Cyberscope

Audit Report

CATLY

Jul 2023

Network	BSC
Address	0x1d78d8b0a7c88421A644230d3E54C8799e30A0A0
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
●	FRV	Fee Restoration Vulnerability	Unresolved
●	MMN	Misleading Method Naming	Unresolved
●	PTRP	Potential Transfer Revert Propagation	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	MEE	Missing Events Emission	Unresolved
●	RED	Redundant Event Declaration	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
●	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	CATLY
Compiler Version	v0.8.4+commit.c7e474f2
Optimization	200 runs
Explorer	https://bscscan.com/address/0x1d78d8b0a7c88421a644230d3e54c8799e30a0a0
Address	0x1d78d8b0a7c88421a644230d3e54c8799e30a0a0
Network	BSC
Symbol	CATLY
Decimals	18
Total Supply	2,100,000,000

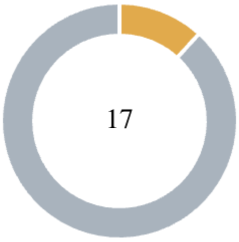
Audit Updates

Initial Audit	01 Jul 2023

Source Files

Filename	SHA256
CATLY.sol	9d2806e518df56f0cf48e3760ba02439265394a046c5872fe10a10aeb0948d69

Findings Breakdown



●	Critical	0
●	Medium	2
●	Minor / Informative	15

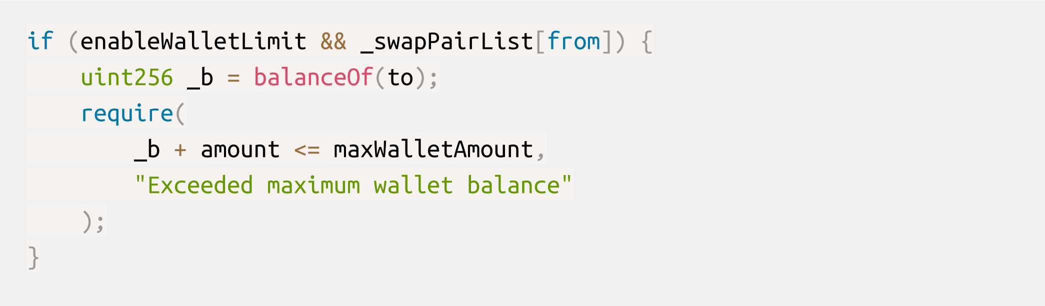
Severity		Unresolved	Acknowledged	Resolved	Other
●	Critical	0	0	0	0
●	Medium	2	0	0	0
●	Minor / Informative	15	0	0	0

ST - Stops Transactions

Criticality	Medium
Location	CATLY.sol#L1061
Status	Unresolved

Description

The contract owner has the authority to stop all users from buying excluding the authorized addresses. The owner may take advantage of it by setting the `maxWalletAmount` to zero.



```
if (enableWalletLimit && _swapPairList[from]) {
    uint256 _b = balanceOf(to);
    require(
        _b + amount <= maxWalletAmount,
        "Exceeded maximum wallet balance"
    );
}
```

Recommendation

The contract could embody a check for not allowing setting the `maxWalletAmount` less than a reasonable amount. A suggested implementation could check that the minimum amount should be more than a fixed percentage of the total supply. 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.

FRV - Fee Restoration Vulnerability

Criticality	Medium
Location	CATLY.sol#L961
Status	Unresolved

Description

The contract demonstrates a potential vulnerability upon removing and restoring the fees. This vulnerability can occur when the fees have been set to zero. During a transaction, if the fees have been set to zero, then both remove fees and restore fees functions will be executed. The remove fees function is executed to temporarily remove the fees, ensuring the sender is not taxed during the transfer. However, the function prematurely returns without setting the variables that hold the previous fee values.

As a result, when the subsequent restore fees function is called after the transfer, it restores the fees to their previous values. However, since the previous fee values were not properly set to zero, there is a risk that the fees will retain their non-zero values from before the fees were removed. This can lead to unintended consequences, potentially causing incorrect fee calculations or unexpected behavior within the contract.

```
function removeAllFee() private {
    if (
        _buyRewardFee == 0 &&
        _LP_MKTBuyFee == 0 &&
        _sellRewardFee == 0 &&
        _LP_MKTSellFee == 0
    ) return;

    _previousBuyTaxFee = _buyRewardFee;
    _previousSellTaxFee = _sellRewardFee;

    _previousBuyLP_MKTFee = _LP_MKTBuyFee;
    _previousSellLP_MKTFee = _LP_MKTSellFee;

    _buyRewardFee = 0;
    _sellRewardFee = 0;

    _LP_MKTBuyFee = 0;
    _LP_MKTSellFee = 0;
}
```

Recommendation

The team is advised to modify the remove fees function to ensure that the previous fee values are correctly set to zero, regardless of their initial values. A recommended approach would be to remove the early return when both fees are zero.

MMN - Misleading Method Naming

Criticality	Minor / Informative
Location	CATLY.sol#L1160
Status	Unresolved

Description

Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand. Methods can have misleading names if their names do not accurately reflect the functionality they contain or the purpose they serve. The contract uses some method names that are too generic or do not clearly convey the underneath functionality. Misleading method names can lead to confusion, making the code more difficult to read and understand.

The `swapTokensForEth` method name indicates that it swaps tokens for ETH. However, its implementation suggests otherwise, meaning it swaps tokens for tokens. As a result, the method name is misleading.

```
function swapTokensForEth(uint256 tokenAmount) private {
    ...
    try
        _swapRouter.swapExactTokensForTokensSupportingFeeOnTransferTokens(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(_tokenDistributor),
            block.timestamp
        )
    {} catch {
        emit failed_swap(0);
    }
    ...
}
```

Recommendation

It's always a good practice for the contract to contain method names that are specific and descriptive. The team is advised to keep in mind the readability of the code.

PTRP - Potential Transfer Revert Propagation

Criticality	Minor / Informative
Location	CATLY.sol#L1129
Status	Unresolved

Description

The contract sends funds to a `fundAddress` 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(fundAddress).transfer(address(this).balance);
```

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	CATLY.sol#L805
Status	Unresolved

Description

The contract accumulates tokens from the taxes to swap them for ETH. The variable `swapTokensAtAmount` 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 setNumTokensSellToAddToLiquidity(uint256 swapNumber)
    public
    onlyOwner
{
    numTokensSellToAddToLiquidity = swapNumber;
}
```

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.

MEE - Missing Events Emission

Criticality	Minor / Informative
Location	CATLY.sol#L590,1007,1011
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.

```
fundAddress = addr;
enableWalletLimit = false;
enableChangeTax = false;
```

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.

RED - Redundant Event Declaration

Criticality	Minor / Informative
Location	CATLY.sol#L498
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 contract declares certain events that are not used in a meaningful way by the contract. As a result, these events are redundant.

```
event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
```

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	CATLY.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	CATLY.sol#L518,519,520,521,522,530,540,562,566,571
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.

```
name
symbol
decimals
_tTotal
currency
kb
currencyIsEth
_swapRouter
_mainPair
_tokenDistributor
```

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	CATLY.sol#L442
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.

```
address deadAddress = 0x000000000000000000000000000000000000dEaD
```

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	CATLY.sol#L279,364,449,470,471,475,476,478,479,481,482,484,485,488,490,491,812,929,937,945,953,1014,1135,1158
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.

```
address public _owner
function WETH() external pure returns (address);
TokenDistributor public _tokenDistributor
uint256 public _buyRewardFee
uint256 public _sellRewardFee
uint256 public _buyLPFee
uint256 public _buyFundFee
uint256 public _sellLPFee
uint256 public _sellFundFee
uint256 public _LP_MKTBuyFee
uint256 public _LP_MKTSellFee
uint256 private _previousBuyLP_MKTFee
uint256 private _previousSellLP_MKTFee
IUniswapV2Router02 public _swapRouter

...
```

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	CATLY.sol#L787,809,1015
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.

```
_buyLPFee = customs[0]
numTokensSellToAddToLiquidity = swapNumber
maxWalletAmount = _amount
```

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	CATLY.sol#L1078,1105
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 LP_Amount = (contractTokenBalance /
            (_buyFundFee + _sellFundFee + _buyLPFee + _sellLPFee)) *
            (_buyLPFee + _sellLPFee)
```

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	CATLY.sol#L1102
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 lpEth
```

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	CATLY.sol#L566,590
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.

```
_mainPair = swapPair
fundAddress = addr
```

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	CATLY.sol#L3
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 ^0.8.4;
```

Recommendation

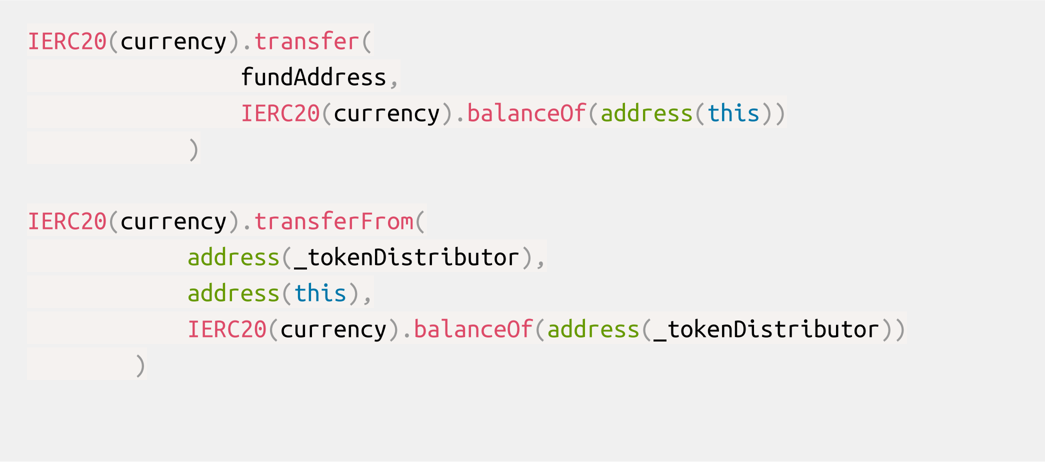
The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest 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	CATLY.sol#L1123,1182
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(currency).transfer(
                fundAddress,
                IERC20(currency).balanceOf(address(this))
            )

IERC20(currency).transferFrom(
            address(_tokenDistributor),
            address(this),
            IERC20(currency).balanceOf(address(_tokenDistributor))
        )
```

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	✓	-
SafeMath	Library			
	add	Internal		
	sub	Internal		
	sub	Internal		
	mul	Internal		
	div	Internal		
	div	Internal		
	mod	Internal		
	mod	Internal		

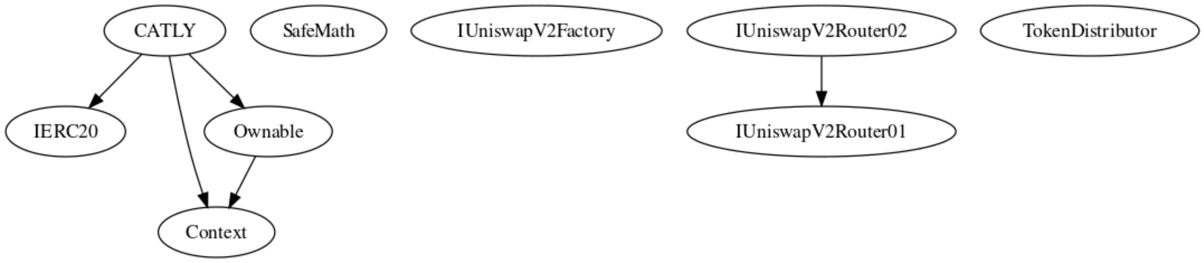
Context	Implementation			
	_msgSender	Internal		
	_msgData	Internal		
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
IUniswapV2Factory	Interface			
	getPair	External		-
	createPair	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-

	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
TokenDistributor	Implementation			
		Public	✓	-
CATLY	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	setFundAddress	External	✓	onlyOwner
	totalSupply	Public		-
	balanceOf	Public		-
	transfer	Public	✓	-
	allowance	Public		-
	approve	Public	✓	-
	transferFrom	Public	✓	-
	increaseAllowance	Public	✓	-
	decreaseAllowance	Public	✓	-
	isExcludedFromReward	Public		-
	totalFees	Public		-
	deliver	Public	✓	-
	reflectionFromToken	Public		-
	tokenFromReflection	Public		-

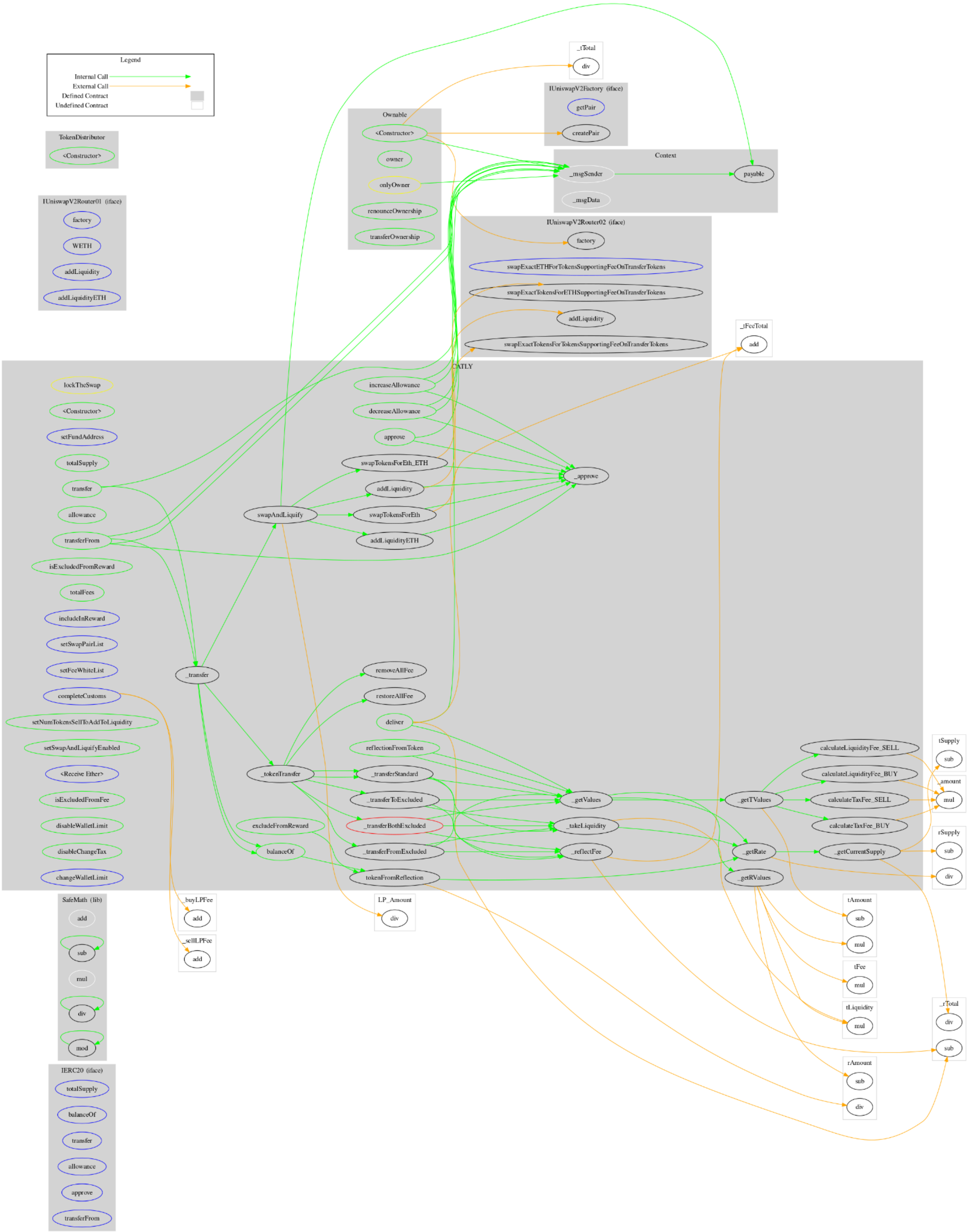
	excludeFromReward	Public	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	_transferBothExcluded	Private	✓	
	setSwapPairList	External	✓	onlyOwner
	setFeeWhiteList	External	✓	onlyOwner
	completeCustoms	External	✓	onlyOwner
	setNumTokensSellToAddToLiquidity	Public	✓	onlyOwner
	setSwapAndLiquifyEnabled	Public	✓	onlyOwner
		External	Payable	-
	_reflectFee	Private	✓	
	_getValues	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeLiquidity	Private	✓	
	calculateTaxFee_BUY	Private		
	calculateTaxFee_SELL	Private		
	calculateLiquidityFee_BUY	Private		
	calculateLiquidityFee_SELL	Private		
	removeAllFee	Private	✓	
	restoreAllFee	Private	✓	
	isExcludedFromFee	Public		-

	_approve	Private	✓	
	disableWalletLimit	Public	✓	onlyOwner
	disableChangeTax	Public	✓	onlyOwner
	changeWalletLimit	External	✓	onlyOwner
	_transfer	Private	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth_ETH	Private	✓	
	swapTokensForEth	Private	✓	
	addLiquidityETH	Private	✓	
	addLiquidity	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	

Inheritance Graph



Flow Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
TokenDistributor
<Constructor>
IUniswapV2Router01 (iface)
factory
WETH
addLiquidity
addLiquidityETH
Ownable
<Constructor>
owner
onlyOwner
renounceOwnership
transferOwnership
_tTotal
div
IUniswapV2Factory (iface)
getPair
createPair
Context
_msgSender
payable
_msgData
IUniswapV2Router02 (iface)
factory
swapExactETHForTokensSupportingFeeOnTransferTokens
swapExactTokensForETHSupportingFeeOnTransferTokens
addLiquidity
swapExactTokensForTokensSupportingFeeOnTransferTokens
_tFeeTotal
add
CATLY
lockTheSwap
<Constructor>
setFundAddress
totalSupply
transfer
allowance
transferFrom
isExcludedFromReward
totalFees
includeInReward
setSwapPairList
setFeeWhiteList
completeCustoms
setNumTokensSellToAddToLiquidity
setSwapAndLiquifyEnabled
<Receive Ether>
isExcludedFromFee
disableWalletLimit
disableChangeTax
changeWalletLimit
increaseAllowance
decreaseAllowance
approve
swapTokensForEth_ETH
addLiquidity
swapAndLiquify
swapTokensForEth
addLiquidityETH
_approve
_transfer
removeAllFee
restoreAllFee
deliver
reflectionFromToken
_tokenTransfer
_transferStandard
_transferToExcluded
excludeFromReward
_transferBothExcluded
balanceOf
_transferFromExcluded
tokenFromReflection
_getValues
_takeLiquidity
_reflectFee
_getTValues
_getRate
_getRValues
calculateLiquidityFee_SELL
calculateLiquidityFee_BUY
calculateTaxFee_SELL
calculateTaxFee_BUY
_getCurrentSupply
tSupply
sub
_amount
mul
rSupply
sub
div
SafeMath (lib)
add
sub
mul
div
mod
_buyLPFee
add
_sellLPFee
add
LP_Amount
div
tAmount
sub
mul
tFee
mul
tLiquidity
mul
rAmount
sub
div
_rTotal
div
sub
IERC20 (iface)
totalSupply
balanceOf
transfer
allowance
approve
transferFrom

Summary

CATLY 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 stopping 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 25% 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