Cyberscope

Audit Report

MollarsToken

January 2024

Network	ETH
Address	0xb2cb194701094239db774049d9d72f7838944a17
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
Medium	UPA	Unexcluded Pinksale Address	Unresolved
Minor	CR	Code Repetition	Unresolved
Minor	DDP	Decimal Division Precision	Unresolved
Minor	EPC	Existing Pair Creation	Unresolved
Minor	IDI	Immutable Declaration Improvement	Unresolved
Minor	MEM	Misleading Error Messages	Unresolved
Minor	PTRP	Potential Transfer Revert Propagation	Unresolved
Minor	PVC	Price Volatility Concern	Unresolved
Minor	RED	Redudant Event Declaration	Unresolved
Minor	RSML	Redundant SafeMath Library	Unresolved
Minor	L02	State Variables could be Declared Constant	Unresolved
Minor	L04	Conformance to Solidity Naming Conventions	Unresolved
Minor	L07	Missing Events Arithmetic	Unresolved
Minor	L16	Validate Variable Setters	Unresolved

Table of Contents

Review

Contract Name	MOLLARS
Compiler Version	v0.8.17+commit.8df45f5f
Optimization	200 runs
Explorer	https://etherscan.io/address/0xb2cb194701094239db774049d9d72f7838944a17
Address	0xb2cb194701094239db774049d9d72f7838944a17
Network	ETH
Symbol	MOLLARS
Decimals	9
Total Supply	10,000,000
Badge Eligibility	Yes

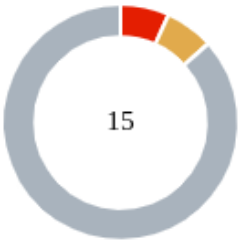
Audit Updates

Initial Audit	15 Jan 2024

Source Files

Filename	SHA256
MOLLARS.sol	29b55e1f6a3527f1953f23cfc2c2296a3e0711ddb681175e705f79960e9d66a2

Findings Breakdown



Critical	1
Medium	1
Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	1	0	0	0
Minor / Informative	13	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	MOLLARS.sol#L382
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.

Additionally, the contract owner has the authority to stop the sales for all users excluding the owner, as described in detail in section `PVC` and `PTRP`. As a result, the contract may operate as a honeypot.

```
if (!isFeeExempt[sender] && !isFeeExempt[recipient]) {
    // trading disable till launch
    if (!trading) {
        require(
            pair != sender && pair != recipient,
            "Trading is disable"
        );
    }
```

Recommendation

The team is strongly encouraged to adhere to the recommendations outlined in the respective sections. 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.

UPA - Unexcluded Pinksale Address

Criticality	Medium
Location	MOLLARS.sol#L412
Status	Unresolved

Description

The contract is designed with a fee mechanism that applies to each transaction. This design, poses a significant obstacle for integration with platforms like Pinksale. Specifically, for the creation of a launchpad on Pinksale, the Pinksale factory address must be exempted from these transaction fees. Without this exemption, the creation of the pool on Pinksale will be prevented.

```
        if (
            isFeeExempt[sender] ||
            isFeeExempt[recipient] ||
            (sender != pair && recipient != pair)
        ) {
            amountReceived = amount;
        } else {
            uint256 feeAmount;
            if (sender == pair) {
                feeAmount =
amount.mul(_totalFee).div(feeDenominator);
                amountReceived = amount.sub(feeAmount);
                takeFee(sender, feeAmount);
                setAccFee(amount);
            } else {
                feeAmount =
amount.mul(_totalFee).div(feeDenominator);
                amountReceived = amount.sub(feeAmount);
                takeFee(sender, feeAmount);
                setAccFee(amount);
            }
```

Recommendation

It is recommended to modify the contract to exclude the Pinksale factory address from the fee mechanism. This will ensure compatibility with Pinksale and facilitate the smooth creation of pools on the platform.

CR - Code Repetition

Criticality	Minor / Informative
Location	MOLLARS.sol#L415
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.

```
feeAmount = amount.mul(_totalFee).div(feeDenominator);
amountReceived = amount.sub(feeAmount);
takeFee(sender, feeAmount);
setAccFee(amount);
    } else {
feeAmount = amount.mul(_totalFee).div(feeDenominator);
amountReceived = amount.sub(feeAmount);
takeFee(sender, feeAmount);
setAccFee(amount);
}
```

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.

DDP - Decimal Division Precision

Criticality	Minor / Informative
Location	MOLLARS.sol#L462
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.

```
_burnFeeCount += _amount.mul(_burnFee).div(feeDenominator);
_marketingFeeCount +=
_amount.mul(_marketingFee).div(feeDenominator);
```

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.

EPC - Existing Pair Creation

Criticality	Minor / Informative
Location	MOLLARS.sol#L512
Status	Unresolved

Description

The contract contains a function that does not handle the scenario where a pair already exists prior to its execution. If a pair for the given tokens has already been established, the `createPair` function will revert and not proceed with the creation of a new pair. As a result, if a pair has been previously set up before the function is invoked, the contract will encounter an error when trying to call the `createPair` function. This will prevent the successful execution, essentially leading the function to revert.

```
    function addLp(address _router) external payable onlyOwner {
        router = IDexRouter(_router);
        pair = IDexFactory(router.factory()).createPair(
            address(this),
            router.WETH()
        );
        isFeeExempt[address(router)] = true;
        _allowances[address(this)][address(router)] =
_totalSupply;
        router.addLiquidityETH{value: msg.value}(
            address(this),
            balanceOf(address(this)),
            0,
            0,
            owner(),
            block.timestamp
        );
        IERC20Extended(pair).approve(address(router),
type(uint256).max);
    }
```

Recommendation

To mitigate the risks associated with attempting to create an already existing pair, it is recommended to implement a check to determine whether the pair already exists before

proceeding to create a new pair. This can be achieved by utilizing the getPair function of the Factory contract to retrieve the address of the pair contract for the specified tokens. If the address returned by the getPair function is the zero address, it indicates that the pair does not exist, and the contract can proceed with the createPair function. Conversely, if a non-zero address is returned, it indicates that the pair already exists, and the createPair function will revert.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	MOLLARS.sol#L297
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.

```
burnReceiver
```

Recommendation

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

MEM - Misleading Error Messages

Criticality	Minor / Informative
Location	MOLLARS.sol#L549
Status	Unresolved

Description

The contract is using misleading error messages. These error messages do not accurately reflect the problem, making it difficult to identify and fix the issue. As a result, the users will not be able to find the root cause of the error.

```
require(_amount > 0)
```

Recommendation

The team is suggested to provide a descriptive message to the errors. This message can be used to provide additional context about the error that occurred or to explain why the contract execution was halted. This can be useful for debugging and for providing more information to users that interact with the contract.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	MOLLARS.sol#L505
Status	Unresolved

Description

The contract sends funds to a `marketingFeeReceiver` as part of the transfer flow. This address can either be a wallet address or a contract. If the address belongs to a contract then it may revert from incoming payment. As a result, the error will propagate to the token's contract and revert the transfer.

```
payable(marketingFeeReceiver).transfer(amountEth);
```

Recommendation

The contract should tolerate the potential revert from the underlying contracts when the interaction is part of the main transfer flow. This could be achieved by not allowing set contract addresses or by sending the funds in a non-revertable way.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	MOLLARS.sol#L471,483,551
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapThreshold` 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 shouldSwapBack() internal view returns (bool) {
        return
            ...
            _balances[address(this)] >= swapThreshold;
    }
    ...
        uint256 amountToSwap =
swapThreshold.sub(amountForBurning);
        _allowances[address(this)][address(router)] =
_totalSupply;
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = router.WETH();
        uint256 balanceBefore = address(this).balance;


router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            amountToSwap,
            0,
            path,
            address(this),
            block.timestamp
        );
    ...
    function setSwapBackSettings(bool _enabled, uint256
_amount)
        external
        onlyOwner
    {
        require(_amount > 0);
        swapEnabled = _enabled;
        swapThreshold = _amount;
    }
```

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.

RED - Redudant Event Declaration

Criticality	Minor / Informative
Location	MOLLARS.sol#L294
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 `AutoLiquify` is declared and not being used in the contract. As a result, it is redundant.

```
event AutoLiquify(uint256 amountEth, uint256 amountBOG);
```

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	MOLLARS.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	MOLLARS.sol#L278
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 feeDenominator = 100_00
```

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	MOLLARS.sol#L141,262,263,264,265,271,272,274,275,277,461,512,541,545
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);
string private constant _name = "MollarsToken"
string private constant _symbol = "MOLLARS"
uint8 private constant _decimals = 9
uint256 private constant _totalSupply = 10_000_000 *
10**_decimals
uint256 _burnFee = 1_00
uint256 _marketingFee = 3_00
uint256 _burnFeeCount
uint256 _marketingFeeCount
uint256 public _totalFee = 4_00
uint256 _amount
address _router
address _marketingFeeReceiver
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.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	MOLLARS.sol#L436
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.

```
_burnFee = burnFee_
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	MOLLARS.sol#L298,542
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.

```
marketingFeeReceiver = marketingFeeReceiver_
marketingFeeReceiver = _marketingFeeReceiver
```

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

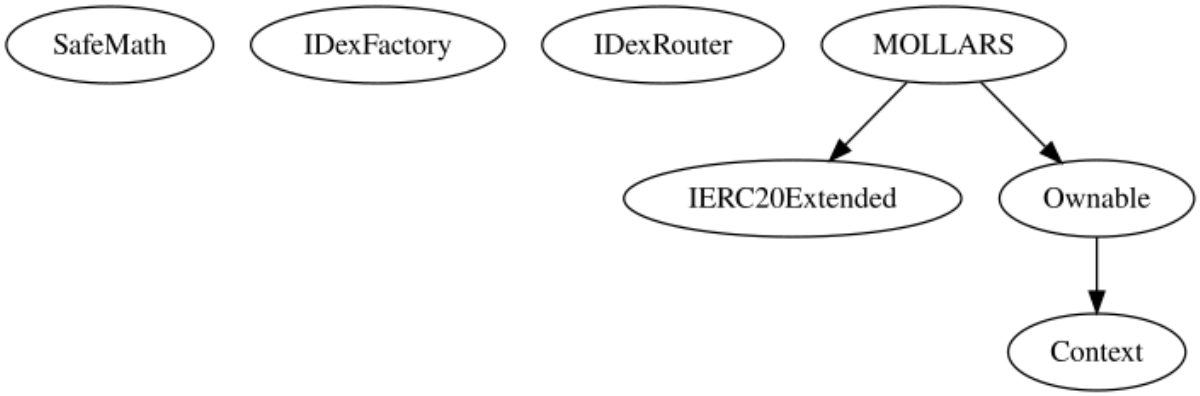
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
SafeMath	Library			
	tryAdd	Internal		
	trySub	Internal		
	tryMul	Internal		
	tryDiv	Internal		
	tryMod	Internal		
	add	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	mod	Internal		
	sub	Internal		
	div	Internal		
	mod	Internal		
IDexFactory	Interface			
	createPair	External	✓	-
IDexRouter	Interface			

	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
IERC20Extended	Interface			
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-

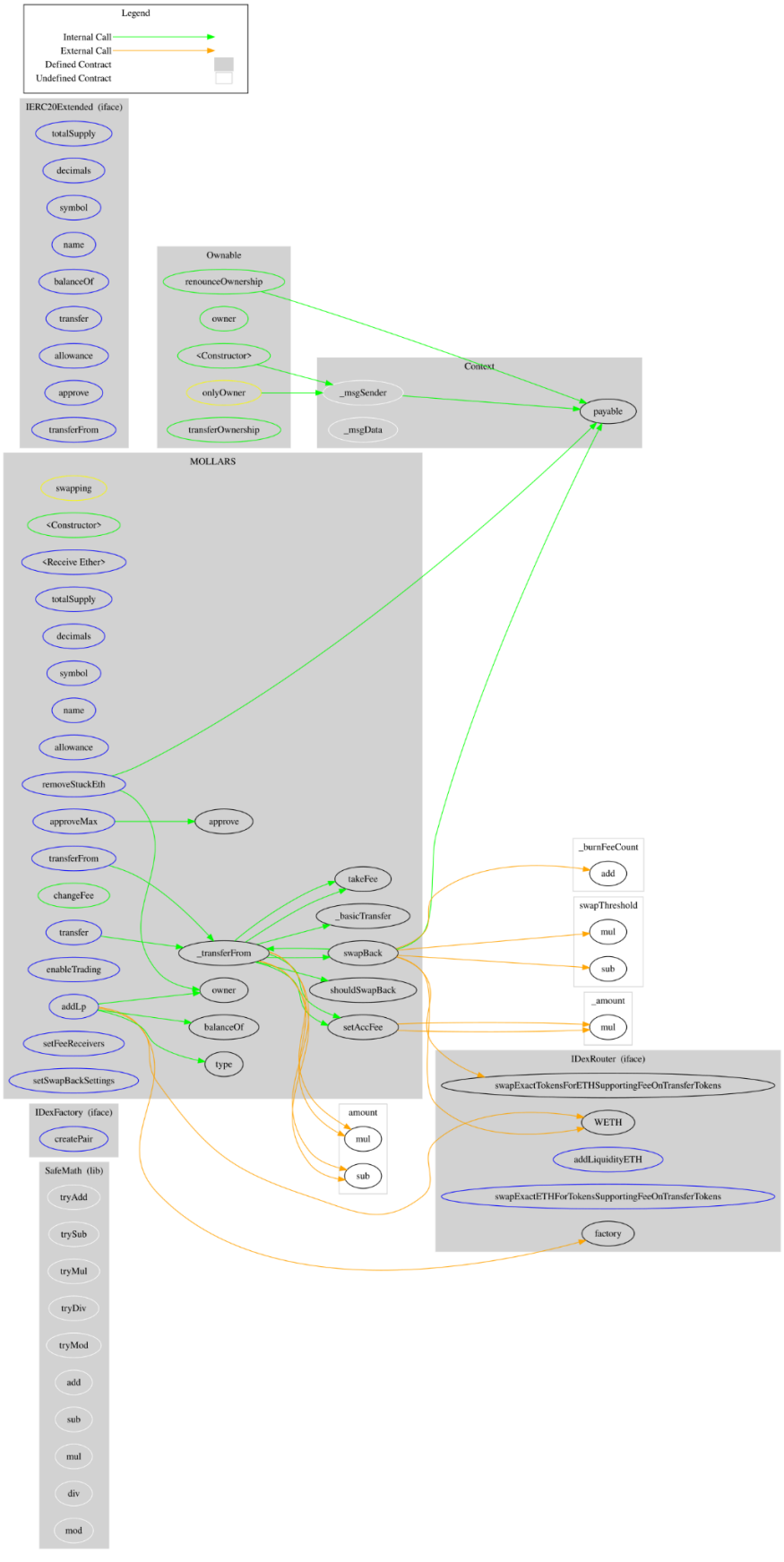
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
MOLLARS	Implementation	IERC20Extended, Ownable		
		Public	✓	Ownable
		External	Payable	-
	totalSupply	External		-
	decimals	External		-
	symbol	External		-
	name	External		-
	balanceOf	Public		-
	allowance	External		-
	approve	Public	✓	-
	approveMax	External	✓	-
	transfer	External	✓	-
	transferFrom	External	✓	-
	_transferFrom	Internal	✓	
	changeFee	Public	✓	onlyOwner
	_basicTransfer	Internal	✓	
	takeFee	Internal	✓	
	setAccFee	Internal	✓	
	shouldSwapBack	Internal		

	swapBack	Internal	✓	swapping
	addLp	External	Payable	onlyOwner
	enableTrading	External	✓	onlyOwner
	removeStuckEth	External	✓	onlyOwner
	setFeeReceivers	External	✓	onlyOwner
	setSwapBackSettings	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IERC20Extended (iface)
totalSupply
decimals
symbol
name
balanceOf
transfer
allowance
approve
transferFrom
Ownable
renounceOwnership
owner
<Constructor>
onlyOwner
transferOwnership
Context
_msgSender
_msgData
payable
MOLLARS
swapping
<Constructor>
<Receive Ether>
totalSupply
decimals
symbol
name
allowance
removeStuckEth
approveMax
approve
transferFrom
changeFee
transfer
enableTrading
addLp
setFeeReceivers
setSwapBackSettings
_transferFrom
takeFee
_basicTransfer
swapBack
shouldSwapBack
setAccFee
owner
balanceOf
type
_burnFeeCount
add
swapThreshold
mul
sub
_amount
mul
amount
mul
sub
IDexRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
addLiquidityETH
swapExactETHForTokensSupportingFeeOnTransferTokens
factory
IDexFactory (iface)
createPair
SafeMath (lib)
tryAdd
trySub
tryMul
tryDiv
tryMod
add
sub
mul
div
mod

Summary

MollarsToken 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. 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 5% 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