Cyberscope

Audit Report

CODIFI

December 2023

Network	ETH
Address	0x7aBf508C022D38F8eCae6258129a96545086FEF6
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
	CCR	Contract Centralization Risk	Unresolved
	RRS	Redundant Require Statement	Unresolved
	RSML	Redundant SafeMath Library	Unresolved
	IDI	Immutable Declaration Improvement	Unresolved
	L02	State Variables could be Declared Constant	Unresolved
	L04	Conformance to Solidity Naming Conventions	Unresolved
	L05	Unused State Variable	Unresolved
	L07	Missing Events Arithmetic	Unresolved

Table of Contents

Review

Contract Name	Codifi
Compiler Version	v0.8.19+commit.7dd6d404
Optimization	200 runs
Explorer	https://etherscan.io/address/0x7abf508c022d38f8ecae6258129a96545086fef6
Address	0x7abf508c022d38f8ecae6258129a96545086fef6
Network	ETH
Symbol	COD
Decimals	18
Total Supply	100,000,000

Audit Updates

Initial Audit	19 Dec 2023

Source Files

Filename	SHA256
Codifi.sol	239432660af2230ae209c0bf4affd161ccea00ea6e677fad20c3f1feb3de2acf

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

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	Codifi.sol#L296
Status	Unresolved

Description

The `startTrading` function introduces a centralization risk due to the requirement for the contract owner to manually supply an amount higher than `_tTotal.mul(_initialBuyTax).div(100)` to the contract. The need for owner-initiated action before trading can commence places substantial control in the hands of a single entity, potentially leading to Single Point of Control issues. It also creates a Trust Dependency, as the other participants must rely on the contract owner to fund the contract appropriately. Furthermore, this manual process can lead to Operational Delays and is Vulnerable to Attacks targeting the owner's address. It also contributes to Decentralization Erosion, as the decentralized ecosystem ideally seeks to minimize such centralized points of influence.

```
function startTrading() external onlyOwner {
    require(!swapEnabled,"trading already open");
    uint256 tokenAmount =
balanceOf(address(this)).sub(_tTotal.mul(_initialBuyTax).div(10
0));
    _approve(address(this), address(uniswapV2Router), _tTotal);
    uniswapV2Router.addLiquidityETH{value:
address(this).balance} (
        address(this),
        tokenAmount,
        0,
        0,
        owner(),
        block.timestamp
    );
    IERC20(uniswapV2Pair).approve(address(uniswapV2Router),
type(uint).max);
    swapEnabled = true;
}
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	Codifi.sol#L46
Status	Unresolved

Description

The contract utilizes a `require` statement within the `add` function aiming to prevent overflow errors. This function is designed based on the SafeMath library's principles. In Solidity version 0.8.0 and later, arithmetic operations revert on overflow and underflow, making the overflow check within the function redundant. This redundancy could lead to extra gas costs and increased complexity without providing additional security.

```
function add(uint256 a, uint256 b) internal pure returns
(uint256) {
      uint256 c = a + b;
      require(c >= a, "SafeMath: addition overflow");
      return c;
}
```

Recommendation

It is recommended to remove the `require` statement from the add function since the contract is using a Solidity pragma version equal to or greater than 0.8.0. By doing so, the contract will leverage the built-in overflow and underflow checks provided by the Solidity language itself, simplifying the code and reducing gas consumption. This change will uphold the contract's integrity in handling arithmetic operations while optimizing for efficiency and cost-effectiveness.

RSML - Redundant SafeMath Library

Criticality	Minor / Informative
Location	Codifi.sol
Status	Unresolved

Description

SafeMath is a popular Solidity library that provides a set of functions for performing common arithmetic operations in a way that is resistant to integer overflows and underflows.

Starting with Solidity versions that are greater than or equal to 0.8.0, the arithmetic operations revert to underflow and overflow. As a result, the native functionality of the Solidity operations replaces the SafeMath library. Hence, the usage of the SafeMath library adds complexity, overhead and increases gas consumption unnecessarily.

```
library SafeMath {...}
```

Recommendation

The team is advised to remove the SafeMath library. Since the version of the contract is greater than `0.8.0` then the pure Solidity arithmetic operations produce the same result.

If the previous functionality is required, then the contract could exploit the `unchecked { ... }` statement.

Read more about the breaking change on https://docs.soliditylang.org/en/v0.8.16/080-breaking-changes.html#solidity-v0-8-0-breaking-changes.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	Codifi.sol#L158
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.

```
uniswapV2Pair
```

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.

L02 - State Variables could be Declared Constant

Criticality	Minor / Informative
Location	Codifi.sol#L152,153
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 _maxTxAmount = 1000000 * 10**_decimals;
uint256 public _maxWalletSize = 1000000 * 10**_decimals
```

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	Codifi.sol#L120,148,149,150,151,152,153,154,155
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);
uint8 private constant _decimals = 18;
uint256 private constant _tTotal = 100000000 * 10**_decimals;
string private constant _name = unicode"CODIFI";
string private constant _symbol = unicode"COD";
uint256 public _maxTxAmount = 1000000 * 10**_decimals;
uint256 public _maxWalletSize = 1000000 * 10**_decimals;
uint256 public constant _taxSwapThreshold= 100000 *
10**_decimals;
uint256 public constant _maxTaxSwap= 1000000 * 10**_decimals;
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.
Find more information on the Solidity documentation
https://docs.soliditylang.org/en/v0.8.17/style-guide.html#naming-convention.

L05 - Unused State Variable

Criticality	Minor / Informative
Location	Codifi.sol#L137
Status	Unresolved

Description

An unused state variable is a state variable that is declared in the contract, but is never used in any of the contract's functions. This can happen if the state variable was originally intended to be used, but was later removed or never used.

Unused state variables can create clutter in the contract and make it more difficult to understand and maintain. They can also increase the size of the contract and the cost of deploying and interacting with it.

```
address private constant deadWallet = address(0xdead);
```

Recommendation

To avoid creating unused state variables, it's important to carefully consider the state variables that are needed for the contract's functionality, and to remove any that are no longer needed. This can help improve the clarity and efficiency of the contract.

L07 - Missing Events Arithmetic

Criticality	Minor / Informative
Location	Codifi.sol#L325,326
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.

```
_finalBuyTax = _value;
_finalSellTax = _value;
```

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.

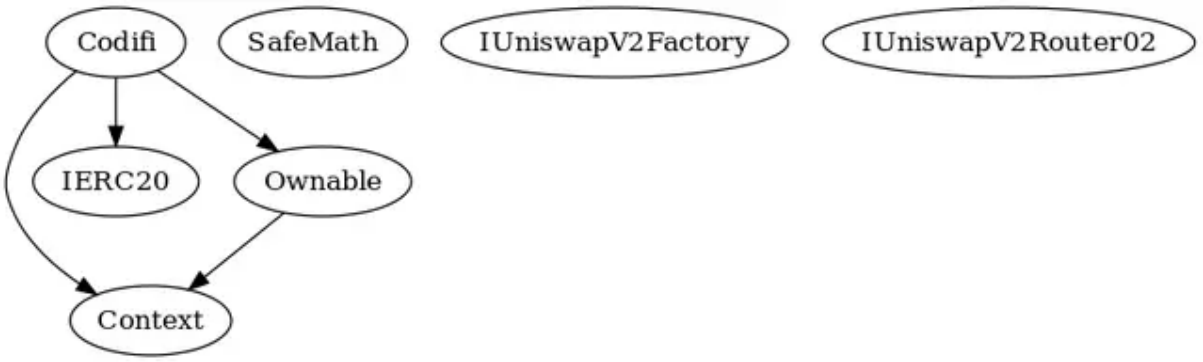
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
Context	Implementation			
	_msgSender	Internal		
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		

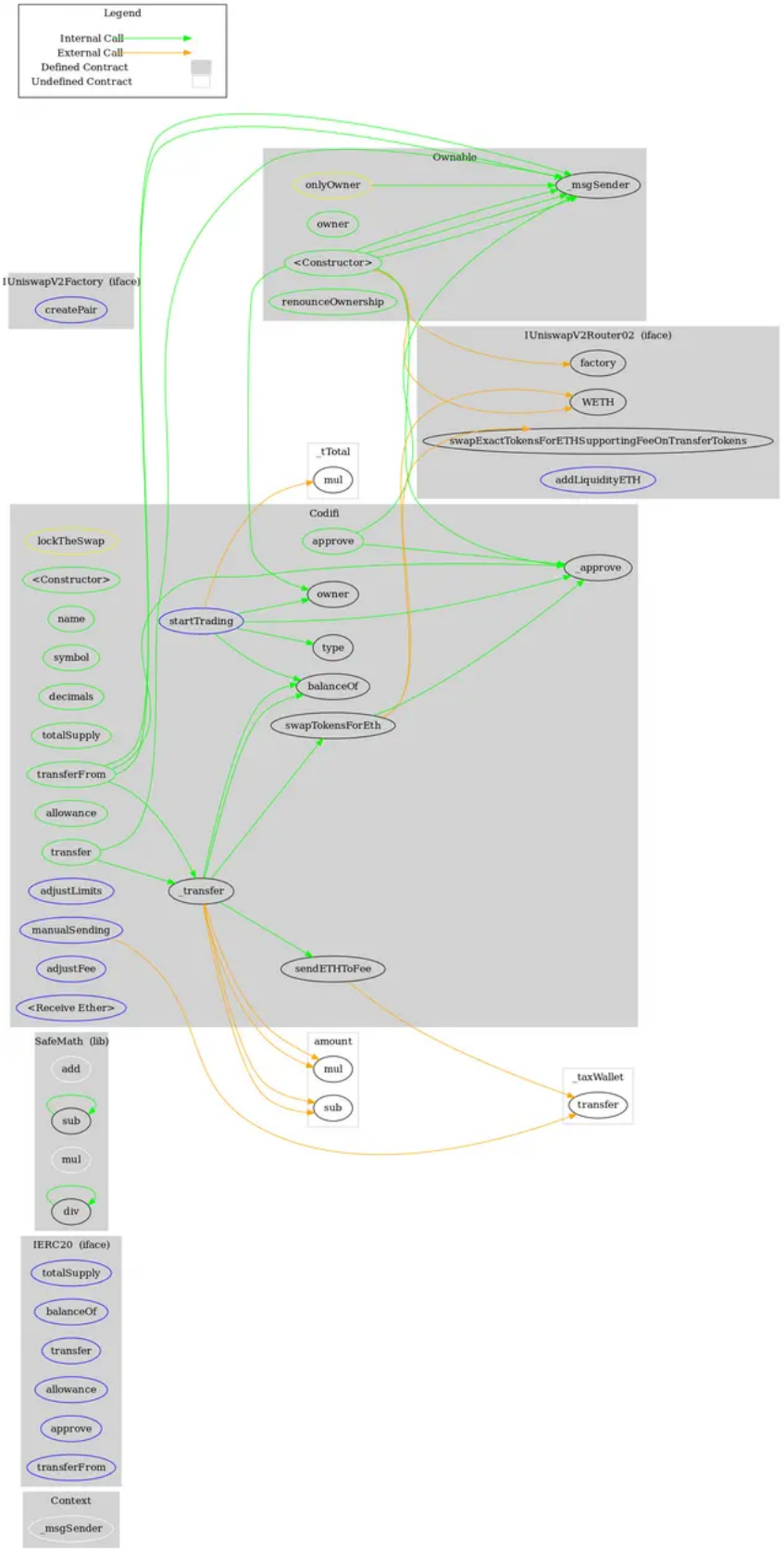
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			
	createPair	External	✓	-
IUniswapV2Router02	Interface			
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
	factory	External		-
	WETH	External		-
	addLiquidityETH	External	Payable	-
Codifi	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	Public		-
	symbol	Public		-
	decimals	Public		-
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-

	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	_approve	Private	✓	
	_transfer	Private	✓	
	sendETHToFee	Private	✓	
	swapTokensForEth	Private	✓	lockTheSwap
	startTrading	External	✓	onlyOwner
	adjustLimits	External	✓	onlyOwner
	manualSending	External	✓	-
	adjustFee	External	✓	onlyOwner
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
msgSender
owner
<Constructor>
renounceOwnership
IUniswapV2Factory (iface)
createPair
IUniswapV2Router02 (iface)
factory
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
addLiquidityETH
_tTotal
mul
Codifi
lockTheSwap
<Constructor>
name
symbol
decimals
totalSupply
transferFrom
allowance
transfer
adjustLimits
manualSending
adjustFee
<Receive Ether>
approve
owner
startTrading
type
balanceOf
swapTokensForEth
_transfer
sendETHToFee
amount
mul
sub
_taxWallet
transfer
SafeMath (lib)
add
sub
mul
div
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom
Context
_msgSender

Summary

CODIFI contract implements a token mechanism. This audit investigates security issues, business logic concerns and potential improvements. CODIFI 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 15% 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