Cyberscope

Audit Report

Catch

January 2024

Repository	https://github.com/EtherAuthority/Smart-Contracts-Library/tree/main
Commit	f52821a932216176f00351e08e679655399dc99d
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
●	IIE	Immutable Initialization Error	Unresolved
●	AOI	Arithmetic Operations Inconsistency	Unresolved
●	MEE	Misleading Event Emission	Unresolved
●	PVC	Price Volatility Concern	Unresolved
●	RBC	Redundant Balance Check	Unresolved
●	RRS	Redundant Require Statement	Unresolved
●	RSML	Redundant SafeMath Library	Unresolved
●	RSW	Redundant Storage Writes	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L07	Missing Events Arithmetic	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L16	Validate Variable Setters	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved

Table of Contents

Review

Repository	https://github.com/EtherAuthority/Smart-Contracts-Library/tree/main
Commit	f52821a932216176f00351e08e679655399dc99d
Badge Eligibility	Must Fix Criticals

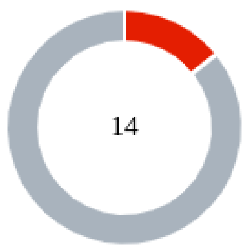
Audit Updates

Initial Audit	15 Jan 2024

Source Files

Filename	SHA256
catch.sol	979ef145504ada6cf28983a790ea1e282320a047ea65c4c05175f547d703056c

Findings Breakdown



🟥	Critical	2
🟧	Medium	0
⬜	Minor / Informative	12

Severity		Unresolved	Acknowledged	Resolved	Other
🟥	Critical	2	0	0	0
🟧	Medium	0	0	0	0
⬜	Minor / Informative	12	0	0	0

ST - Stops Transactions

Criticality	Critical
Location	catch.sol#L1398
Status	Unresolved

Description

The contract owner has the authority to stop the sales for all users excluding the owner. The owner may take advantage of it by setting the `_maxTxAmount` to zero and as described in detail in section `PVC`. As a result, the contract may operate as a honeypot.

```
if(from != owner() && to != owner())
    require(amount <= _maxTxAmount, "Transfer amount exceeds
the maxTxAmount.");
```

Recommendation

The team is strongly encouraged to adhere to the recommendations outlined in the respective sections. Additionally, the contract could embody a check for not allowing setting the `_maxTxAmount` 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.

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.

IIE - Immutable Initialization Error

Criticality	Critical
Location	catch.sol#L711
Status	Unresolved

Description

The issue arises in networks that do not support Pragma Solidity above version `8.19.0`, such as the Binance Smart Chain (BSC) network. The problem lies in the contract's attempt to initialize the immutable variable `uniswapV2Router` within an if statement, which is not allowed in Solidity. Immutable variables must receive their initial value at the time of declaration or within the contract's constructor and cannot be changed afterward. The code contains multiple instances where `uniswapV2Router` is assigned different values based on different conditions, leading to a `TypeError`. Immutable variables cannot be modified or initialized within an if statement or any other function after the contract has been deployed.

```
    constructor (address _fundWallet)  {
     ...
        if (block.chainid == 56) {
            uniswapV2Router =
IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E); //BNB
Smart Chain Mainnet
        } else if (block.chainid == 97) {
            uniswapV2Router =
IUniswapV2Router02(0xD99D1c33F9fC3444f8101754aBC46c52416550D1); //BNB
Smart Chain testnet
        } else if (block.chainid == 1 || block.chainid == 4 ||
block.chainid == 3) {
            uniswapV2Router =
IUniswapV2Router02(0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D);
//Ethereum Mainnet
        } else if (block.chainid == 43114) {
            uniswapV2Router =
IUniswapV2Router02(0x60aE616a2155Ee3d9A68541Ba4544862310933d4); //
Avalanche C-Chain
        } else if (block.chainid == 250) {
            uniswapV2Router =
IUniswapV2Router02(0xF491e7B69E4244ad4002BC14e878a34207E38c29); //
Fantom Opera
        } else {
            revert("Chain not valid");
        }

}
```

```
TypeError: Cannot write to immutable here: Immutable variables cannot be
initialized inside an if statement.
```

Recommendation

It is recommended to refactor the contract such that the `uniswapV2Router` is assigned a value once and only in the constructor or at its declaration. If different router addresses are required based on conditions, consider using a different design pattern, such as a function to set the router address or using a mapping to hold different router addresses. This change will ensure compliance with Solidity's constraints on immutable variables and improve the contract's overall design and reliability.

AOI - Arithmetic Operations Inconsistency

Criticality	Minor / Informative
Location	catch.sol#L1194,1122,1165
Status	Unresolved

Description

The contract uses both the SafeMath library and native arithmetic operations. The SafeMath library is commonly used to mitigate vulnerabilities related to integer overflow and underflow issues. However, it was observed that the contract also employs native arithmetic operators (such as +, -, *, /) in certain sections of the code.

The combination of SafeMath library and native arithmetic operations can introduce inconsistencies and undermine the intended safety measures. This discrepancy creates an inconsistency in the contract's arithmetic operations, increasing the risk of unintended consequences such as inconsistency in error handling, or unexpected behavior.

```
   uint256 allTax = tFee+tLiquidity+tCoinOperation+tBurn;

    uint256 allTax = rFee+rCoinOperation+rBurn+rLiquidity;

    _tOwned[recipient] =
_tOwned[recipient].add(tTransferAmount);
```

Recommendation

To address this finding and ensure consistency in arithmetic operations, it is recommended to standardize the usage of arithmetic operations throughout the contract. The contract should be modified to either exclusively use SafeMath library functions or entirely rely on native arithmetic operations, depending on the specific requirements and design considerations. This consistency will help maintain the contract's integrity and mitigate potential vulnerabilities arising from inconsistent arithmetic operations.

MEE - Misleading Event Emission

Criticality	Minor / Informative
Location	catch.sol#L449,472
Status	Unresolved

Description

The contract is currently emitting the `OwnershipTransferred(_owner, address(0));` event during the `lock` function execution. However, this event suggestsa transfer of ownership to the zero address, which is not accurately represent the intent of this action. This event is emitted to signify the locking of the contract, not a transfer to the zero address, and as a result the `OwnershipTransferred` event within the `lock` function becomes misleading.

```
   function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }


    function lock(uint256 time) public virtual onlyOwner {
        _previousOwner = _owner;
        _owner = address(0);
        _lockTime = block.timestamp + time;
        emit OwnershipTransferred(_owner, address(0));
    }
```

Recommendation

It is recommended to review the usage of the `OwnershipTransferred` event within the `lock` function. To prevent any misunderstanding, consider emitting a different event specifically for `locking` the contract, or provide a comment in the code to clarify the purpose of this event emission during the lock function execution. This will help improve the code's readability and reduce potential misconceptions.

PVC - Price Volatility Concern

Criticality	Minor / Informative
Location	catch.sol#L1414
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.

```
       bool overMinTokenBalance = contractTokenBalance >=
numTokensSellToAddToLiquidity;
        if (
            overMinTokenBalance &&
            !inSwapAndLiquify &&
            from != uniswapV2Pair &&
            swapAndLiquifyEnabled
        ) {
            contractTokenBalance =
numTokensSellToAddToLiquidity;
            //add liquidity
            swapAndLiquify(contractTokenBalance);
        }
```

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

RBC - Redundant Balance Check

Criticality	Minor / Informative
Location	catch.sol#L1400
Status	Unresolved

Description

The contract is setting the `contractTokenBalance` to `_maxTxAmount` within an if-check, even though there is already a subsequent if-check that determines whether `contractTokenBalance` is greater than or equal to `numTokensSellToAddToLiquidity`. In this segment, the initial check and assignment of `contractTokenBalance` to `_maxTxAmount` is redundant since the `contractTokenBalance` is later evaluated against `numTokensSellToAddToLiquidity`, and the essential logic is to determine whether the balance is sufficient for adding to liquidity and the amount of tokens that is needed fot the swap.

```
       uint256 contractTokenBalance = balanceOf(address(this));

        if(contractTokenBalance >= _maxTxAmount)
        {
            contractTokenBalance = _maxTxAmount;
        }

        bool overMinTokenBalance = contractTokenBalance >=
numTokensSellToAddToLiquidity;
        if (
            overMinTokenBalance &&
            !inSwapAndLiquify &&
            from != uniswapV2Pair &&
            swapAndLiquifyEnabled
        ) {
            contractTokenBalance =
numTokensSellToAddToLiquidity;
        ...
    }
```

Recommendation

t is recommended to remove the initial if-check that sets `contractTokenBalance` to `_maxTxAmount`. This simplification will eliminate the redundant code and improve the contract's clarity and efficiency. The key logic should solely rely on whether `contractTokenBalance` is greater than or equal to `numTokensSellToAddToLiquidity` for determining actions related to liquidity addition. This change will streamline the contract logic without affecting its intended functionality.

RRS - Redundant Require Statement

Criticality	Minor / Informative
Location	catch.sol#L112
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	catch.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.

RSW - Redundant Storage Writes

Criticality	Minor / Informative
Location	catch.sol#L1016,1026
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.

```
_isExcludedFromFee[account] = true;
_isExcludedFromFee[account] = false;
```

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.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	catch.sol#L512,650,651,663,664,665,673,686,687,1040,1066,1082,1279,1292,1306,1321
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 _rOwned;
    mapping (address => uint256) public _tOwned;
    string  private constant _name = "catchcoin";
    string  private  constant _symbol = "CATCH";
    uint8  private constant _decimals = 18;
    uint256 public _maxTxAmount = 1 * 10**6 * 10**18;
    event fundWalletChange(address wallet);
    event thresholdUpdated(uint256 amount);
    ...
```

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	catch.sol#L1055,1106,1130,1263
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.

```
   function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
        _maxTxAmount = _tTotal.mul(maxTxPercent).div(
            10**2
        );
    }
    function _reflectFee(uint256 rFee, uint256 tFee, uint256 tBurn)
private {
        uint256 currentRate = _getRate();
        uint256 rBurn = tBurn.mul(currentRate);

        _rTotal = _rTotal.sub(rFee).sub(rBurn);
        _tFeeTotal = _tFeeTotal.add(tFee);

        _tTotal = _tTotal.sub(tBurn);

    }

    function _takeCoinFund(uint256 tCoinOperation) private {
          ...
    }
    function _takeLiquidity(uint256 tLiquidity) private {
          ...
    }
```

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.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	catch.sol#L281,308,334,344,359,369,374
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 sendValue(address payable recipient, uint256
amount) internal {
        require(address(this).balance >= amount, "Address:
insufficient balance");

        // solhint-disable-next-line avoid-low-level-calls,
avoid-call-value
        (bool success, ) = recipient.call{ value: amount }("");
        require(success, "Address: unable to send value,
recipient may have reverted");
    }

    function functionCall(address target, bytes memory data)
internal returns (bytes memory) {
      return functionCall(target, data, "Address: low-level
call failed");
    }
...
```

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.

L16 - Validate Variable Setters

Criticality	Minor / Informative
Location	catch.sol#L708,1041
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.

```
fundWallet = _fundWallet;
```

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.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	catch.sol#L288,387
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly { codehash := extcodehash(account) }

assembly {
    let returndata_size := mload(returndata)
    revert(add(32, returndata), returndata_size)
}
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

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		
Address	Library			
	isContract	Internal		
	sendValue	Internal	✓	
	functionCall	Internal	✓	
	functionCall	Internal	✓	
	functionCallWithValue	Internal	✓	
	functionCallWithValue	Internal	✓	
	_functionCallWithValue	Private	✓	
Ownable	Implementation	Context		
		Public	✓	-
	owner	Public		-
	renounceOwnership	Public	✓	onlyOwner
	transferOwnership	Public	✓	onlyOwner
	geUnlockTime	Public		-
	lock	Public	✓	onlyOwner
	unlock	Public	✓	-
IUniswapV2Factory	Interface			
	feeTo	External		-

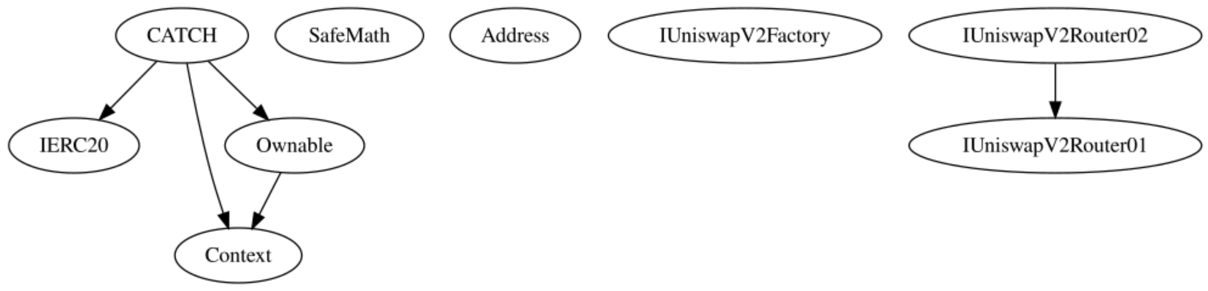
	feeToSetter	External		-
	getPair	External		-
	allPairs	External		-
	allPairsLength	External		-
	createPair	External	✓	-
	setFeeTo	External	✓	-
	setFeeToSetter	External	✓	-
IUniswapV2Router01	Interface			
	factory	External		-
	WETH	External		-
	addLiquidity	External	✓	-
	addLiquidityETH	External	Payable	-
	removeLiquidity	External	✓	-
	removeLiquidityETH	External	✓	-
	removeLiquidityWithPermit	External	✓	-
	removeLiquidityETHWithPermit	External	✓	-
	swapExactTokensForTokens	External	✓	-
	swapTokensForExactTokens	External	✓	-
	swapExactETHForTokens	External	Payable	-
	swapTokensForExactETH	External	✓	-
	swapExactTokensForETH	External	✓	-
	swapETHForExactTokens	External	Payable	-

	quote	External		-
	getAmountOut	External		-
	getAmountIn	External		-
	getAmountsOut	External		-
	getAmountsIn	External		-
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External	✓	-
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External	✓	-
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External	✓	-
	swapExactETHForTokensSupportingFeeOnTransferTokens	External	Payable	-
	swapExactTokensForETHSupportingFeeOnTransferTokens	External	✓	-
CATCH	Implementation	Context, IERC20, Ownable		
		Public	✓	-
	name	External		-
	symbol	External		-
	decimals	External		-
	totalSupply	External		-
	balanceOf	Public		-
	transfer	External	✓	-

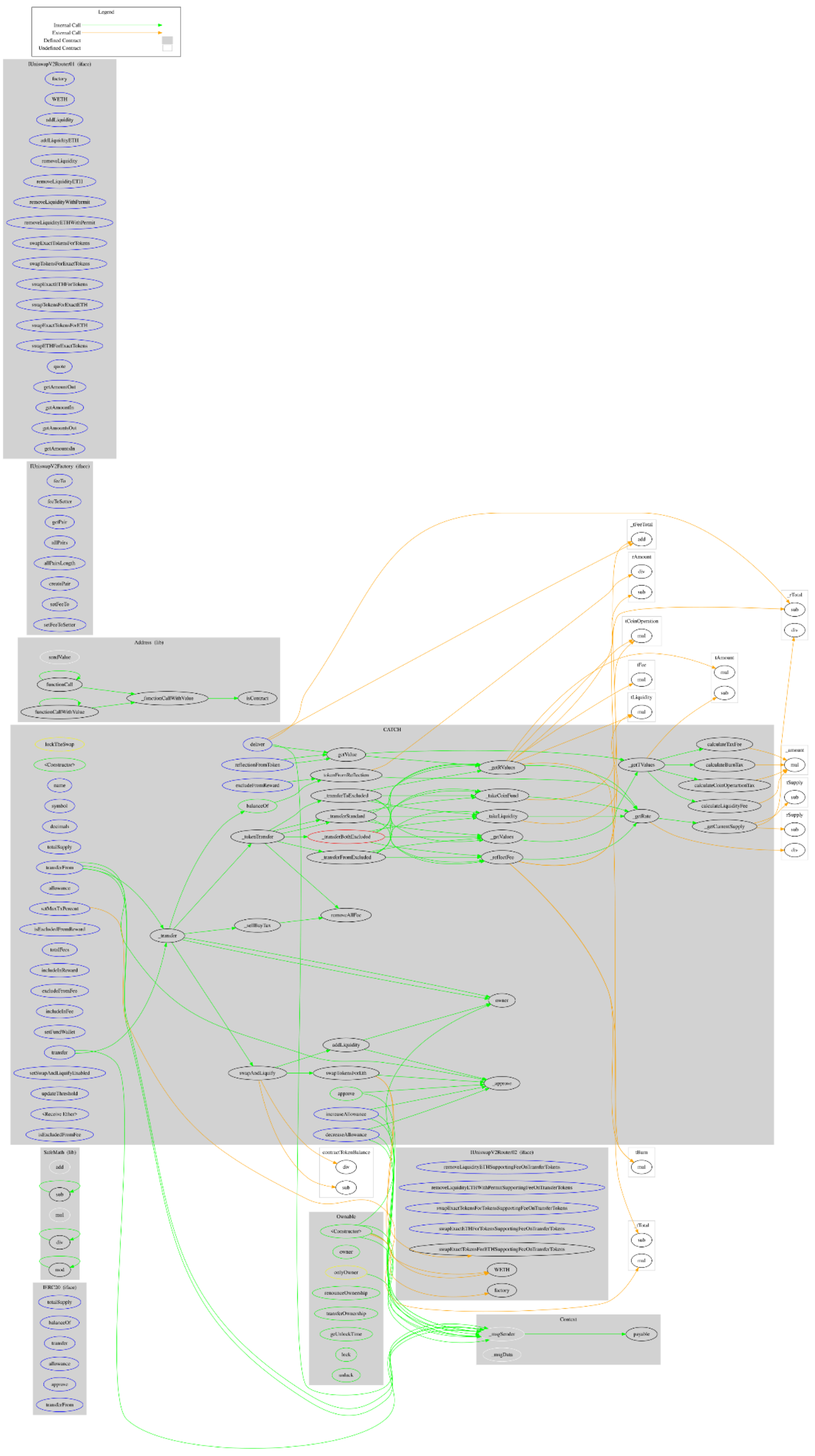
	allowance	External		-
	approve	Public	✓	-
	transferFrom	External	✓	-
	increaseAllowance	External	✓	-
	decreaseAllowance	External	✓	-
	isExcludedFromReward	External		-
	totalFees	External		-
	deliver	External	✓	-
	reflectionFromToken	External		-
	tokenFromReflection	Public		-
	excludeFromReward	External	✓	onlyOwner
	includeInReward	External	✓	onlyOwner
	_transferBothExcluded	Private	✓	
	excludeFromFee	External	✓	onlyOwner
	includeInFee	External	✓	onlyOwner
	setFundWallet	External	✓	onlyOwner
	setMaxTxPercent	External	✓	onlyOwner
	setSwapAndLiquifyEnabled	External	✓	onlyOwner
	updateThreshold	External	✓	onlyOwner
		External	Payable	-
	_reflectFee	Private	✓	
	_takeCoinFund	Private	✓	
	_getValues	Private		

	_getValue	Private		
	_getTValues	Private		
	_getRValues	Private		
	_getRate	Private		
	_getCurrentSupply	Private		
	_takeLiquidity	Private	✓	
	calculateTaxFee	Private		
	calculateLiquidityFee	Private		
	calculateCoinOperartionTax	Private		
	calculateBurnTax	Private		
	removeAllFee	Private	✓	
	isExcludedFromFee	External		-
	_approve	Private	✓	
	_transfer	Private	✓	
	_sellBuyTax	Private	✓	
	swapAndLiquify	Private	✓	lockTheSwap
	swapTokensForEth	Private	✓	
	addLiquidity	Private	✓	
	_tokenTransfer	Private	✓	
	_transferStandard	Private	✓	
	_transferToExcluded	Private	✓	
	_transferFromExcluded	Private	✓	

Inheritance Graph



Flow Graph



Summary

Catch 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. 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 a 3% fee on buy transactions and a 6 % fee on 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