Cyberscope

Audit Report

Flokinomics 2.0

July 2023

Network	ETH
Address	0x1D3B169406091C28eE8f166b81F6855672564e4e
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
●	RC	Repetitive Calculations	Unresolved
●	DKO	Delete Keyword Optimization	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	OCTD	Transfers Contract's Tokens	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	FSA	Fixed Swap Address	Unresolved
●	EIS	Excessively Integer Size	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

Table of Contents

Review

Contract Name	ERC20FLOKINOMICS2
Compiler Version	v0.8.18+commit.87f61d96
Optimization	200 runs
Explorer	https://etherscan.io/address/0x1d3b169406091c28ee8f166b81f6855672564e4e
Address	0x1d3b169406091c28ee8f166b81f6855672564e4e
Network	ETH
Symbol	Flokin2
Decimals	18
Total Supply	1,000,000,000

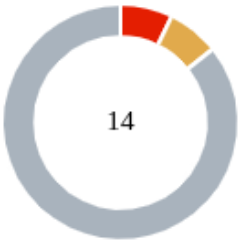
Audit Updates

Initial Audit	11 Jul 2023

Source Files

Filename	SHA256
ERC20FLOKINOMICS2.sol	c350af1f5eaee05f849e651130ebe3a92a465065fcd6be288e47ad7ed6460dfc

Findings Breakdown



Critical	1
Medium	1
Minor / Informative	12

Severity	Unresolved	Acknowledged	Resolved	Other
Critical	1	0	0	0
Medium	1	0	0	0
Minor / Informative	12	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	ERC20FLOKINOMICS2.sol#L358
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.

```
if(!isFeeExempt[from] && !isFeeExempt[to]){
    require(tradingOpen,"Trading not open yet");
}
```

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. 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.

ELFM - Exceeds Fees Limit

Criticality	Medium
Location	ERC20FLOKINOMICS2.sol#L315,215,304,266,281
Status	Unresolved

Description

The contract owner has the power to potentially impose excessive fees on transactions by using the set_All_Fees function. Specifically, the owner can set the _fee_marketing to the max value of 20 by calling the set_All_Fees function. When the goLive function is called, the buyMultiplier and sellMultiplier are set to 50 and 200 respectively.

In the _getValues function, these multipliers are applied to the respective fees, which could result in a maximum tReflection fees of 40%, since a multiplier value of 200 is used for selling transactions. This means that for selling trasactions there will be: _fee_reflection = 20 and multiplier = 200 , which set a fee of 40%

Furthermore, until the setMultipliers function is called to adjust the multipliers, the total fee for a selling transactions remain at this high level. When setMultipliers is called, the maximum of 100 can be set to buyMultiplier or sellMultiplier , meaning that the total fee for a transaction could be as high as 20%.

This ability to start with such high fees could be used by the contract owner to increase the tax fee beyond the allowed limit of 25% by calling the set_All_Fees function with the max allowed value.

```
function set_All_Fees(uint256 Reflection_Fees, uint256
Marketing_Fee) external onlyOwner {
    uint256 total_fees =  Marketing_Fee + Reflection_Fees;
    require(total_fees <= 20, "Max fee allowed is 20%");
    _setAllFees( Marketing_Fee, Reflection_Fees);
}

function goLive() external onlyOwner {
    ...
    buyMultiplier = 50;
    sellMultiplier = 200;
    transferMultiplier = 0;
}

function setMultipliers(uint256 _buy, uint256 _sell, uint256
_trans) external onlyOwner {

    require(_buy <= 100, "Max buy multiplier allowed is 1x");
    require(_sell <= 100, "Max sell multiplier allowed is 1x");
    require(_trans <= 100, "Max transfer multiplier allowed is
1x");

    sellMultiplier = _sell;
    buyMultiplier = _buy;
    transferMultiplier = _trans;
}
```

```
function _getValues(uint256 tAmount, address recipient, address
sender) private view returns (
        uint256 rAmount, uint256 rTransferAmount, uint256
rReflection,
        uint256 tTransferAmount, uint256 tMarketing, uint256
tReflection) {

        uint256 multiplier = transferMultiplier;

        if(recipient == uniswapV2Pair) {
            multiplier = sellMultiplier;
        } else if(sender == uniswapV2Pair) {
            multiplier = buyMultiplier;
        }

        tMarketing = ( tAmount * _fee_marketing ) * multiplier
/ (_fee_denominator * 100);
        tReflection = ( tAmount * _fee_reflection ) *
multiplier  / (_fee_denominator * 100);

        tTransferAmount = tAmount - ( tMarketing +
tReflection);
        rReflection = tReflection * _getRate();
        rAmount = tAmount * _getRate();
        rTransferAmount = tTransferAmount * _getRate();
}
```

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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

RC - Repetitive Calculations

Criticality	Minor / Informative
Location	ERC20FLOKINOMICS2.sol#L282
Status	Unresolved

Description

The contract contains methods with multiple occurrences of the same calculation being performed. The calculation is repeated without utilizing a variable to store its result, which leads to redundant code, hinders code readability, and increases gas consumption. Each repetition of the calculation requires computational resources and can impact the performance of the contract, especially if the calculation is resource-intensive.

Specifically, in the `_getValues` function, there are multiple instances where the `_getRate` function is called to perform the same calculation.

```
    function _getValues(uint256 tAmount, address recipient, address
sender) private view returns (
        ...

        tTransferAmount = tAmount - ( tMarketing + tReflection);
        rReflection = tReflection * _getRate();
        rAmount = tAmount * _getRate();
        rTransferAmount = tTransferAmount * _getRate();
    }
```

Recommendation

To address this finding and enhance the efficiency and maintainability of the contract, it is recommended to refactor the code by assigning the calculation result to a variable once and then utilizing that variable throughout the method. By storing the calculation result in a variable, the contract eliminates the need for redundant calculations and optimizes code execution.

Refactoring the code to assign the calculation result to a variable has several benefits. It improves code readability by making the purpose and intent of the calculation explicit. It

also reduces code redundancy, making the method more concise, easier to maintain, and gas effective. Additionally, by performing the calculation once and reusing the variable, the contract improves performance by avoiding unnecessary computations

DKO - Delete Keyword Optimization

Criticality	Minor / Informative
Location	ERC20FLOKINOMICS2.sol#L325
Status	Unresolved

Description

The contract resets variables to the default state by setting the initial values. Setting values to state variables increases the gas cost.

```
function removeAllFee() private {
    _fee_marketing_old = _fee_marketing;
    _fee_reflection_old = _fee_reflection;

    _setAllFees(0,0);
}
```

Recommendation

The team is advised to use the `delete` keyword instead of setting variables. This can be more efficient than setting the variable to a new value, using delete can reduce the gas cost associated with storing data on the blockchain.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	ERC20FLOKINOMICS2.sol#L224,361
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 setSwapSettings(bool _status, uint256 _threshold)
external onlyOwner {
    require(_threshold > 0,"swap threshold cannot be 0");
    swapAndLiquifyEnabled = _status;
    swapThreshold = _threshold;
}

function _transfer(address from, address to, uint256 amount)
private {
    ...
    if(!inSwapAndLiquify && from != uniswapV2Pair &&
swapAndLiquifyEnabled && balanceOf(address(this)) >
swapThreshold){
        swapTokensForEth(swapThreshold);
    }
    ...
}
```

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 total supply. Hence, the contract will guarantee that it cannot accumulate a huge amount of tokens in order to sell them.

OCTD - Transfers Contract's Tokens

Criticality	Minor / Informative
Location	ERC20FLOKINOMICS2.sol#L241
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 `clearStuckToken` function.

```
function clearStuckToken(address tokenAddress, uint256 tokens)
external onlyOwner returns (bool success) {
    if(tokens == 0){
        tokens = IERC20(tokenAddress).balanceOf(address(this));
    }
    return IERC20(tokenAddress).transfer(msg.sender, tokens);
}
```

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. Some suggestions are:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-sign wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.
- Renouncing the ownership will eliminate the threats but it is non-reversible.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	ERC20FLOKINOMICS2.sol#L193,202,304,315,
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 excludeFromReward(address account) external
onlyOwner {
        require(!_isExcluded[account], "Account is already
excluded");
        if(_balance_reflected[account] > 0) {
            _balance_total[account] =
tokenFromReflection(_balance_reflected[account]);
        }
        _isExcluded[account] = true;
        _excluded.push(account);
    }

    function includeInReward(address account) external
onlyOwner {
        require(_isExcluded[account], "Account is already
included");

        ...
    }

    function setMultipliers(uint256 _buy, uint256 _sell,
uint256 _trans) external onlyOwner {
    ...
    }

    function set_All_Fees(uint256 Reflection_Fees, uint256
Marketing_Fee) external onlyOwner {
        uint256 total_fees =  Marketing_Fee + Reflection_Fees;
        require(total_fees <= 20, "Max fee allowed is 20%");
        _setAllFees( Marketing_Fee, Reflection_Fees);
    }
```

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.

FSA - Fixed Swap Address

Criticality	Minor / Informative
Location	ERC20FLOKINOMICS2.sol#L134
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 () {
    ...
    IUniswapV2Router _uniswapV2Router =
IUniswapV2Router(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
    uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(t
his), _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.

EIS - Excessively Integer Size

Criticality	Minor / Informative
Location	ERC20FLOKINOMICS2.sol#L88,100
Status	Unresolved

Description

The contract is using a bigger unsigned integer data type that the maximum size that is required. By using an unsigned integer data type larger than necessary, the smart contract consumes more storage space and requires additional computational resources for calculations and operations involving these variables. This can result in higher transaction costs, longer execution times, and potential scalability bottlenecks.

```
uint256 public constant decimals = 18;

uint256 public constant _fee_denominator = 100;
```

Recommendation

To address the inefficiency associated with using an oversized unsigned integer data type, it is recommended to accurately determine the required size based on the range of values the variable needs to represent.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	ERC20FLOKINOMICS2.sol#L63,75,76,79,84,87,92,93,94,96,97,98,110,183,224,230,304,315
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);
mapping (address => uint256) public _balance_reflected
mapping (address => uint256) public _balance_total
mapping (address => bool) public _isExcluded
address constant deadAddress =
0x000000000000000000000000000000000000dEaD
uint256 private _supply_reflected = (MAX - (MAX % totalSupply))
uint256 public _fee_reflection = 5
uint256 private _fee_reflection_old = _fee_reflection
uint256 public _contractReflectionStored = 0
uint256 public _fee_marketing = 5
uint256 private _fee_marketing_old = _fee_marketing
address payable public _wallet_marketing
address[] public _excluded
address _newMarketing

...
```

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	ERC20FLOKINOMICS2.sol#L227,310
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.

```
swapThreshold = _threshold
sellMultiplier = _sell
```

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	ERC20FLOKINOMICS2.sol#L279,282
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.

```
tReflection = ( tAmount * _fee_reflection ) * multiplier  /
(_fee_denominator * 100)
rReflection = tReflection * _getRate()
```

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	ERC20FLOKINOMICS2.sol#L231
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 i
```

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	ERC20FLOKINOMICS2.sol#L184
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.

```
_wallet_marketing = payable(_newMarketing)
```

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.

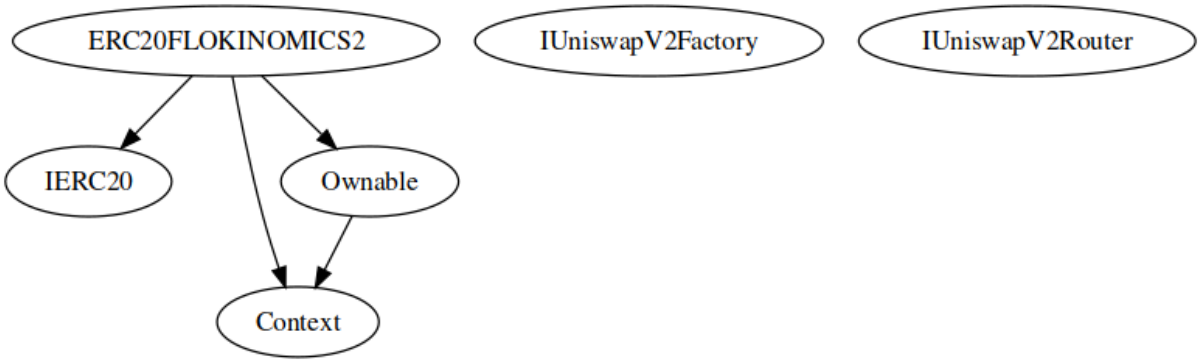
Functions Analysis

Contract	Type	Bases		
	Function Name	**Visibility**	**Mutability**	**Modifiers**
IERC20	Interface			
	totalSupply	External		-
	balanceOf	External		-
	transfer	External	✓	-
	allowance	External		-
	approve	External	✓	-
	transferFrom	External	✓	-
Context	Implementation			
	_msgSender	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	transferOwnership	Public	✓	onlyOwner
	renounceOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			

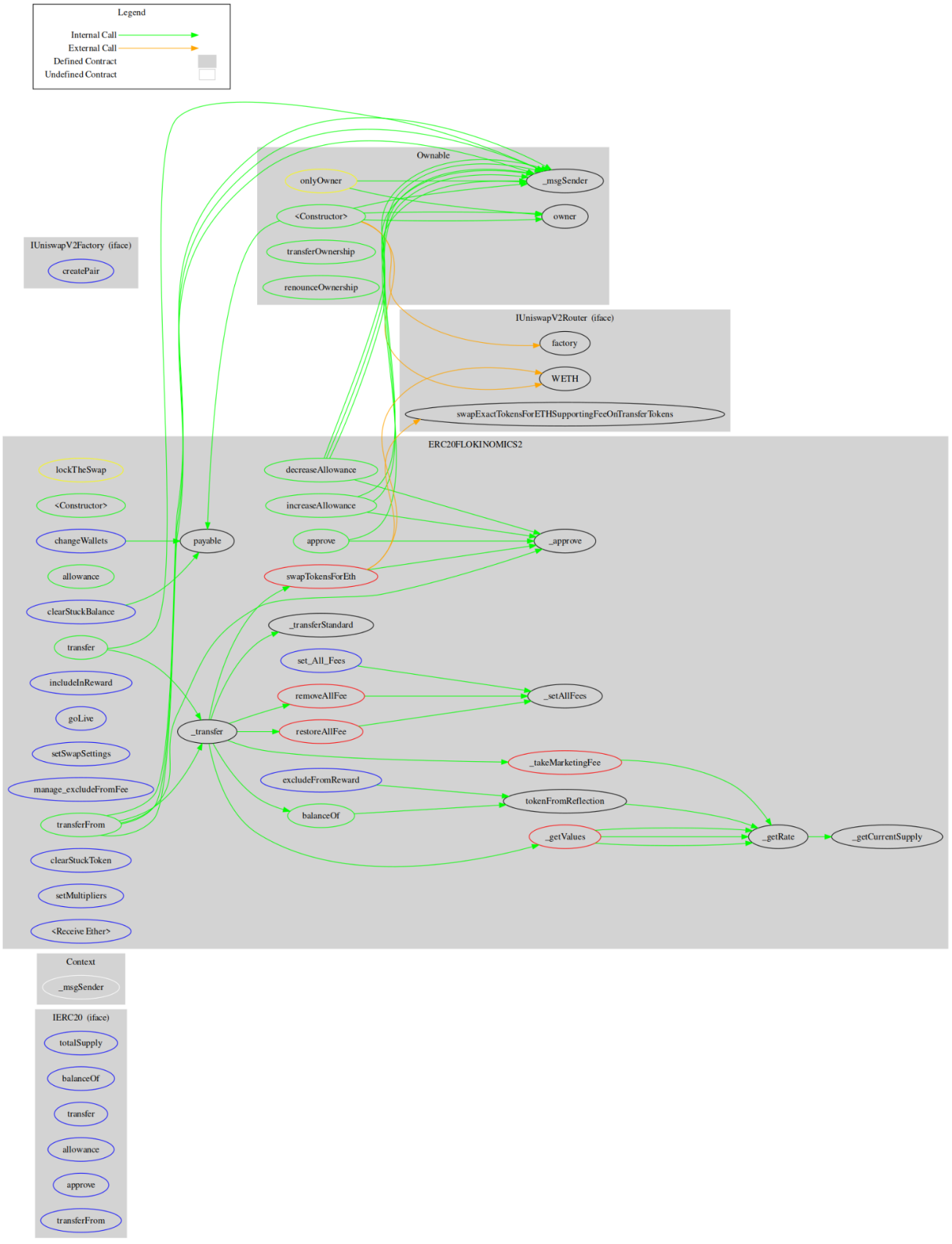
	createPair	External	✓	-
IUniswapV2Router	Interface			
	factory	External		-
	WETH	External		-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
ERC20FLOKINOMICS2	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	changeWallets	External	✓	onlyOwner
	tokenFromReflection	Public		-
	excludeFromReward	External	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	goLive	External	✓	onlyOwner
	setSwapSettings	External	✓	onlyOwner

	manage_excludeFromFee	External	✓	onlyOwner
	clearStuckBalance	External	✓	onlyOwner
	clearStuckToken	External	✓	onlyOwner
	_getRate	Private		
	_getCurrentSupply	Private		
	_getValues	Private		
	_takeMarketingFee	Private	✓	
	_setAllFees	Private	✓	
	setMultipliers	External	✓	onlyOwner
	set_All_Fees	External	✓	onlyOwner
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	swapTokensForEth	Private	✓	lockTheSwap
	_approve	Private	✓	
	_transfer	Private	✓	
	_transferStandard	Private	✓	
		External	Payable	-

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
Ownable
onlyOwner
_msgSender
<Constructor>
owner
transferOwnership
renounceOwnership
IUniswapV2Factory (iface)
createPair
IUniswapV2Router (iface)
factory
WETH
swapExactTokensForETHSupportingFeeOnTransferTokens
ERC20FLOKINOMICS2
lockTheSwap
<Constructor>
changeWallets
allowance
clearStuckBalance
transfer
includeInReward
goLive
setSwapSettings
manage_excludeFromFee
transferFrom
clearStuckToken
setMultipliers
<Receive Ether>
payable
decreaseAllowance
increaseAllowance
approve
swapTokensForEth
_approve
_transferStandard
set_All_Fees
removeAllFee
restoreAllFee
_setAllFees
_transfer
_takeMarketingFee
excludeFromReward
balanceOf
tokenFromReflection
_getValues
_getRate
_getCurrentSupply
Context
_msgSender
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

Flokinomics 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 will eliminate all the contract threats. The contract allows the owner to set transaction fees as high as 80% at the start, which can later be reduced to a maximum fee of 40%.

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