SecureWise

AUDIT REPORT
SecureWise

CORE ZILLA

https://securewise.info/

https://t.me/securewisehub

https://github.com/securewise

Table of Contents

Disclaimer

SecureWise provides the smart contract audit of solidity. Audit and report are for informational purposes only and not, nor should be considered, as an endorsement to engage with, invest in, participate, provide an incentive, or disapprove, criticise, discourage, or purport to provide an opinion on any particular project or team.

This audit report doesn't provide any warranty or guarantee regarding the nature of the technology analysed. These reports, in no way, provide investment advice, nor should be used as investment advice of any sort. Investors must always do their own research and manage their risk.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis, and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and SecureWise and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) SecureWise owe no duty of care towards you or any other person, nor does SecureWise make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and SecureWise hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, SecureWise hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against SecureWise, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report.

Overview

Token Name: CoreZilla**(Czilla)**

Methodology: Automated Analysis, Manual Code Review

Language: Solidity

Contract Address: 0x1dAE0f16307Ea40fCC34bB1aF56B9e94A984793C

ContractLink: https://scan.coredao.org/address/0x1dAE0f16307Ea40fCC34bB1aF56B9e94A984793C

Network: Core

Supply: 1.000.000.000.000

Website: -

Twitter: https://twitter.com/corezilla83

Telegram: https://t.me/CorezillaCZ

Report Date: March 3, 2023

Quick Result

SecureWise has applied the automated and manual analysis of Smart Contract and were reviewed for common contract vulnerabilities and centralized exploits

⚠	Contract locking ether found
⚠	The owner can exclude accounts from fees
⚠	The owner can set fees with limit up to 10%
⚠	The owner can set max transaction amount within reasonable limits

Page 13 for more details

CoreZilla(Czilla) has succesfully **PASSED** the smart contract audit with **LOW severity issue**

Auditing Approach and Methodologies

SecureWise has performed starting with analyzing the code, issues, code quality, and libraries. Reviewed line-by-line by our team. Finding any potential issue like race conditions, transaction-ordering dependence, timestamp dependence, and denial of service attacks.

Methodology

- Understanding the size, scope and functionality of your project's source code
- Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
- Testing and automated analysis of the Smart Contract to determine proper logic has been followed throughout the whole process
- Deploying the code on testnet using multiple live test
- Analyzing a program to determine the specific input that causes different parts of a program to execute its functions.
- Checking whether all the libraries used in the code are on the latest version.

Goals

Smart Contract System is secure, resilient and working according to the specifications and without any vulnerabilities.

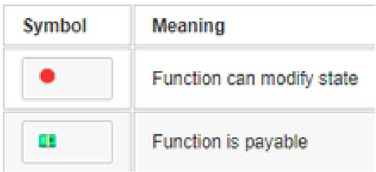
Risk Classification

High: Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, of the contract and its functions. Must be fixed as soon as possible.

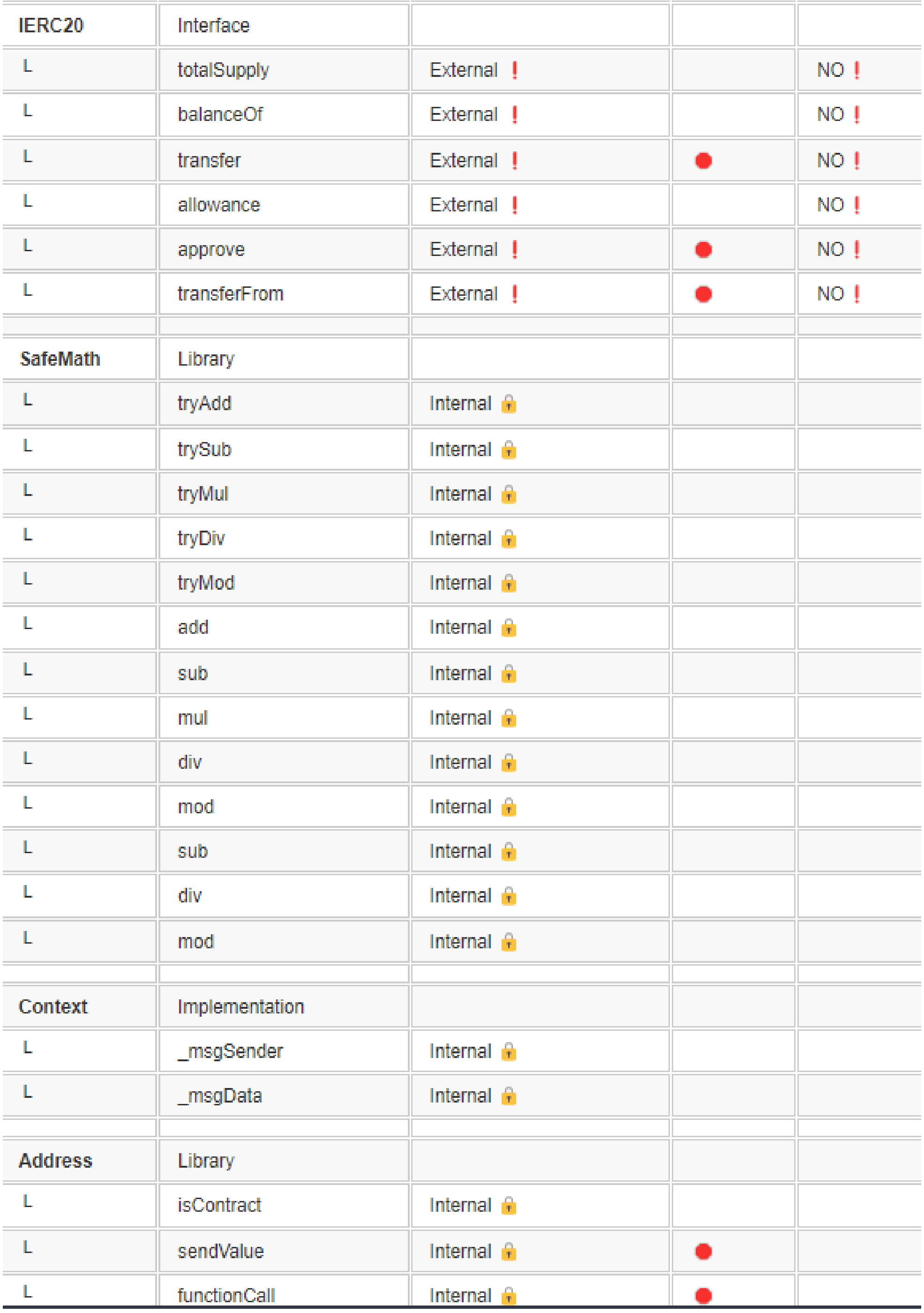
Medium: Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Must be fxed as soon as possible.

Low: Effects are minimal in isolation and do not pose a signifcant danger to the project or its users. Issues under this classifcation are recommended to be fixed nonetheless.

Automated Analysis

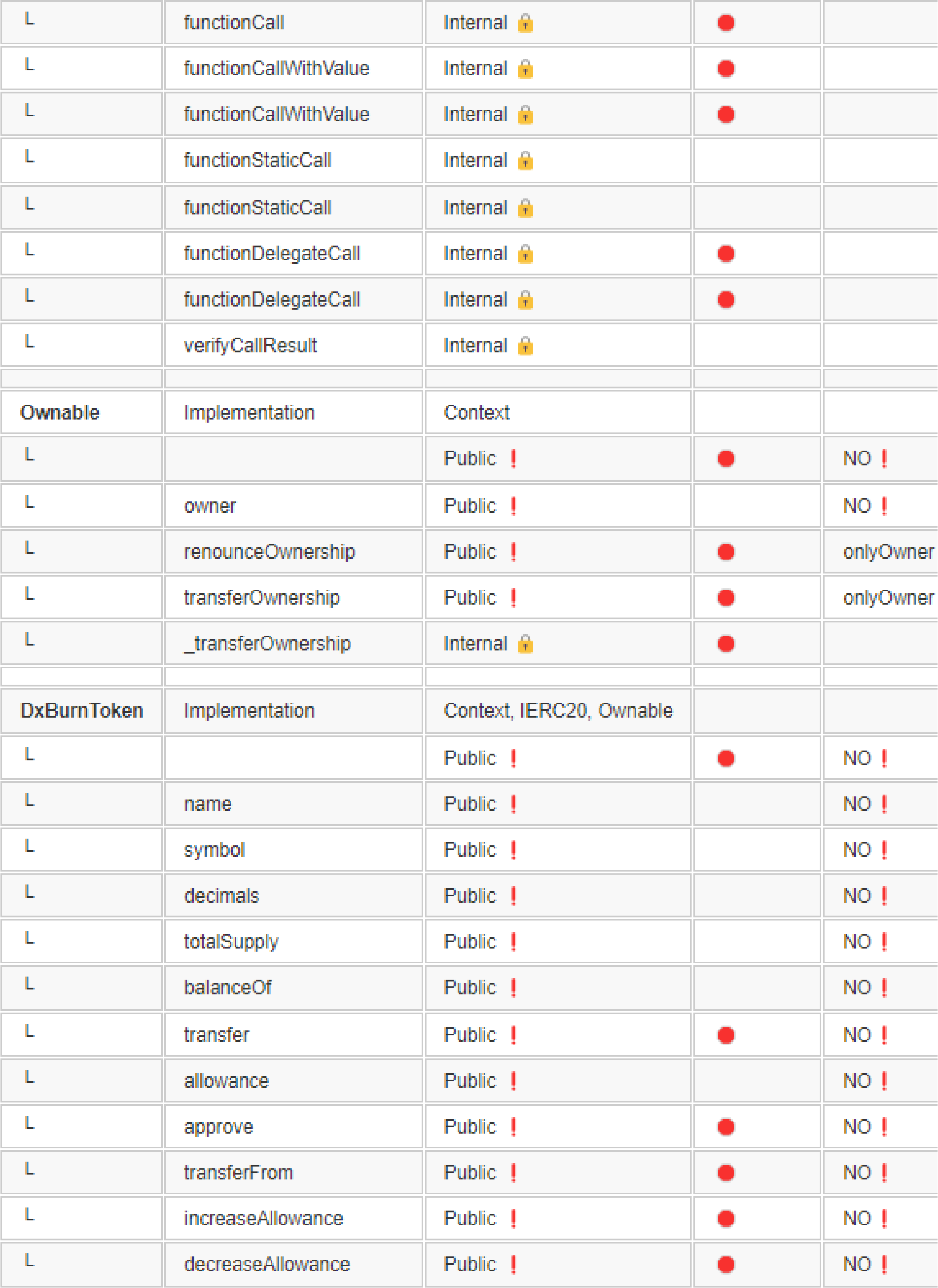


Symbol	Meaning
●	Function can modify state
💵	Function is payable



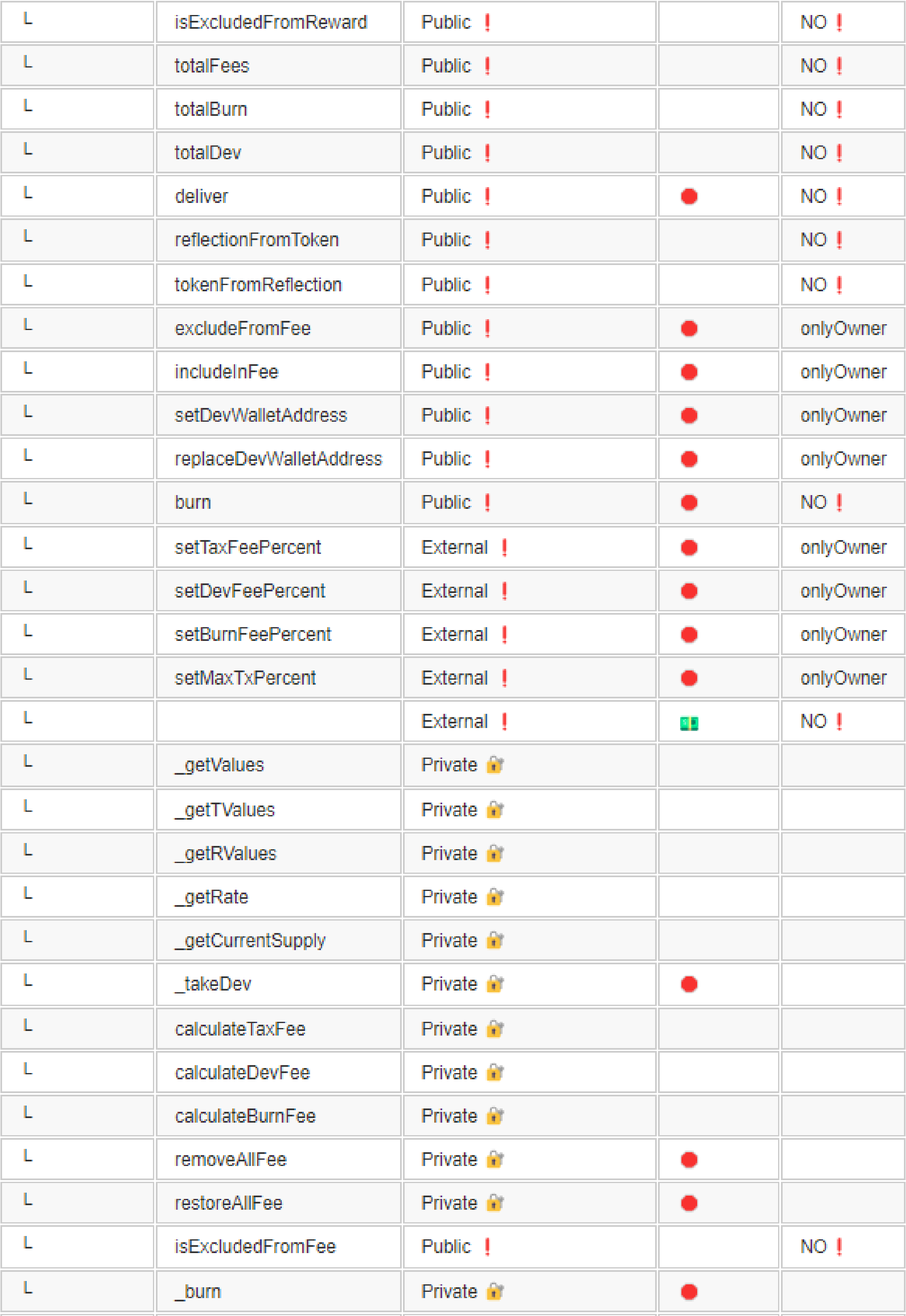
IERC20	Interface			
L	totalSupply	External ❗		NO ❗
L	balanceOf	External ❗		NO ❗
L	transfer	External ❗	●	NO ❗
L	allowance	External ❗		NO ❗
L	approve	External ❗	●	NO ❗
L	transferFrom	External ❗	●	NO ❗
SafeMath	Library			
L	tryAdd	Internal 🔒		
L	trySub	Internal 🔒		
L	tryMul	Internal 🔒		
L	tryDiv	Internal 🔒		
L	tryMod	Internal 🔒		
L	add	Internal 🔒		
L	sub	Internal 🔒		
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	mod	Internal 🔒		
L	sub	Internal 🔒		
L	div	Internal 🔒		
L	mod	Internal 🔒		
Context	Implementation			
L	_msgSender	Internal 🔒		
L	_msgData	Internal 🔒		
Address	Library			
L	isContract	Internal 🔒		
L	sendValue	Internal 🔒	●	
L	functionCall	Internal 🔒	●	

Automated Analysis



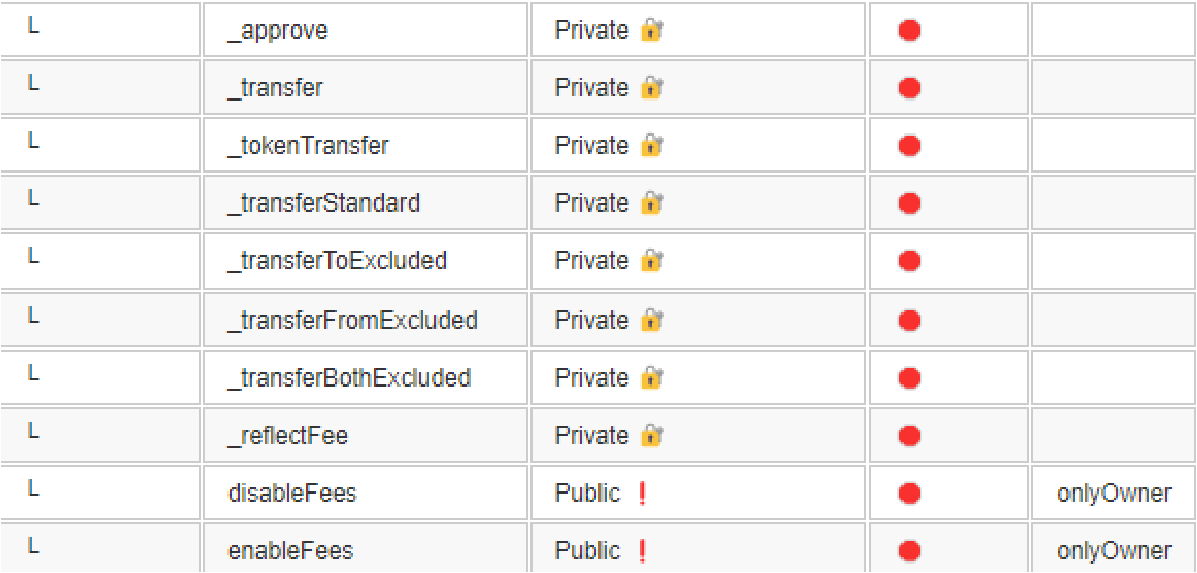
L	functionCall	Internal 🔒	●	
L	functionCallWithValue	Internal 🔒	●	
L	functionCallWithValue	Internal 🔒	●	
L	functionStaticCall	Internal 🔒		
L	functionStaticCall	Internal 🔒		
L	functionDelegateCall	Internal 🔒	●	
L	functionDelegateCall	Internal 🔒	●	
L	verifyCallResult	Internal 🔒		
Ownable	Implementation	Context		
L		Public ❗	●	NO ❗
L	owner	Public ❗		NO ❗
L	renounceOwnership	Public ❗	●	onlyOwner
L	transferOwnership	Public ❗	●	onlyOwner
L	_transferOwnership	Internal 🔒	●	
DxBurnToken	Implementation	Context, IERC20, Ownable		
L		Public ❗	●	NO ❗
L	name	Public ❗		NO ❗
L	symbol	Public ❗		NO ❗
L	decimals	Public ❗		NO ❗
L	totalSupply	Public ❗		NO ❗
L	balanceOf	Public ❗		NO ❗
L	transfer	Public ❗	●	NO ❗
L	allowance	Public ❗		NO ❗
L	approve	Public ❗	●	NO ❗
L	transferFrom	Public ❗	●	NO ❗
L	increaseAllowance	Public ❗	●	NO ❗
L	decreaseAllowance	Public ❗	●	NO ❗

Automated Analysis



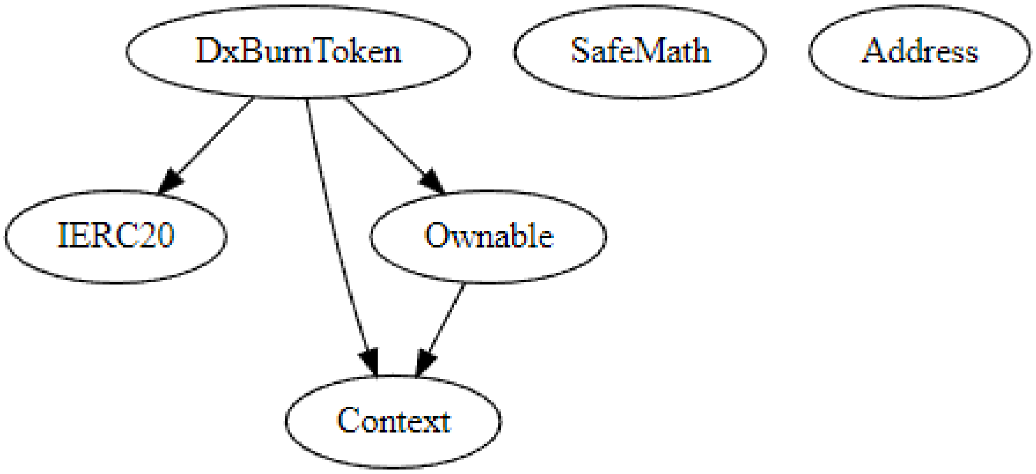
└	isExcludedFromReward	Public ❗		NO ❗
└	totalFees	Public ❗		NO ❗
└	totalBurn	Public ❗		NO ❗
└	totalDev	Public ❗		NO ❗
└	deliver	Public ❗	🔴	NO ❗
└	reflectionFromToken	Public ❗		NO ❗
└	tokenFromReflection	Public ❗		NO ❗
└	excludeFromFee	Public ❗	🔴	onlyOwner
└	includeInFee	Public ❗	🔴	onlyOwner
└	setDevWalletAddress	Public ❗	🔴	onlyOwner
└	replaceDevWalletAddress	Public ❗	🔴	onlyOwner
└	burn	Public ❗	🔴	NO ❗
└	setTaxFeePercent	External ❗	🔴	onlyOwner
└	setDevFeePercent	External ❗	🔴	onlyOwner
└	setBurnFeePercent	External ❗	🔴	onlyOwner
└	setMaxTxPercent	External ❗	🔴	onlyOwner
└		External ❗	💵	NO ❗
└	_getValues	Private 🔐		
└	_getTValues	Private 🔐		
└	_getRValues	Private 🔐		
└	_getRate	Private 🔐		
└	_getCurrentSupply	Private 🔐		
└	_takeDev	Private 🔐	🔴	
└	calculateTaxFee	Private 🔐		
└	calculateDevFee	Private 🔐		
└	calculateBurnFee	Private 🔐		
└	removeAllFee	Private 🔐	🔴	
└	restoreAllFee	Private 🔐	🔴	
└	isExcludedFromFee	Public ❗		NO ❗
└	_burn	Private 🔐	🔴	

Automated Analysis

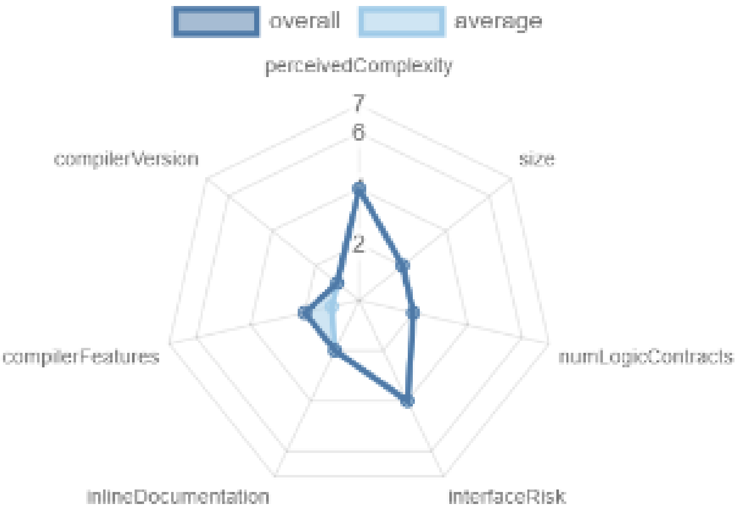


L	_approve	Private 🔐	●	
L	_transfer	Private 🔐	●	
L	_tokenTransfer	Private 🔐	●	
L	_transferStandard	Private 🔐	●	
L	_transferToExcluded	Private 🔐	●	
L	_transferFromExcluded	Private 🔐	●	
L	_transferBothExcluded	Private 🔐	●	
L	_reflectFee	Private 🔐	●	
L	disableFees	Public ❗	●	onlyOwner
L	enableFees	Public ❗	●	onlyOwner

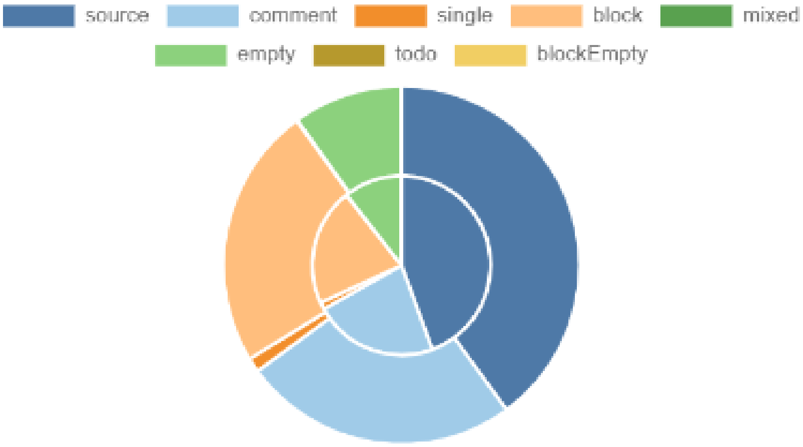
Inheritance Graph



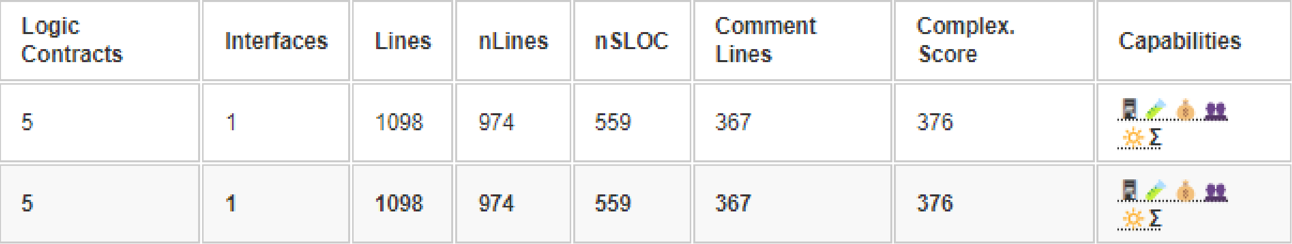
Risk



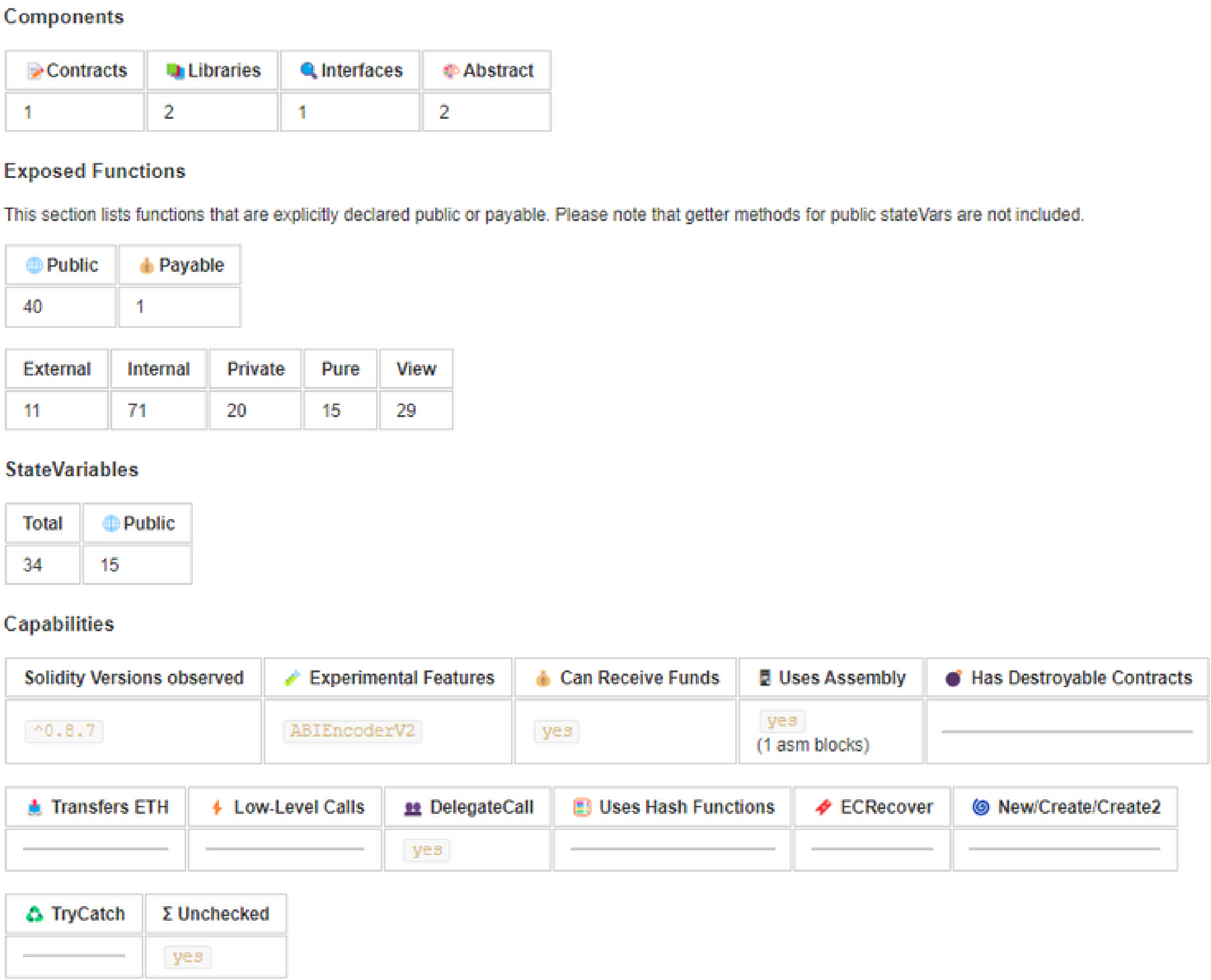
Source Lines



Contract Summary



Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
5	1	1098	974	559	367	376	🖥 ✏ 💰 👥 ☀ Σ
5	1	1098	974	559	367	376	🖥 ✏ 💰 👥 ☀ Σ



Components

Contracts	Libraries	Interfaces	Abstract
1	2	1	2

Exposed Functions

This section lists functions that are explicitly declared public or payable. Please note that getter methods for public stateVars are not included.

Public	Payable
40	1

External	Internal	Private	Pure	View
11	71	20	15	29

StateVariables

Total	Public
34	15

Capabilities

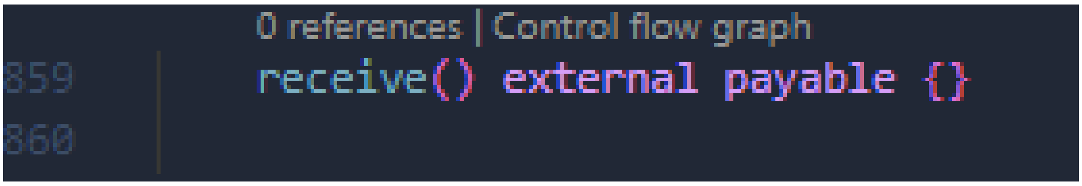
Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
`^0.8.7`	`ABIEncoderV2`	`yes`	`yes` (1 asm blocks)	—

Transfers ETH	Low-Level Calls	DelegateCall	Uses Hash Functions	ECRecover	New/Create/Create2
—	—	`yes`	—	—	—

TryCatch	Σ Unchecked
—	`yes`

Manual Review

Contract locking ether found



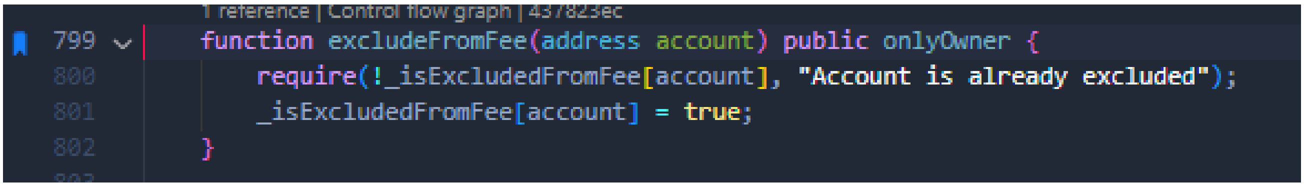
```
receive() external payable {}
```

A contract can now have only one receive function that is declared with the syntax **receive() external payable {...}** (without the function keyword). It executes on calls to the contract with **no data (calldata)**, such as **calls** made via **send() or transfer().** But does not have a function to withdraw the ether. If someone accidentally send ether to contract. Ether will stuck on the contract.

Recommendation

*You should use write withdraw function **but one important thing** you shouldn't withdraw native token from the contract. (e.g, put require address(this))*

The owner can exclude accounts from fees



```
function excludeFromFee(address account) public onlyOwner {
    require(!_isExcludedFromFee[account], "Account is already excluded");
    _isExcludedFromFee[account] = true;
}
```

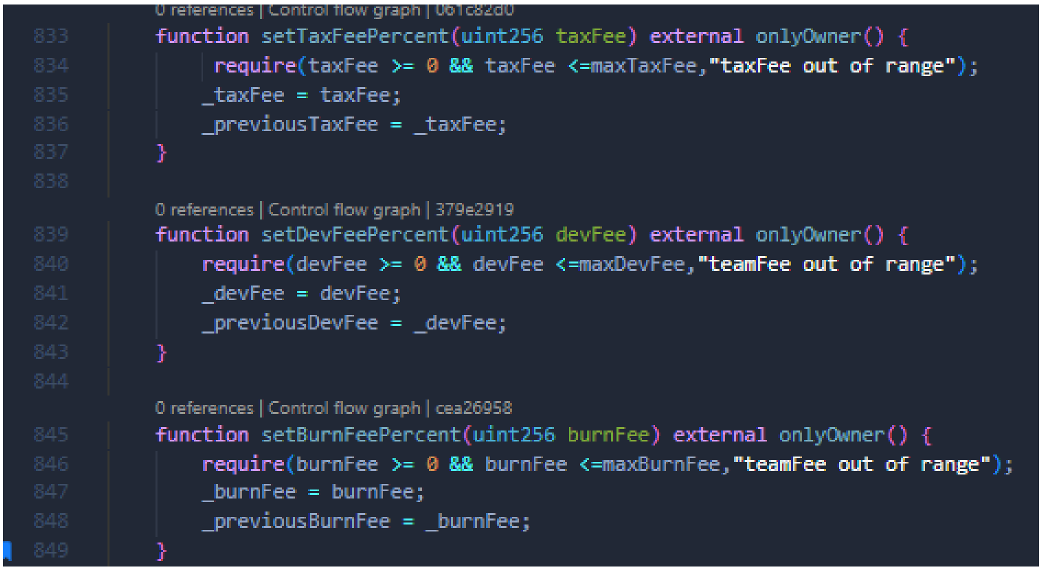
Authorizing privileged roles to exclude accounts from fees. These cause can affect decentralization. After excluding the user from accounts, the user trades without paying a any fee and the other user sees it). But may apply in some cases like (owner wallets, contract...)

Recommendation

You should carefully manage the private key of the owner's account. You should use powerful security mechanism that will prevent a single user from accessing the contract owner functions. That risk can be prevented by temporarily locking the contract or renouncing ownership

Manual Review

The owner can set fees with limit up to 10%



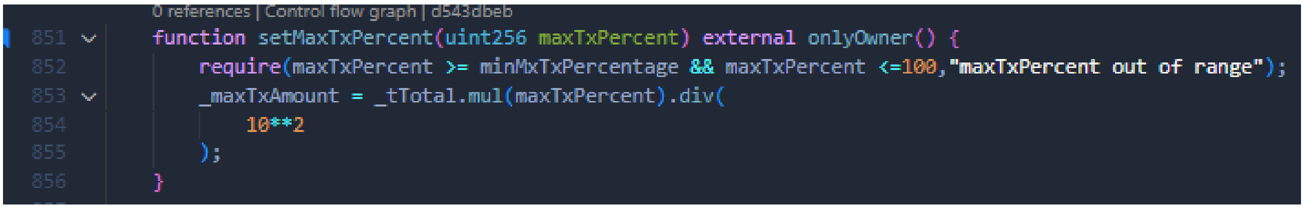
```
0 references | Control flow graph | 061c82d0
function setTaxFeePercent(uint256 taxFee) external onlyOwner() {
    require(taxFee >= 0 && taxFee <=maxTaxFee,"taxFee out of range");
    _taxFee = taxFee;
    _previousTaxFee = _taxFee;
}

0 references | Control flow graph | 379e2919
function setDevFeePercent(uint256 devFee) external onlyOwner() {
    require(devFee >= 0 && devFee <=maxDevFee,"teamFee out of range");
    _devFee = devFee;
    _previousDevFee = _devFee;
}

0 references | Control flow graph | cea26958
function setBurnFeePercent(uint256 burnFee) external onlyOwner() {
    require(burnFee >= 0 && burnFee <=maxBurnFee,"teamFee out of range");
    _burnFee = burnFee;
    _previousBurnFee = _burnFee;
}
```

The owner can set fees within reasonable limits.

The owner can set max transaction amount within reasonable limits



```
0 references | Control flow graph | d543dbeb
function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
    require(maxTxPercent >= minMxTxPercentage && maxTxPercent <=100,"maxTxPercent out of range");
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(
        10**2
    );
}
```

The owner can set maximum transaction within reasonable limits.

Manual Review

Access Modifiers Vulnerabilities

```
burn()
renounceOwnership()
totalSupply()
replaceDevWalletAddress()
transferOwnership()
enableFees()
disableFees()
decimals()
deliver()
decreaseAllowance()
symbol()
balanceOf()
transfer()
increaseAllowance()
name()
approve()
allowance()
```

These functions are used as public instead of external.

Recommendation

Access control identifiers must be authenticated and set adequately to avoid possible vulnerabilities

Out date compiler version

```
pragma solidity ^0.8.7;
```

Compiler is set an outdated version.

Recommendation

Set and use new versions

Manual Review

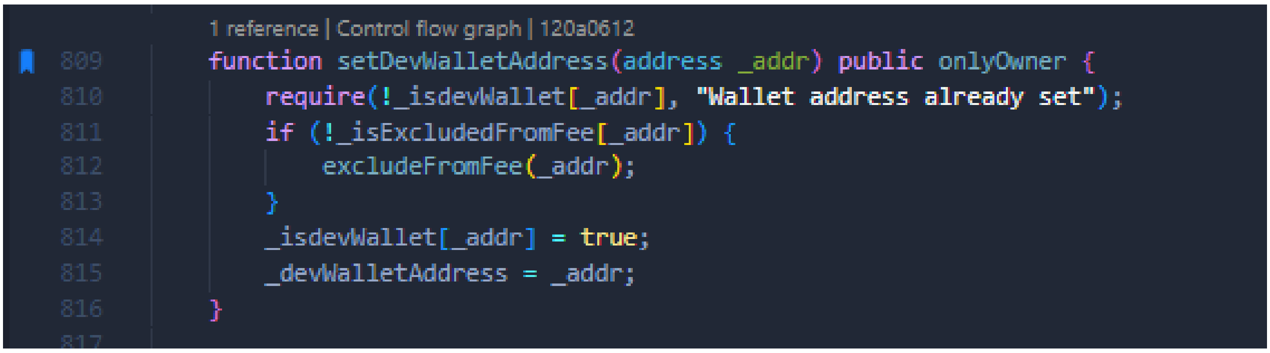
Floating Pragma

```
pragma solidity ^0.8.7;
```

Recommendation

Lock the pragma version and also consider known bugs (https://github.com/ethereum/solidity/releases) for the compiler version that is chosen.

Lacks a zero-check on set wallets function



```
function setDevWalletAddress(address _addr) public onlyOwner {
    require(!_isdevWallet[_addr], "Wallet address already set");
    if (!_isExcludedFromFee[_addr]) {
        excludeFromFee(_addr);
    }
    _isdevWallet[_addr] = true;
    _devWalletAddress = _addr;
}
```

Zero-address checks as input validation on address parameters is always a best practice. This is especially true for critical addresses that are immutable and set in the constructor because they cannot be changed later. Accidentally using zero addresses here will lead to failing logic or force contract redeployment and increased gas costs.

Recommendation

Add zero-address input validation for these addresses.

AUDIT REPORT

SecureWise

https://securewise.info/ https://t.me/securewisehub https://github.com/securewise