Cyberscope

Audit Report

MOCHI on BASE

May 2024

Network	BASE
Address	0xF6e932Ca12afa26665dC4dDE7e27be02A7c02e50
Audited by	© cyberscope

Table of Contents

Review

Contract Name	MOCHI
Compiler Version	v0.8.22+commit.4fc1097e
Optimization	200 runs
Explorer	https://basescan.org/address/0xf6e932ca12afa26665dc4dde7e27be02a7c02e50
Address	0xf6e932ca12afa26665dc4dde7e27be02a7c02e50
Network	BASE
Symbol	MOCHI
Decimals	18
Total Supply	1,000,000,000,000

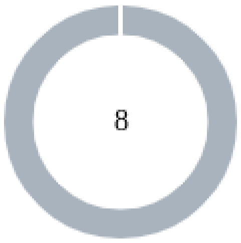
Audit Updates

Initial Audit	19 May 2024

Source Files

Filename	SHA256
MOCHI.sol	6f0bd1326ac734e52ddcb69c1cbb46e11154468434796b43805036d086249603

Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	8

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	8	0	0	0

Diagnostics

🔴 Critical 🟠 Medium ● Minor / Informative

Severity	Code	Description	Status
●	MEE	Missing Events Emission	Unresolved
●	PLPI	Potential Liquidity Provision Inadequacy	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	L02	State Variables could be Declared Constant	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L15	Local Scope Variable Shadowing	Unresolved
●	L19	Stable Compiler Version	Unresolved
●	L20	Succeeded Transfer Check	Unresolved

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	MOCHI.sol#L792,842
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 setSwapTokensAtAmount(uint256 amount) external onlyOwner {
        uint256 _totalSupply = totalSupply();
        require(amount >= (_totalSupply * 1) / 100000, "Swap amount cannot be lower than 0.001% total supply.");
        require(amount <= (_totalSupply * 5) / 1000, "Swap amount cannot be higher than 0.5% total supply.");
        swapTokensAtAmount = amount;
    }

    function setTaxEnabled(bool value) external onlyOwner {
        taxEnabled = value;
    }

    function setSwapTokensAtAmount(uint256 amount) external onlyOwner {
        uint256 _totalSupply = totalSupply();
        require(
            amount >= (_totalSupply * 1) / 100000,
            "Swap amount cannot be lower than 0.001% total supply."
        );
        require(
            amount <= (_totalSupply * 5) / 1000,
            "Swap amount cannot be higher than 0.5% total supply."
        );
        swapTokensAtAmount = amount;
    }

    function setBuyFees(uint256 _buyFee) external onlyOwner {
        require(_buyFee <= MAX_FEE, "Must keep fees at 3% or less");
        buyTotalFees = _buyFee;
    }

    function setSellFees(uint256 _sellFee) external onlyOwner {
        require(_sellFee <= MAX_FEE, "Must keep fees at 3% or less");
        sellTotalFees = _sellFee;
    }
```

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.

PLPI - Potential Liquidity Provision Inadequacy

Criticality	Minor / Informative
Location	MOCHI.sol#L950
Status	Unresolved

Description

The contract operates under the assumption that liquidity is consistently provided to the pair between the contract's token and the native currency. However, there is a possibility that liquidity is provided to a different pair. This inadequacy in liquidity provision in the main pair could expose the contract to risks. Specifically, during eligible transactions, where the contract attempts to swap tokens with the main pair, a failure may occur if liquidity has been added to a pair other than the primary one. Consequently, transactions triggering the swap functionality will result in a revert.

```
uniswapV2Router.swapExactTokensForETH(
    swapAmount,
    0,
    path,
    owner(),
    block.timestamp
);
```

Recommendation

The team is advised to implement a runtime mechanism to check if the pair has adequate liquidity provisions. This feature allows the contract to omit token swaps if the pair does not have adequate liquidity provisions, significantly minimizing the risk of potential failures.

Furthermore, the team could ensure the contract has the capability to switch its active pair in case liquidity is added to another pair.

Additionally, the contract could be designed to tolerate potential reverts from the swap functionality, especially when it is a part of the main transfer flow. This can be achieved by executing the contract's token swaps in a non-reversible manner, thereby ensuring a more resilient and predictable operation.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	MOCHI.sol#L742
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.

```
public initialized;
```

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	MOCHI.sol#L716,799,804
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)
function setBuyFees(uint256 _buyFee)
function setSellFees(uint256 _sellFee)
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	MOCHI.sol#L531
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function _burn(address account, uint256 value) internal {
        if (account == address(0)) {
            revert ERC20InvalidSender(address(0));
        }
        _update(account, address(0), value);
    }
```

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.

L15 - Local Scope Variable Shadowing

Criticality	Minor / Informative
Location	MOCHI.sol#L793
Status	Unresolved

Description

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

```
uint256 _totalSupply = totalSupply();
```

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.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	MOCHI.sol#L6
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 ^0.8.20;
```

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	MOCHI.sol#L884
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(tkn).transfer(msg.sender, 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

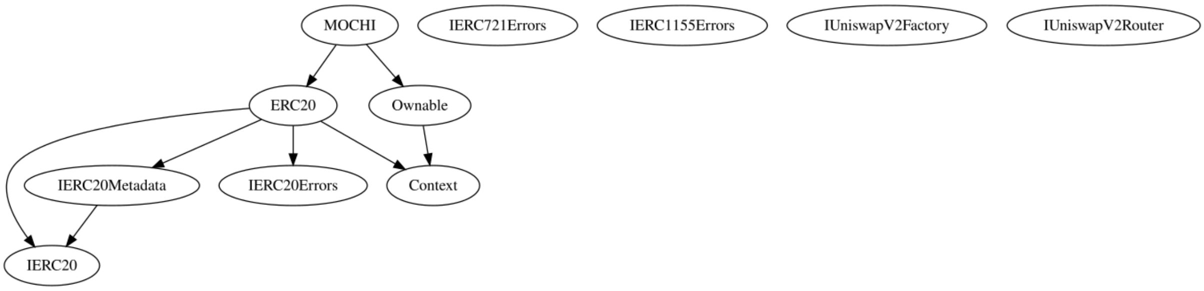
Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
IERC20Metadata	Interface	IERC20		
	name	External		-
	symbol	External		-
	decimals	External		-
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
IERC20Errors	Interface			

IERC721Errors	Interface			
IERC1155Errors	Interface			
ERC20	Implementation	Context, IERC20, IERC20Metadata, IERC20Errors		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_transfer	Internal	✓	
	_update	Internal	✓	
	_mint	Internal	✓	
	_burn	Internal	✓	
	_approve	Internal	✓	
	_approve	Internal	✓	

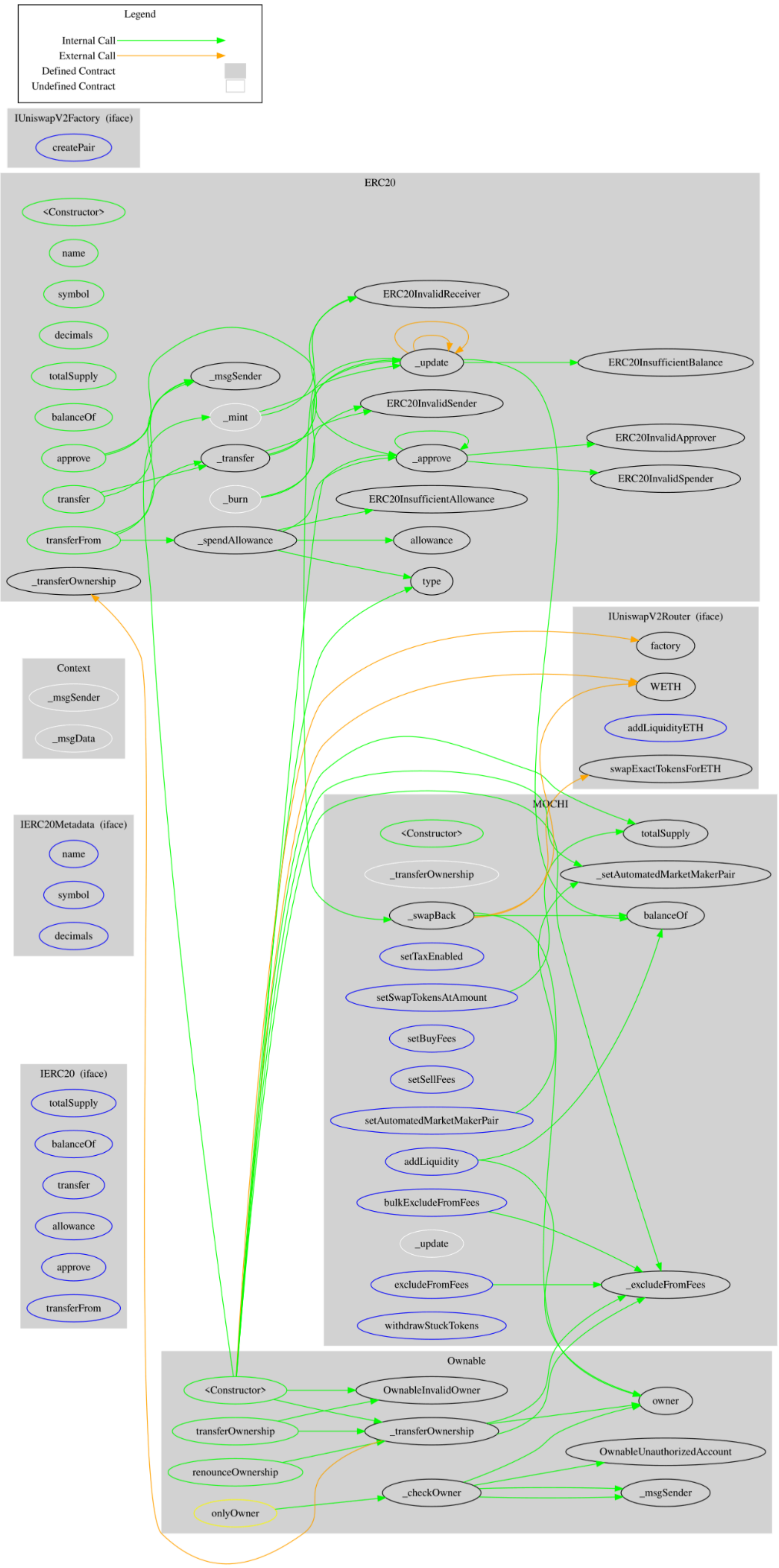
	_spendAllowance	Internal	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	_checkOwner	Internal		
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	_transferOwnership	Internal	✓	
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IUniswapV2Router	Interface			
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
	swapExactTokensForETH	External	✓	-
MOCHI	Implementation	Ownable, ERC20		
		Public	✓	Ownable ERC20
	_transferOwnership	Internal	✓	

	addLiquidity	External	Payable	onlyOwner
	setTaxEnabled	External	✓	onlyOwner
	setSwapTokensAtAmount	External	✓	onlyOwner
	setBuyFees	External	✓	onlyOwner
	setSellFees	External	✓	onlyOwner
	_excludeFromFees	Internal	✓	
	excludeFromFees	External	✓	onlyOwner
	bulkExcludeFromFees	External	✓	onlyOwner
	_setAutomatedMarketMakerPair	Internal	✓	
	setAutomatedMarketMakerPair	External	✓	onlyOwner
	_update	Internal	✓	
	_swapBack	Internal	✓	
	withdrawStuckTokens	External	✓	onlyOwner

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
IUniswapV2Factory (iface)
createPair
ERC20
<Constructor>
name
symbol
decimals
totalSupply
balanceOf
approve
transfer
transferFrom
_transferOwnership
_msgSender
_mint
_transfer
_burn
_spendAllowance
ERC20InvalidReceiver
_update
ERC20InvalidSender
_approve
ERC20InsufficientAllowance
allowance
type
ERC20InsufficientBalance
ERC20InvalidApprover
ERC20InvalidSpender
Context
_msgSender
_msgData
IERC20Metadata (iface)
name
symbol
decimals
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
IUniswapV2Router (iface)
factory
WETH
addLiquidityETH
swapExactTokensForETH
MOCHI
<Constructor>
_transferOwnership
_swapBack
setTaxEnabled
setSwapTokensAtAmount
setBuyFees
setSellFees
setAutomatedMarketMakerPair
addLiquidity
bulkExcludeFromFees
_update
excludeFromFees
withdrawStuckTokens
totalSupply
_setAutomatedMarketMakerPair
balanceOf
_excludeFromFees
Ownable
<Constructor>
transferOwnership
renounceOwnership
onlyOwner
OwnableInvalidOwner
_transferOwnership
_checkOwner
owner
OwnableUnauthorizedAccount
_msgSender

Summary

MOCHI on BASE contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. MOCHI on BASE 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. There is also a limit of max 3% fee on buy and sell 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