Cyberscope

Audit Report

Mongoose

December 2023

Network ETH

Address 0x47f7b78436d5d93fa36953fd6540b397d305c0df

Audited by © cyberscope

Analysis

Critical Medium Minor / Informative Pass

Severity	Code	Description	Status
●	ST	Stops Transactions	Passed
●	OTUT	Transfers User's Tokens	Passed
●	ELFM	Exceeds Fees Limit	Passed
●	MT	Mints Tokens	Passed
●	BT	Burns Tokens	Passed
●	BC	Blacklists Addresses	Passed

Diagnostics

Critical Medium Minor / Informative

Severity	Code	Description	Status
	RFV	Redundant Fee Variables	Unresolved
	FSA	Fixed Swap Address	Unresolved
	OCTD	Transfers Contract's Tokens	Unresolved
	RSW	Redundant Storage Writes	Unresolved
	DPI	Decimals Precision Inconsistency	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L07	Missing Events Arithmetic	Unresolved
	L13	Divide before Multiply Operation	Unresolved
	L14	Uninitialized Variables in Local Scope	Unresolved
	L16	Validate Variable Setters	Unresolved
	L19	Stable Compiler Version	Unresolved
	L20	Succeeded Transfer Check	Unresolved

Table of Contents

Review

Contract Name	Mongoose
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://etherscan.io/address/0x47f7b78436d5d93fa36953fd6540b397d305c0df
Address	0x47f7b78436d5d93fa36953fd6540b397d305c0df
Network	ETH
Symbol	MONGOOSE
Decimals	18
Total Supply	420,690,000,000,000

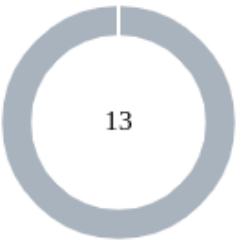
Audit Updates

Initial Audit	05 Dec 2023

Source Files

Filename	SHA256
Mongoose.sol	d7fb428493a7d38732894d5f91257b007cb0460fd8fe1a9b0e97fc516f5c8f46

Findings Breakdown



Critical	0
Medium	0
Minor / Informative	13

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

RFV - Redundant Fee Variables

Criticality	Minor / Informative
Location	Mongoose.sol#L571,602
Status	Unresolved

Description

The contract is currently utilizing two separate variables, `feeswap` and `feesum`, both of which are set to the sum of various fee values (`liquidity` , `marketing` , and `dev` fees). This redundancy is observed in different conditional branches of the contract's logic. In each case, both `feeswap` and `feesum` are assigned the same value, whether it's the sum of the regular taxes or the `launchtax` . This redundancy does not add functional value to the contract and leads to unnecessary complexity and potential confusion. The use of two variables for storing the same value could be streamlined to enhance the contract's efficiency and readability.

Additionally, since both `fee` and `feeswap` variables are assigned the same values under the same conditions the nested `if` statements checking `fee > 0` and `feeswap > 0` are redundant.

```
    feeswap =
          sellTaxes.liquidity +
          sellTaxes.marketing +
          sellTaxes.dev ;
          feesum = feeswap;
          currentTaxes = sellTaxes;
      } else if (!useLaunchFee) {
          feeswap =
          taxes.liquidity +
          taxes.marketing +
          taxes.dev ;
          feesum = feeswap;
          currentTaxes = taxes;
      } else if (useLaunchFee) {
          feeswap = launchtax;
          feesum = launchtax;
      ...
              if (fee > 0) {
      ...
               if (feeswap > 0) {
```

Recommendation

It is recommended to consolidate the fee calculation by utilizing only one of the two variables (`feeswap` or `feesum`) for storing and calculating the total fees. This change would simplify the contract's logic, making it more readable and maintainable. Additionally, it would slightly reduce the gas cost associated with variable storage and assignment in the contract. The remaining variable should be clearly named to reflect its purpose, ensuring that its role in representing the total fee amount is easily understood. This modification would not only streamline the contract but also uphold best practices in smart contract development by avoiding unnecessary redundancy. Additionally the second if statement (`if (feeswap > 0)`) becomes superfluous, as its condition is effectively the same as the first (`if (fee > 0)`). Simplifying these conditions by removing the redundant check would enhance the contract's clarity and efficiency.

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	Mongoose.sol#L472
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() ERC20("Mongoose", "MONGOOSE") {
        ...
        IRouter _router =
IRouter(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
        // Create a pair for this new token
        address _pair =
IFactory(_router.factory()).createPair(address(this),
_router.WETH());
        ...
    }
```

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.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Status	Unresolved

Description

The contract owner has the authority to claim all the balance of the contract. The owner may take advantage of it by calling the `rescueERC20` function.

```
   function rescueERC20(address tokenAdd, uint256 amount)
external {
        IERC20(tokenAdd).transfer(devWallet, amount);
    }
```

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.

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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	Mongoose.sol#L691,738,742,746,751,756
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.

```
providingLiquidity = state;
marketingWallet = newWallet;
devWallet = newWallet;
isearlybuyer[account] = state;
exemptFee[_address] = state;
maxWalletLimit = maxWallet * 10**decimals();
```

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.

DPI - Decimals Precision Inconsistency

Criticality	Minor / Informative
Location	Mongoose.sol#L605,643,648
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.

```
uint256 feeAmount = (amount * feeswap) / 100;
```

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.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Mongoose.sol#L477
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.

```
pair
```

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	Mongoose.sol#L71,73,400,440,612,694,699,700,701,702,707,708,709,710,716,717,731,755
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.

```
mapping(address => uint256) internal _balances
mapping(address => mapping(address => uint256)) internal
_allowances
function WETH() external pure returns (address);
uint256 private  genesis_block

...
```

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	Mongoose.sol#L696,720,734,766
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.

```
tokenLiquidityThreshold = new_amount * 10**decimals()
launchtax = _launchtax
deadline = _deadline
maxWalletLimit = maxWallet * 10**decimals()
```

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.

L13 - Divide before Multiply Operation

Criticality	Minor / Informative
Location	Mongoose.sol#L635,636,643,648
Status	Unresolved

Description

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

```
uint256 unitBalance = deltaBalance / (denominator -
swapTaxes.liquidity)
uint256 ethToAddLiquidityWith = unitBalance *
swapTaxes.liquidity
```

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	Mongoose.sol#L561,562,564
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 feeswap
uint256 feesum
Taxes memory currentTaxes
```

Recommendation

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

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	Mongoose.sol#L738,742
Status	Unresolved

Description

The contract performs operations on variables that have been configured on user-supplied input. These variables are missing of proper check for the case where a value is zero. This can lead to problems when the contract is executed, as certain actions may not be properly handled when the value is zero.

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

Recommendation

By adding the proper check, the contract will not allow the variables to be configured with zero value. This will ensure that the contract can handle all possible input values and avoid unexpected behavior or errors. Hence, it can help to prevent the contract from being exploited or operating unexpectedly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	Mongoose.sol#L17
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 ^0.8.19;
```

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.

L20 - Succeeded Transfer Check

Criticality	Minor / Informative
Location	Mongoose.sol#L774
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(tokenAdd).transfer(devWallet, amount)
```

Recommendation

The contract should check if the result of the transfer methods is successful. The team is advised to check the SafeERC20 library from the Openzeppelin library.

Functions Analysis

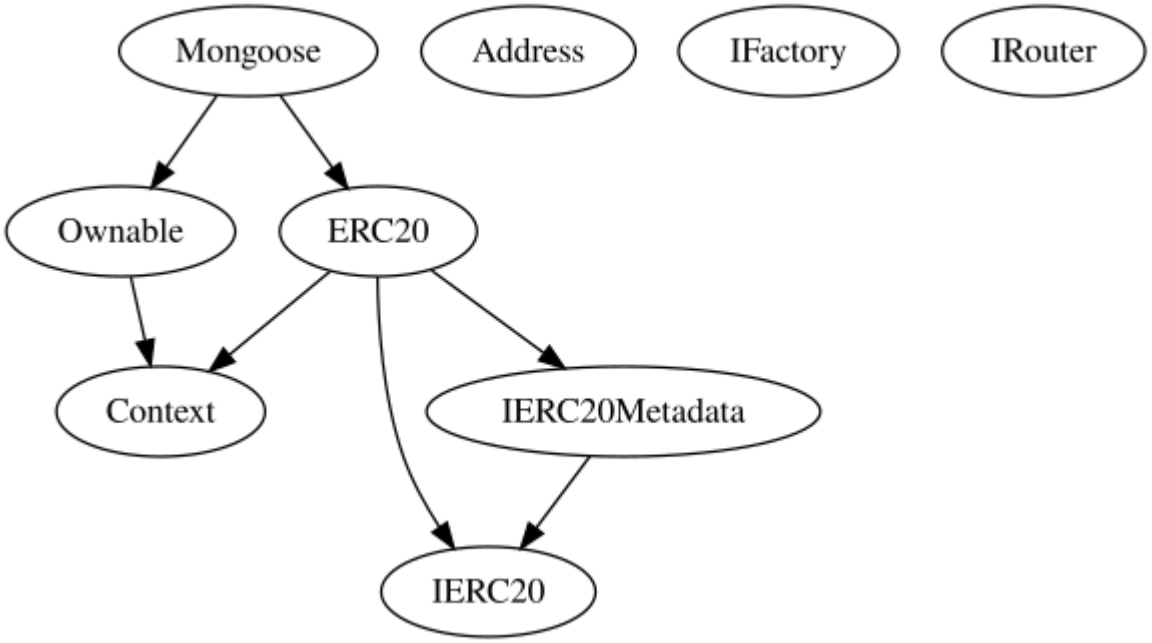
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
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	✓	
	_tokengeneration	Internal	✓	
	_approve	Internal	✓	
	_beforeTokenTransfer	Internal	✓	
Address	Library			
	sendValue	Internal	✓	
Ownable	Implementation	Context		

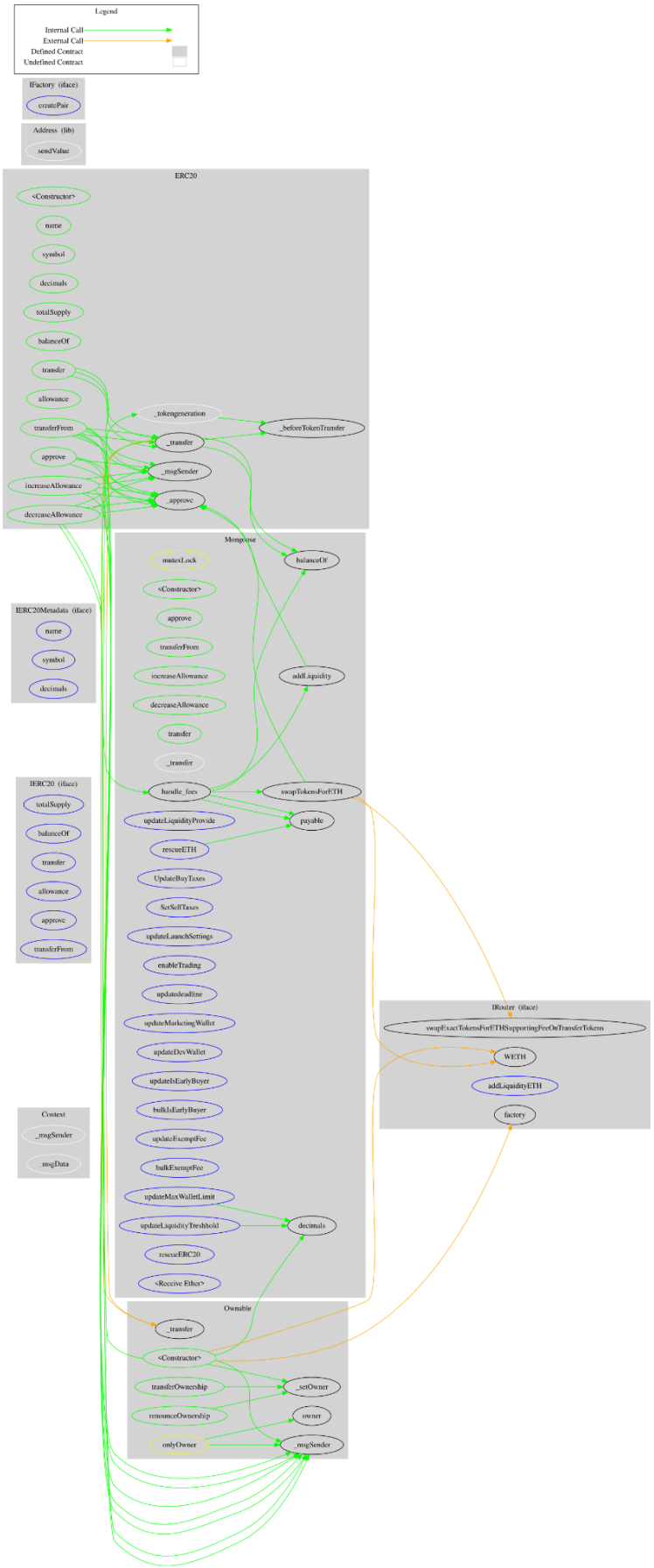
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_setOwner	Private	✓	
IFactory	Interface			
	createPair	External	✓	-
IRouter	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
Mongoose	Implementation	ERC20, Ownable		
		Public	✓	ERC20
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	transfer	Public	✓	-
	_transfer	Internal	✓	

	handle_fees	Private	✓	mutexLock
	swapTokensForETH	Private	✓	
	addLiquidity	Private	✓	
	updateLiquidityProvide	External	✓	onlyOwner
	updateLiquidityTreshhold	External	✓	onlyOwner
	UpdateBuyTaxes	External	✓	onlyOwner
	SetSellTaxes	External	✓	onlyOwner
	updateLaunchSettings	External	✓	onlyOwner
	enableTrading	External	✓	onlyOwner
	updatedeadline	External	✓	onlyOwner
	updateMarketingWallet	External	✓	onlyOwner
	updateDevWallet	External	✓	onlyOwner
	updateIsEarlyBuyer	External	✓	onlyOwner
	bulkIsEarlyBuyer	External	✓	onlyOwner
	updateExemptFee	External	✓	onlyOwner
	bulkExemptFee	External	✓	onlyOwner
	updateMaxWalletLimit	External	✓	onlyOwner
	rescueETH	External	✓	-
	rescueERC20	External	✓	-
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IFactory (iface)
createPair
Address (lib)
sendValue
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
transfer
allowance
transferFrom
approve
increaseAllowance
decreaseAllowance
_tokengeneration
_transfer
_beforeTokenTransfer
_msgSender
_approve
Mongoose
mutexLock
<Constructor>
approve
transferFrom
increaseAllowance
decreaseAllowance
transfer
_transfer
handle_fees
updateLiquidityProvide
rescueETH
UpdateBuyTaxes
SetSellTaxes
updateLaunchSettings
enableTrading
updatedeadline
updateMarketingWallet
updateDevWallet
updateIsEarlyBuyer
bulkIsEarlyBuyer
updateExemptFee
bulkExemptFee
updateMaxWalletLimit
updateLiquidityTreshhold
rescueERC20
<Receive Ether>
balanceOf
addLiquidity
swapTokensForETH
payable
decimals
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender
_msgData
IRouter (iface)
swapExactTokensForETHSupportingFeeOnTransferTokens
WETH
addLiquidityETH
factory
Ownable
_transfer
<Constructor>
transferOwnership
renounceOwnership
onlyOwner
_setOwner
owner
_msgSender

Summary

Mongoose contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. 12 Days Of Xmas is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues. The contract Owner can access some admin functions that can not be used in a malicious way to disturb the users' transactions.

The contract's ownership has been renounced. The information regarding the transaction can be accessed through the following link: https://etherscan.io/tx/0xe65151f84eb7e2cafd2cc0f4892bf24c250e0f9def6d2e3c628c96ecc1111976. The fees are locked at 1% on buy, sell and transfer transactions.

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