - Misleading Address Usage

Criticality	Minor / Informative
Location	Token.sol#L382,596
Status	Unresolved

Description

The contract contains the `_burnWalletAddress` variable, which misleadingly suggests it represents a conventional burn address,typically the zero address in blockchain systems. However, this variable is initialized with the actual wallet address `0x43db6dC456A25Ff3B43E07C05beb5cbE0E144E9B` and is used within the `_takeBurn` function. This practice deviates from the standard use of a burn address, where tokens are permanently removed from circulation. Instead, tokens are transferred to this specified address, potentially leading to unintended behaviors and misinterpretation of the contract's token burn mechanism.

```
   address private _burnWalletAddress =
0x43db6dC456A25Ff3B43E07C05beb5cbE0E144E9B;

    function _takeBurn(uint256 tBurn) private {
        uint256 currentRate =  _getRate();
        uint256 rBurn = tBurn.mul(currentRate);
        _rOwned[_burnWalletAddress] =
_rOwned[_burnWalletAddress].add(rBurn);
        if(_isExcluded[_burnWalletAddress])
            _tOwned[_burnWalletAddress] =
_tOwned[_burnWalletAddress].add(tBurn);
    }
```

Recommendation

It is recommended to revise the contract to align with standard practices for token burning. This can be achieved by renaming the `_burnWalletAddress` variable to more accurately reflect its actual usage. If the intention is to remove tokens from circulation permanently, the variable should be initialized with the zero address. Alternatively, if the current functionality is intended, a more descriptive name should be used to avoid confusion and ensure clarity in the contract's operations. Additionally, reviewing and amending the `_takeBurn` function to align with these changes is crucial for maintaining consistency and transparency in the contract's token handling processes.

FRV - Fee Restoration Vulnerability

Criticality	Minor / Informative
Location	Token.sol#L618,627,705
Status	Unresolved

Description

The contract demonstrates a potential vulnerability upon removing and restoring the fees. This vulnerability can occur when the fees have been set to zero. During a transaction, if the fees have been set to zero, then both remove fees and restore fees functions will be executed. The remove fees function is executed to temporarily remove the fees, ensuring the sender is not taxed during the transfer. However, the function prematurely returns without setting the variables that hold the previous fee values.

As a result, when the subsequent restore fees function is called after the transfer, it restores the fees to their previous values. However, since the previous fee values were not properly set to zero, there is a risk that the fees will retain their non-zero values from before the fees were removed. This can lead to unintended consequences, potentially causing incorrect fee calculations or unexpected behavior within the contract.

```
   function removeAllFee() private {
        if(_taxFee == 0 && _liquidityFee == 0) return;
        _previousTaxFee = _taxFee;
        _previousBurnFee = _BurnFee;
        _previousLiquidityFee = _liquidityFee;
        _taxFee = 0;
        _BurnFee = 0;
        _liquidityFee = 0;
    }
    function restoreAllFee() private {
        _taxFee = _previousTaxFee;
        _BurnFee = _previousBurnFee;
        _liquidityFee = _previousLiquidityFee;
    }
    function _tokenTransfer(address sender, address recipient,
uint256 amount,bool takeFee) private {
        if(!takeFee)
            removeAllFee();
        ...
        } else {
            _transferStandard(sender, recipient, amount);
        }
        if(!takeFee)
            restoreAllFee();
    }
```

Recommendation

The team is advised to modify the remove fees function to ensure that the previous fee values are correctly set to zero, regardless of their initial values. A recommended approach would be to remove the early return when both fees are zero.

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	Token.sol#L417
Status	Unresolved

Description

The swap address is assigned once and it can not be changed. It is a common practice in decentralized exchanges to create new swap versions. A contract that cannot change the swap address may not be able to catch up to the upgrade. As a result, the contract will not be able to migrate to a new liquidity pool pair or decentralized exchange.

```
   constructor () {
        ...
        IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(0xa5E0829CaCEd8fFDD4De3c43696c57F7D7A678ff);
        uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(address(this),
_uniswapV2Router.WETH());
        uniswapV2Router = _uniswapV2Router;
        ....
        }
```

Recommendation

The team is advised to add the ability to change the pair and router address in order to cover potential liquidity pool migrations. It would be better to support multiple pair addresses so the token will be able to have the same behavior in all the decentralized liquidity pairs.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Token.sol#L526,529
Status	Unresolved

Description

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

```
   function excludeFromFee(address account) public onlyOwner {
        _isExcludedFromFee[account] = true;
    }
    function includeInFee(address account) public onlyOwner {
        _isExcludedFromFee[account] = false;
    }
```

Recommendation

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

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	Token.sol#L403
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 event `MinTokensBeforeSwapUpdated` is declared and not being used in the contract. As a result, it is redundant.

```
event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
```

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.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Token.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	Token.sol#L382,384,387,388,389,402
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 _burnWalletAddress =
0x43db6dC456A25Ff3B43E07C05beb5cbE0E144E9B;
uint256 private _tTotal = 10000000 * 10**18;
string private _name = "Mx Million Metaverse DAO";
string private _symbol = "MXMDAO";
uint8 private _decimals = 18;
uint256 private numTokensSellToAddToLiquidity = 10000000 *
10**18;
```

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	Token.sol#L210,211,225,243,390,392,394,401,534,545,603,608,613
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
function DOMAIN_SEPARATOR() external view returns (bytes32);
function PERMIT_TYPEHASH() external pure returns (bytes32);
function MINIMUM_LIQUIDITY() external pure returns (uint);
function WETH() external pure returns (address);
uint256 public _taxFee = 20;
uint256 public _BurnFee = 20;
uint256 public _liquidityFee = 20;
uint256 public _maxTxAmount = 10000000 * 10**18;
...
```

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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Token.sol#L532,535,538,541
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.

```
_taxFee = taxFee;
 _BurnFee = BurnFee;
_liquidityFee = liquidityFee;
_maxTxAmount = _tTotal.mul(maxTxPercent).div(10**3);
```

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	Token.sol#L104,109,114,117,120,123,129,132,137,140,145
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) {
        uint256 size;
        assembly { size := extcodesize(account) }
        return size > 0;
    }
    function sendValue(address payable recipient, uint256
amount) internal {
        ...
    }
    function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
      ...
    }
    function functionCall(address target, bytes memory data,
string memory errorMessage) internal returns (bytes memory) {
        ...
    }
    function functionCallWithValue(address target, bytes memory
data, uint256 value) internal returns (bytes memory) {
        ...
    }
    function functionCallWithValue(address target, bytes memory
data, uint256 value, string memory errorMessage) internal
returns (bytes memory) {
        ...
    }
    function functionStaticCall(address target, bytes memory
data) internal view returns (bytes memory) {
       ...
    }
    function functionStaticCall(address target, bytes memory
data, string memory errorMessage) internal view returns (bytes
memory) {
        ..
    }
    function functionDelegateCall(address target, bytes memory
data) internal returns (bytes memory) {
        ..
    }
    function functionDelegateCall(address target, bytes memory
data, string memory errorMessage) internal returns (bytes
memory) {
        ...
    }
    function _verifyCallResult(bool success, bytes memory
returndata, string memory errorMessage) private pure
returns(bytes memory) {
    ...
    }
```

```
...
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	Token.sol#L106,150
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.

```
   function isContract(address account) internal view returns
(bool) {
        uint256 size;
        assembly { size := extcodesize(account) }
        return size > 0;
    }
    ...
            if (returndata.length > 0) {
                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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Token.sol#L11
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.8.4;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		

	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionStaticCall	Internal		
	functionStaticCall	Internal		
	functionDelegateCall	Internal	✓	
	functionDelegateCall	Internal	✓	
	_verifyCallResult	Private		
Ownable	Implementation	Context		

		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			
	feeTo	External		-
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
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	✓	-
IUniswapV2Router01	Interface			
	factory	External		-

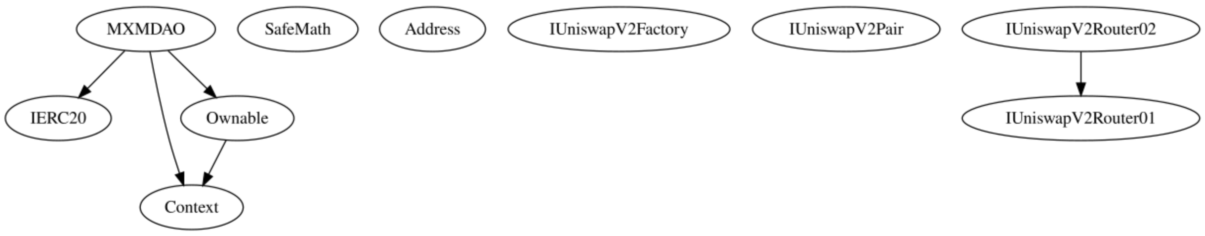
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	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	✓	-
MXMDAO	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	✓	-
	isExcludedFromReward	Public		-
	totalFees	Public		-
	deliver	Public	✓	-
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-

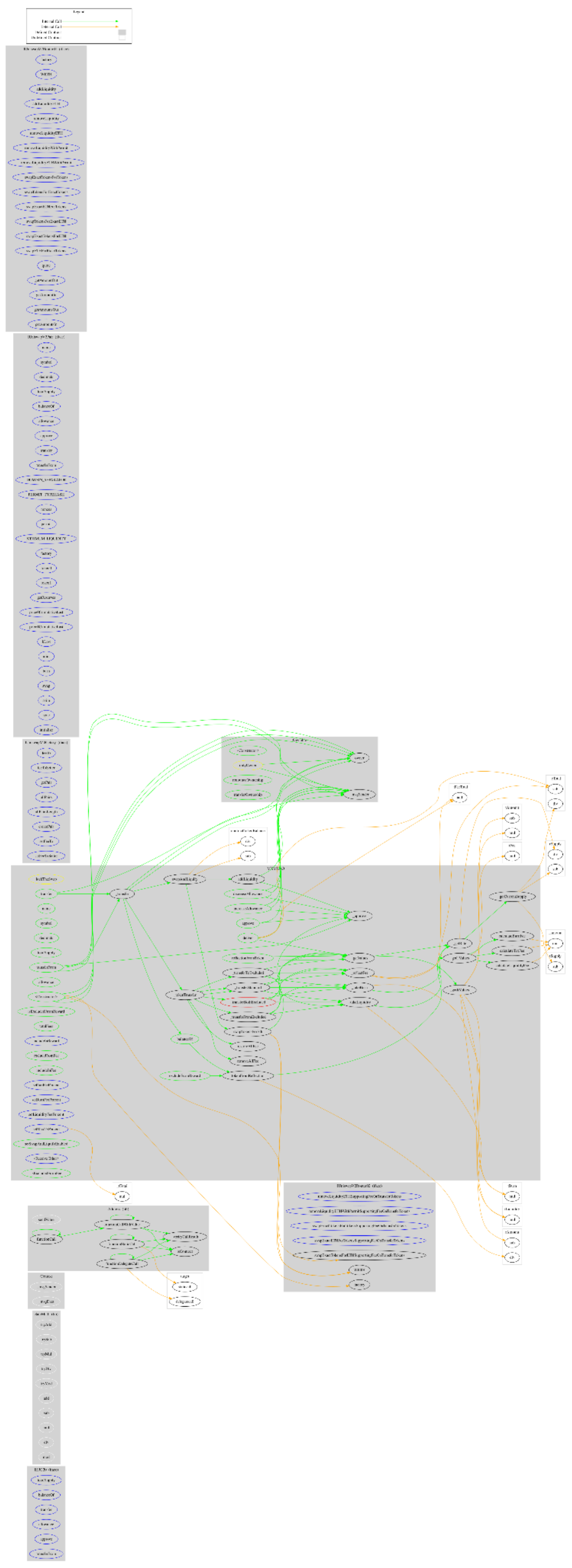
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	_transferBothExcluded	Private	✓	
	excludeFromFee	Public	✓	onlyOwner
	includeInFee	Public	✓	onlyOwner
	setTaxFeePercent	External	✓	onlyOwner
	setBurnFeePercent	External	✓	onlyOwner
	setLiquidityFeePercent	External	✓	onlyOwner
	setMaxTxPercent	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
		External	Payable	-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeLiquidity	Private	✓	
	_takeBurn	Private	✓	
	calculateTaxFee	Private		
	calculateBurnFee	Private		
	calculateLiquidityFee	Private		
	removeAllFee	Private	✓	

	restoreAllFee	Private	✓	
	isExcludedFromFee	Public		-
	_approve	Private	✓	
	_transfer	Private	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	

Inheritance Graph



Flow Graph



Summary

Mx Million Metaverse DAO 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 or renouncing ownership will eliminate all the contract threats.

Disclaimer

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.

The Cyberscope team

https://www.cyberscope.io