Ether Authority

www.EtherAuthority.io
audit@etherauthority.io

SMART CONTRACT

Security Audit Report

Project: Catcoin Token
Website: https://catcoin.io
Platform: Ethereum
Language: Solidity
Date: February 22nd, 2023

Table of contents

Email: audit@EtherAuthority.io

`

THIS IS SECURITY AUDIT REPORT DOCUMENT AND WHICH MAY CONTAIN INFORMATION WHICH IS CONFIDENTIAL. WHICH INCLUDES ANY POTENTIAL VULNERABILITIES AND MALICIOUS CODES WHICH CAN BE USED TO EXPLOIT THE SOFTWARE. THIS MUST BE REFERRED INTERNALLY AND ONLY SHOULD BE MADE AVAILABLE TO THE PUBLIC AFTER ISSUES ARE RESOLVED.

Email: audit@EtherAuthority.io

Introduction

EtherAuthority was contracted by the Catcoin team to perform the Security audit of the Catcoin smart contract code. The audit has been performed using manual analysis as well as using automated software tools. This report presents all the findings regarding the audit performed on February 22nd, 2023.

The purpose of this audit was to address the following:

- Ensure that all claimed functions exist and function correctly.
- Identify any security vulnerabilities that may be present in the smart contract.

Project Background

- Catcoin Contract is an ERC20 reflection token. This Contract uses uniswap.
- Catcoin Contracts have functions like swapAndLiquify, removeStuckToken, addLiquidity, withdrawStuckETH, receive, etc.

Audit scope

Name	**Code Review and Security Analysis Report for Catcoin Token Smart Contract**
Platform	**Ethereum / Solidity**
File	Catcoin.sol
Online Code	0x06f06745fd679070aeb82d1675b09a556d0177e4
Audit Date	February 22nd, 2023
Revised Code	0xe62d0999bddce44cdc3e97f3b8ef646a7e29567a
Revised Audit Date	February 23rd, 2023

Claimed Smart Contract Features

Claimed Feature Detail	Our Observation
Tokenomics: • Name: Catcoin • Symbol: CATS • Decimals: 0 • Total Supply: 9 Trillion	**YES, This is valid.**
Trading: • Number of tokens to be sold to add liquidity: 2 Billion • Buy Liquidity Fee: 7% • Sell Tax Fee: 2% • Sell Liquidity Fee: 7%	**YES, This is valid.**
Ownership Control: • The owner can set a buy and sell fee percentage. • The owner can set Numbers of token sales to add to liquidity. • The owner can update the router address. • The owner can update swap and liquify enabled status. • The owner can withdraw the stuck ether token. • The owner can remove the stuck token.	**YES, This is valid.**

Audit Summary

According to the standard audit assessment, Customer`s solidity based smart contracts are **“Secured”**. This token contract does contain owner control, which does not make it fully decentralized.

We used various tools like Slither, Solhint and Remix IDE. At the same time this finding is based on critical analysis of the manual audit.
All issues found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the Audit overview section. General overview is presented in AS-IS section and all identified issues can be found in the Audit overview section.

We found 0 critical, 2 high, 0 medium and 3 low and some very low level issues. These issues are fixed/acknowledged in the revised contract code.

Investors Advice: Technical audit of the smart contract does not guarantee the ethical nature of the project. Any owner controlled functions should be executed by the owner with responsibility. All investors/users are advised to do their due diligence before investing in the project.

Technical Quick Stats

Main Category	Subcategory	Result
Contract Programming	Solidity version not specified	Passed
	Solidity version too old	Passed
	Integer overflow/underflow	Passed
	Function input parameters lack of check	Passed
	Function input parameters check bypass	Passed
	Function access control lacks management	Passed
	Critical operation lacks event log	Passed
	Human/contract checks bypass	Passed
	Random number generation/use vulnerability	N/A
	Fallback function misuse	Passed
	Race condition	Passed
	Logical vulnerability	Passed
	Features claimed	Passed
	Other programming issues	Moderated
Code Specification	Function visibility not explicitly declared	Passed
	Var. storage location not explicitly declared	Passed
	Use keywords/functions to be deprecated	Passed
	Unused code	Passed
Gas Optimization	“Out of Gas” Issue	Passed
	High consumption ‘for/while’ loop	Passed
	High consumption ‘storage’ storage	Passed
	Assert() misuse	Passed
Business Risk	The maximum limit for mintage not set	Passed
	“Short Address” Attack	Passed
	“Double Spend” Attack	Passed

Overall Audit Result: **PASSED**

Code Quality

This audit scope has 1 smart contract. Smart contract contains Libraries, Smart contracts, inherits and Interfaces. This is a compact and well written smart contract.

The libraries in Catcoin Token are part of its logical algorithm. A library is a different type of smart contract that contains reusable code. Once deployed on the blockchain (only once), it is assigned a specific address and its properties / methods can be reused many times by other contracts in the Catcoin Token.

The Catcoin Token team has **not** provided scenario and unit test scripts, which would have helped to determine the integrity of the code in an automated way.

Code parts are **not well** commented on in the smart contracts. Ethereum's NatSpec commenting style is recommended.

Documentation

We were given a Catcoin Token smart contract code in the form of a file. The hash of that code is mentioned above in the table.

As mentioned above, code parts are **not well** commented. But the logic is straightforward. So it is easy to quickly understand the programming flow as well as complex code logic. Comments are very helpful in understanding the overall architecture of the protocol.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure that are based on well known industry standard open source projects.

Apart from libraries, its functions are used in external smart contract calls.

AS-IS overview

Functions

Sl.	Functions	Type	Observation	Conclusion
1	constructor	write	Decimal is set to 0, Programming issue	Refer Audit Findings
2	owner	read	Passed	No Issue
3	onlyOwner	modifier	Passed	No Issue
4	renounceOwnership	write	access only Owner	No Issue
5	transferOwnership	write	access only Owner	No Issue
6	lockTheSwap	modifier	Passed	No Issue
7	name	read	Passed	No Issue
8	symbol	read	Passed	No Issue
9	decimals	read	Passed	No Issue
10	totalSupply	read	Passed	No Issue
11	balanceOf	read	Passed	No Issue
12	transfer	write	Passed	No Issue
13	allowance	read	Passed	No Issue
14	approve	write	Passed	No Issue
15	transferFrom	write	Passed	No Issue
16	increaseAllowance	write	Passed	No Issue
17	decreaseAllowance	write	Passed	No Issue
18	isExcludedFromReward	read	Passed	No Issue
19	totalFees	read	Passed	No Issue
20	deliver	write	Passed	No Issue
21	reflectionFromToken	read	Passed	No Issue
22	tokenFromReflection	read	Passed	No Issue
23	excludeFromReward	write	access only Owner	No Issue
24	includeInReward	external	access only Owner	No Issue
25	_transferBothExcluded	write	Passed	No Issue
26	excludeFromFee	write	access only Owner	No Issue
27	includeInFee	write	access only Owner	No Issue
28	setSellFeePercent	external	access only Owner	No Issue
29	setBuyFeePercent	external	access only Owner	No Issue
30	setNumTokensSellToAddToLiquidity	external	access only Owner	No Issue
31	setRouterAddress	external	Update Router address	Refer Audit Findings
32	setSwapAndLiquifyEnabled	external	access only Owner	No Issue
33	receive	external	Passed	No Issue
34	_reflectFee	write	Passed	No Issue
35	_getValues	read	Passed	No Issue
36	_getTValues	read	Passed	No Issue
37	_getRValues	write	Passed	No Issue
38	_getRate	read	Passed	No Issue
39	_getCurrentSupply	read	Passed	No Issue

40	_takeLiquidity	write	Passed	No Issue
41	calculateTaxFee	read	Passed	No Issue
42	calculateLiquidityFee	read	Passed	No Issue
43	removeAllFee	write	Passed	No Issue
44	restoreAllFee	write	Passed	No Issue
45	isExcludedFromFee	read	Passed	No Issue
46	_approve	write	Passed	No Issue
47	_transfer	write	Passed	No Issue
48	swapAndLiquify	write	Passed	No Issue
49	swapTokensForEth	write	Passed	No Issue
50	addLiquidity	write	Programming issue	Refer Audit Findings
51	_tokenTransfer	write	Passed	No Issue
52	_transferStandard	write	Passed	No Issue
53	_transferToExcluded	write	Passed	No Issue
54	_transferFromExcluded	write	Passed	No Issue
55	withdrawStuckETH	external	access only Owner	No Issue
56	removeStuckToken	external	access only Owner	No Issue

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to token loss etc.
High	High-level vulnerabilities are difficult to exploit; however, they also have significant impact on smart contract execution, e.g. public access to crucial
Medium	Medium-level vulnerabilities are important to fix; however, they can't lead to tokens lose
Low	Low-level vulnerabilities are mostly related to outdated, unused etc. code snippets, that can't have significant impact on execution
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations and info statements can't affect smart contract execution and can be ignored.

Email: audit@EtherAuthority.io

Audit Findings

Critical Severity

No Critical severity vulnerabilities were found.

High Severity

(1) Programming issue:

```
    // Create a uniswap pair for this new token
    uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory())
        .createPair(address(this), _uniswapV2Router.WETH());

    // set the rest of the contract variables
    uniswapV2Router = _uniswapV2Router;

    //exclude owner and this contract from fee
    _isExcludedFromFee[_msgSender()] = true;
    _isExcludedFromFee[address(this)] = true;

    //exclude from rewards
    _isExcluded[_burnAddress] = true;
    _isExcluded[uniswapV2Pair] = true;
    _isExcluded[_msgSender()] = true;
    _isExcluded[address(this)] = true;

    _owner = _msgSender();
    emit Transfer(address(0), _msgSender(), _tTotal);
}
```

In the constructor , all addresses excluded from rewards should use excludeFromReward function and not _isExcluded. Because _isExcluded will not push this address to an array which eventually will not include the address in Reward.

Resolution: We suggest using the excludeFromReward function to exclude addresses from rewards.

Status: This issue is fixed to use excludeFromReward functions to exclude addresses from rewards.

```
    //exclude owner and this contract from fee
    _isExcludedFromFee[_msgSender()] = true;
    _isExcludedFromFee[address(this)] = true;

    //exclude from rewards
    _isExcluded[_burnAddress] = true;
    _isExcluded[uniswapV2Pair] = true;
    _isExcluded[_msgSender()] = true;
    _isExcluded[address(this)] = true;

    _owner = _msgSender();
    emit Transfer(address(0), _msgSender(), _tTotal);
}
```

In the constructor, maxSupply is not assigned to tOwned[owner].

Resolution: We suggest assigning maxsupply to the owner.

Status: This issue is fixed with transfer tTotal to owner() in the constructor.

(2) No mint and Initial Supply:
There is no mint function in contract , Also initial supply is not set for the owner.So There is no way to get token supply.

Resolution: We suggest adding initial supply for the owner or provide a mint function.

Status: This issue is fixed in the revised code.

Medium

No Medium severity vulnerabilities were found.

Low

(1) Decimal is set to 0:

```
constructor() {
    _name = "Catcoin";
    _symbol = "CATS";
    _decimals = 0;
```

Decimal value is set to 0 , there is no provision to change the decimal value. This will not have any fractional amount of the tokens. Only whole numbers will be there and no fractional value. For example: there will not be any token 1.5 or 100.35 but it will always be a whole number like 1,100, etc.

Resolution: We suggest setting appropriate values to the decimal variable.

Status: This issue is acknowledged as a desired feature.

(2) Update Router address:
The owner can update the router that generates liquidity to an address or contract of choice (including the zero address). This contract could be a malicious contract that simply keeps the tokens sent to it and thus siphons all the fees. Additionally, this contract could be used to revert sell transactions turning the token into a honeypot.

Resolution: We suggest removing the “setRouterAddress” function.

Status: This issue is fixed by removing the function.

(3) Programming issue:

```
function addLiquidity(uint256 tokenAmount, uint256 ethAmount) private {
    _approve(address(this), address(uniswapV2Router), tokenAmount);
    uniswapV2Router.addLiquidityETH{value: ethAmount}(
        address(this),
        tokenAmount,
        0, // slippage is unavoidable
        0, // slippage is unavoidable
        address(0xdead),
        block.timestamp
    );
}
```

In addLiquidity private function, liquidity is added to dead addresses which will not be recovered.

Resolution: We suggest replacing dead addresses with addresses(this).

Status: This issue is fixed by replacing the dead address.

Very Low / Informational / Best practices:

No Informational severity vulnerabilities were found.

Centralization

This smart contract has some functions which can be executed by the Admin (Owner) only. If the admin wallet private key would be compromised, then it would create trouble. Following are Admin functions:

- excludeFromReward: The owner can exclude from reward.
- includeInReward: The owner can include in reward.
- excludeFromFee: The owner can exclude fees from the account.
- includeInFee: The owner can include a fee from the account.
- setSellFeePercent: The owner can set a sell fee percentage.
- setBuyFeePercent: The owner can set a buy fee percentage.
- setNumTokensSellToAddToLiquidity: The owner can set Numbers of token sells to add to liquidity.
- setRouterAddress: The owner can update the router address.
- setSwapAndLiquifyEnabled: The owner can update swap and liquify enabled status.
- withdrawStuckETH: The owner can withdraw the stuck ether token.
- removeStuckToken: The owner can remove the stuck token.

To make the smart contract 100% decentralized, we suggest renouncing ownership in the smart contract once its function is completed.

Conclusion

We were given a contract code. And we have used all possible tests based on given objects as files. We have observed 2 high severity issues, 3 low severity issues. And these issues are fixed / acknowledged in the revised code. So, **it's good to go for the production**.

Since possible test cases can be unlimited for such smart contracts protocol, we provide no such guarantee of future outcomes. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. Smart Contract's high-level description of functionality was presented in the As-is overview section of the report.

Audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security state of the reviewed smart contract, based on standard audit procedure scope, is **"Secured".**

Email: audit@EtherAuthority.io

Our Methodology

We like to work with a transparent process and make our reviews a collaborative effort. The goals of our security audits are to improve the quality of systems we review and aim for sufficient remediation to help protect users. The following is the methodology we use in our security audit process.

Manual Code Review:

In manually reviewing all of the code, we look for any potential issues with code logic, error handling, protocol and header parsing, cryptographic errors, and random number generators. We also watch for areas where more defensive programming could reduce the risk of future mistakes and speed up future audits. Although our primary focus is on the in-scope code, we examine dependency code and behavior when it is relevant to a particular line of investigation.

Vulnerability Analysis:

Our audit techniques included manual code analysis, user interface interaction, and whitebox penetration testing. We look at the project's web site to get a high level understanding of what functionality the software under review provides. We then meet with the developers to gain an appreciation of their vision of the software. We install and use the relevant software, exploring the user interactions and roles. While we do this, we brainstorm threat models and attack surfaces. We read design documentation, review other audit results, search for similar projects, examine source code dependencies, skim open issue tickets, and generally investigate details other than the implementation.

Documenting Results:

We follow a conservative, transparent process for analyzing potential security vulnerabilities and seeing them through successful remediation. Whenever a potential issue is discovered, we immediately create an Issue entry for it in this document, even though we have not yet verified the feasibility and impact of the issue. This process is conservative because we document our suspicions early even if they are later shown to not represent exploitable vulnerabilities. We generally follow a process of first documenting the suspicion with unresolved questions, then confirming the issue through code analysis, live experimentation, or automated tests. Code analysis is the most tentative, and we strive to provide test code, log captures, or screenshots demonstrating our confirmation. After this we analyze the feasibility of an attack in a live system.

Suggested Solutions:

We search for immediate mitigations that live deployments can take, and finally we suggest the requirements for remediation engineering for future releases. The mitigation and remediation recommendations should be scrutinized by the developers and deployment engineers, and successful mitigation and remediation is an ongoing collaborative process after we deliver our report, and before the details are made public.

Disclaimers

EtherAuthority.io Disclaimer

EtherAuthority team has analyzed this smart contract in accordance with the best industry practices at the date of this report, in relation to: cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report, (Source Code); the Source Code compilation, deployment and functionality (performing the intended functions).

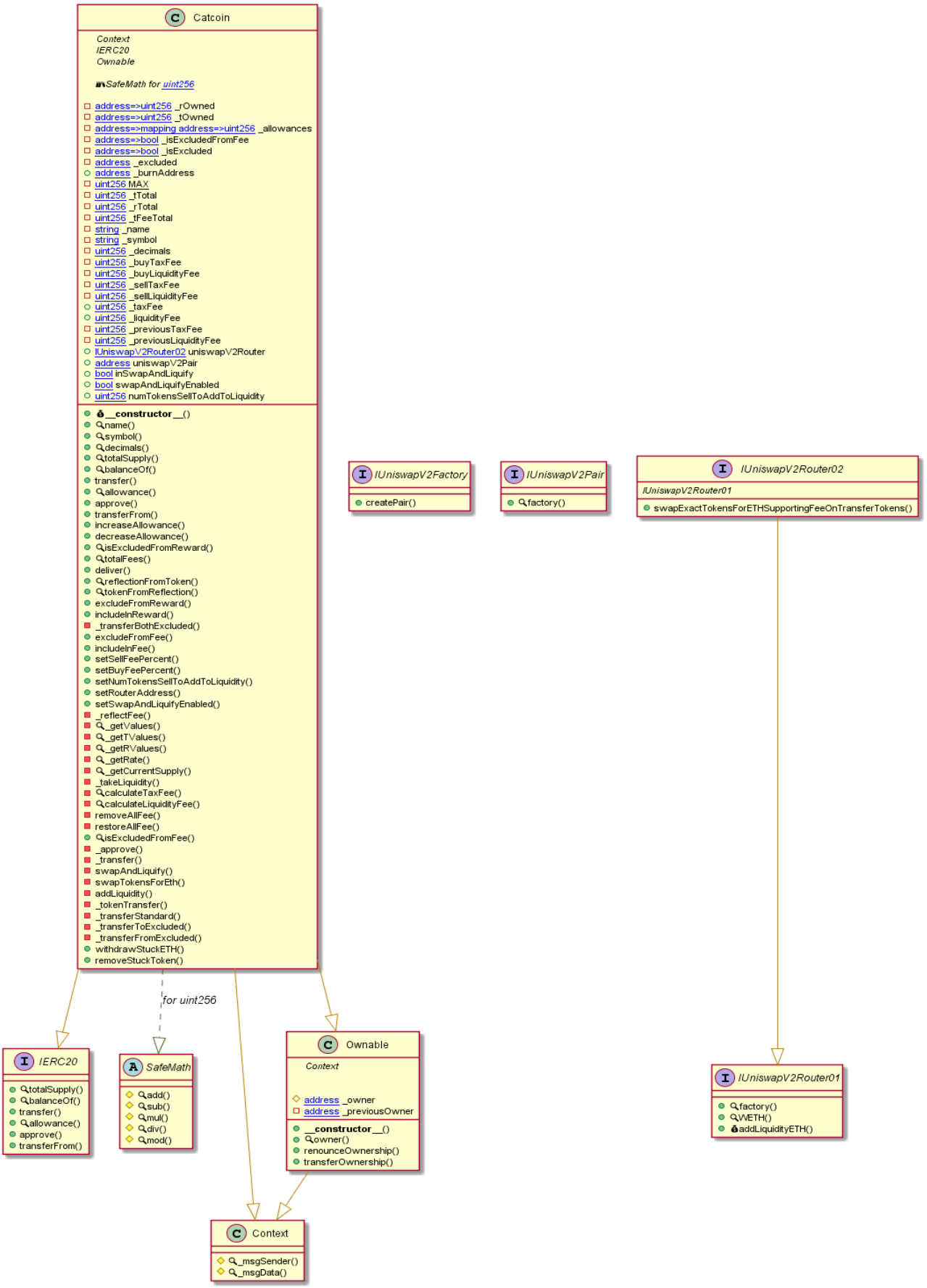
Due to the fact that the total number of test cases are unlimited, the audit makes no statements or warranties on security of the code. It also cannot be considered as a sufficient assessment regarding the utility and safety of the code, bugfree status or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only. We also suggest conducting a bug bounty program to confirm the high level of security of this smart contract.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have their own vulnerabilities that can lead to hacks. Thus, the audit can't guarantee explicit security of the audited smart contracts.

Appendix

Code Flow Diagram - CatcoinToken



Slither Results Log

Slither Log >> Catcoin.sol

```
Catcoin.allowance(address,address).owner (Catecoin.sol#292) shadows:
        - Ownable.owner() (Catecoin.sol#105-107) (function)
Catcoin._approve(address,address,uint256).owner (Catecoin.sol#644) shadows:
        - Ownable.owner() (Catecoin.sol#105-107) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing

Catcoin.setSellFeePercent(uint256,uint256) (Catecoin.sol#456-467) should emit an event for:
        - _sellTaxFee = tFee (Catecoin.sol#460)
        - _sellLiquidityFee = lFee (Catecoin.sol#461)
        - _taxFee = _sellTaxFee (Catecoin.sol#462)
        - _liquidityFee = _sellLiquidityFee (Catecoin.sol#463)
Catcoin.setBuyFeePercent(uint256,uint256) (Catecoin.sol#469-480) should emit an event for:
        - _buyTaxFee = tFee (Catecoin.sol#473)
        - _buyLiquidityFee = lFee (Catecoin.sol#474)
        - _taxFee = _buyTaxFee (Catecoin.sol#475)
        - _liquidityFee = _buyLiquidityFee (Catecoin.sol#476)
Catcoin.setNumTokensSellToAddToLiquidity(uint256) (Catecoin.sol#482-487) should emit an event for:
        - numTokensSellToAddToLiquidity = amount (Catecoin.sol#486)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-events-arithmetic

Reentrancy in Catcoin._transfer(address,address,uint256) (Catecoin.sol#655-697):
        External calls:
        - swapAndLiquify(contractTokenBalance) (Catecoin.sol#675)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timest
amp) (Catecoin.sol#725-732)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.ti
mestamp) (Catecoin.sol#714-720)
        External calls sending eth:
        - swapAndLiquify(contractTokenBalance) (Catecoin.sol#675)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timest
amp) (Catecoin.sol#725-732)
        State variables written after the call(s):
        External calls sending eth:
        - swapAndLiquify(contractTokenBalance) (Catecoin.sol#675)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timest
amp) (Catecoin.sol#725-732)
        State variables written after the call(s):
        - _liquidityFee = _buyLiquidityFee (Catecoin.sol#685)
        - _liquidityFee = _sellLiquidityFee (Catecoin.sol#689)
        - _liquidityFee = 0 (Catecoin.sol#693)
        - _tokenTransfer(from,to,amount,takeFee) (Catecoin.sol#696)
                - _liquidityFee = _previousLiquidityFee (Catecoin.sol#636)
                - _liquidityFee = 0 (Catecoin.sol#631)
        - _tokenTransfer(from,to,amount,takeFee) (Catecoin.sol#696)
                - _previousLiquidityFee = _liquidityFee (Catecoin.sol#628)
        - _tokenTransfer(from,to,amount,takeFee) (Catecoin.sol#696)
                - _previousTaxFee = _taxFee (Catecoin.sol#627)
        - _tokenTransfer(from,to,amount,takeFee) (Catecoin.sol#696)
                - _tFeeTotal = _tFeeTotal.add(tFee) (Catecoin.sol#506)
        - _taxFee = _buyTaxFee (Catecoin.sol#684)
        - _taxFee = _sellTaxFee (Catecoin.sol#688)
        - _taxFee = 0 (Catecoin.sol#692)
        - _tokenTransfer(from,to,amount,takeFee) (Catecoin.sol#696)
                - _taxFee = _previousTaxFee (Catecoin.sol#635)
                - _taxFee = 0 (Catecoin.sol#630)
Reentrancy in Catcoin.setRouterAddress(address) (Catecoin.sol#489-494):
        External calls:
        - uniswapV2Pair = IUniswapV2Factory(_uniswapV2Router.factory()).createPair(address(this),_uniswapV2Router.WETH()) (Cat
ecoin.sol#491-492)
        State variables written after the call(s):
        - uniswapV2Router = _uniswapV2Router (Catecoin.sol#493)
Reentrancy in Catcoin.swapAndLiquify(uint256) (Catecoin.sol#699-707):
        External calls:
        - swapTokensForEth(half) (Catecoin.sol#703)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.ti
mestamp) (Catecoin.sol#714-720)
        - addLiquidity(otherHalf,newBalance) (Catecoin.sol#705)
        External calls:
        - swapAndLiquify(contractTokenBalance) (Catecoin.sol#675)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timest
amp) (Catecoin.sol#725-732)
                - uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.ti
mestamp) (Catecoin.sol#714-720)
        External calls sending eth:
        - swapAndLiquify(contractTokenBalance) (Catecoin.sol#675)
                - uniswapV2Router.addLiquidityETH{value: ethAmount}(address(this),tokenAmount,0,0,address(0xdead),block.timest
amp) (Catecoin.sol#725-732)
        Event emitted after the call(s):
        - Transfer(sender,recipient,tTransferAmount) (Catecoin.sol#775)
                - _tokenTransfer(from,to,amount,takeFee) (Catecoin.sol#696)
        - Transfer(sender,recipient,tTransferAmount) (Catecoin.sol#817)
                - _tokenTransfer(from,to,amount,takeFee) (Catecoin.sol#696)
        - Transfer(sender,recipient,tTransferAmount) (Catecoin.sol#796)
                - _tokenTransfer(from,to,amount,takeFee) (Catecoin.sol#696)
        - Transfer(sender,recipient,tTransferAmount) (Catecoin.sol#445)
                - _tokenTransfer(from,to,amount,takeFee) (Catecoin.sol#696)
Reentrancy in Catcoin.swapAndLiquify(uint256) (Catecoin.sol#699-707):
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (Catecoin.sol#652)
                - _approve(sender,_msgSender(),_allowances[sender][_msgSender()].sub(amount,ERC20: transfer amount exceeds all
owance)) (Catecoin.sol#316-323)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-3

Catcoin.includeInReward(address) (Catecoin.sol#413-424) has costly operations inside a loop:
        - _excluded.pop() (Catecoin.sol#420)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop
```

```
Catcoin.includeInReward(address) (Catecoin.sol#413-424) has costly operations inside a loop:
        - _excluded.pop() (Catecoin.sol#420)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#costly-operations-inside-a-loop

Context._msgData() (Catecoin.sol#85-88) is never used and should be removed
SafeMath.div(uint256,uint256,string) (Catecoin.sol#68-77) is never used and should be removed
SafeMath.mod(uint256,uint256) (Catecoin.sol#53-55) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code

Catcoin._taxFee (Catecoin.sol#203) is set pre-construction with a non-constant function or state variable:
        - _buyTaxFee
Catcoin._liquidityFee (Catecoin.sol#204) is set pre-construction with a non-constant function or state variable:
        - _buyLiquidityFee
Catcoin._previousTaxFee (Catecoin.sol#206) is set pre-construction with a non-constant function or state variable:
        - _taxFee
Catcoin._previousLiquidityFee (Catecoin.sol#207) is set pre-construction with a non-constant function or state variable:
        - _liquidityFee
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state

Pragma version^0.8.4 (Catecoin.sol#3) allows old versions
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
```

```
Catcoin._burnAddress (Catecoin.sol#186) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant

Catcoin._decimals (Catecoin.sol#193) should be immutable
Catcoin._name (Catecoin.sol#191) should be immutable
Catcoin._symbol (Catecoin.sol#192) should be immutable
Catcoin._tTotal (Catecoin.sol#188) should be immutable
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-immutable
Catecoin.sol analyzed (9 contracts with 84 detectors), 83 result(s) found
```

Solidity Static Analysis

Catcoin.sol

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Catcoin.(): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 228:4:

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Catcoin.setRouterAddress(address): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.

more

Pos: 489:4:

Gas & Economy

Gas costs:

Gas requirement of function Catcoin.includeInReward is infinite: If the gas requirement of a function is higher than the block gas limit, it cannot be executed. Please avoid loops in your functions or actions that modify large areas of storage (this includes clearing or copying arrays in storage)

Pos: 413:4:

For loop over dynamic array:

Loops that do not have a fixed number of iterations, for example, loops that depend on storage values, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point. Additionally, using unbounded loops incurs in a lot of avoidable gas costs. Carefully test how many items at maximum you can pass to such functions to make it successful.

more

Pos: 586:8:

Email: audit@EtherAuthority.io

ERC

ERC20:

ERC20 contract's "decimals" function should have "uint8" as return type

more

Pos: 270:4:

Miscellaneous

Constant/View/Pure functions:

Catcoin.reflectionFromToken(uint256,bool) : Is constant but potentially should not be.
Note: Modifiers are currently not considered by this static analysis.

more

Pos: 376:4:

Guard conditions:

Use "assert(x)" if you never ever want x to be false, not in any circumstance (apart from a bug in your code). Use "require(x)" if x can be false, due to e.g. invalid input or a failing external component.

more

Pos: 827:8:

Data truncated:

Division of integer values yields an integer value again. That means e.g. 10 / 100 = 0 instead of 0.1 since the result is an integer again. This does not hold for division of (only) literal values since those yield rational constants.

Pos: 75:19:

Solhint Linter

Catcoin.sol

```
Catcoin.sol:62:18: Error: Parse error: missing ';' at '{'
Catcoin.sol:73:18: Error: Parse error: missing ';' at '{'
```

Software analysis result:

These software reported many false positive results and some are informational issues. So, those issues can be safely ignored.

Email: audit@EtherAuthority.io