AUDIT REPORT

SecureWise

SMART CONTRACT AUDIT

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

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: DOGE SQ

Methodology: Automated Analysis, Manual Code Review

Language: Solidity

Contract Address: 0x219A756D08694Cbe0b8f4d0298094104A2ED1357

ContractLink: https://bscscan.com/address/0x219A756D08694Cbe0b8f4d0298094104A2ED1357

Network: Binance Smart Chain (BSC)

Decimals: 9

Supply: 100,000,000,000.0

Website: https://dogesq.io/

Twitter: https://twitter.com/Dogesq_official

Telegram: https://t.me/DogeSQArmy

Report Date: September 20, 2022

Quick Result

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

Owner Privileges

- ⚠ The owner can stop trading
- ⚠ The owner can set fees up to 100%
- ⚠ Auto liquidity is going to an externally owned account
- ⚠ The owner can set a blacklist any account.
- ⚠ The owner can change transaction lock-time without limit
- ⚠ The owner can change max wallet token amount to "0"
- ⚠ The owner can exclude accounts from fees

DOGE SQ has succesfully **PASSED** the smart contract audit with **HIGH** **MEDIUM** and **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.

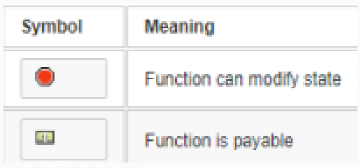
Risk Classification

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.

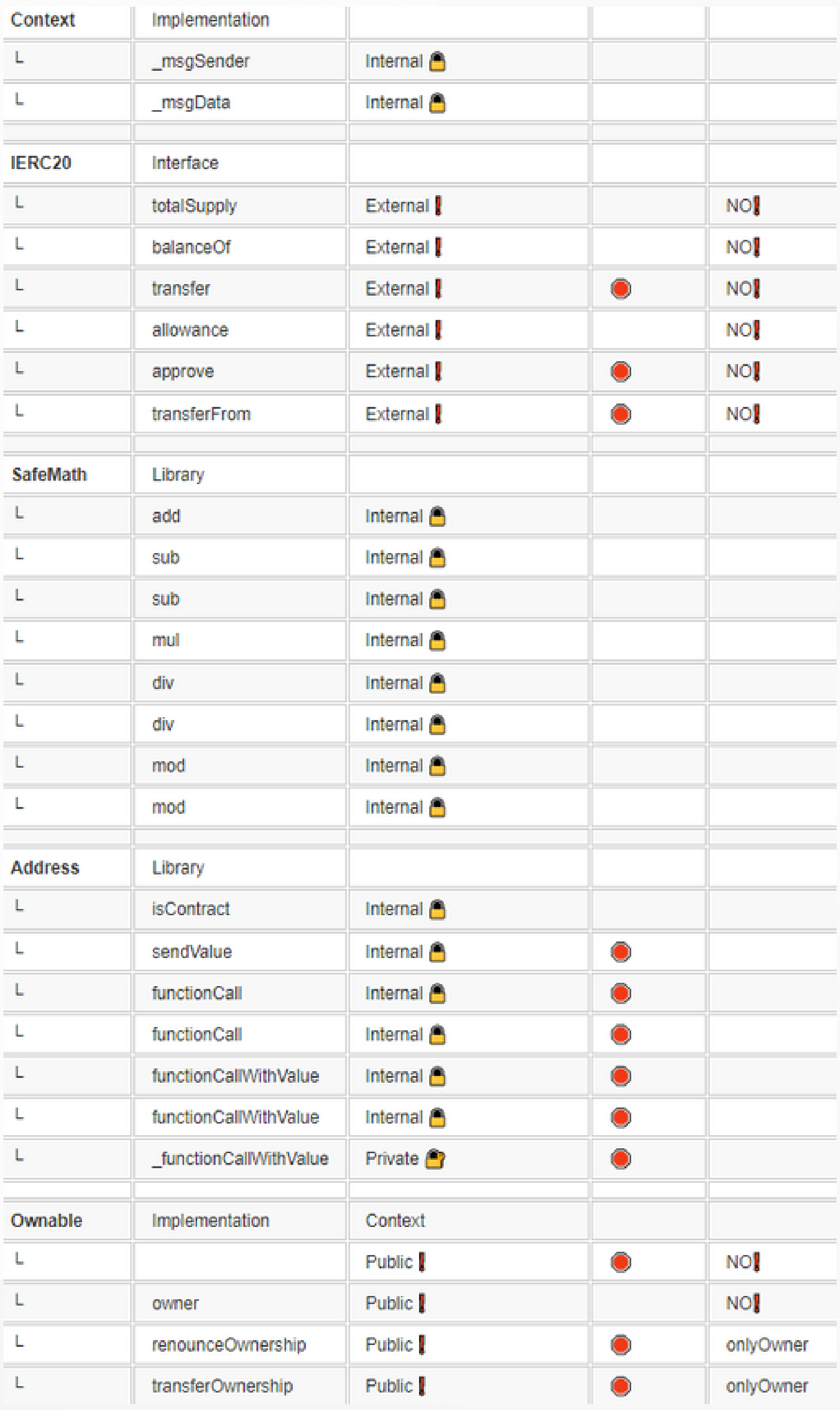
Medium: Bugs or issues with that may be subject to exploit, though their impact is somewhat limited. Must be fxed as soon as possible.

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

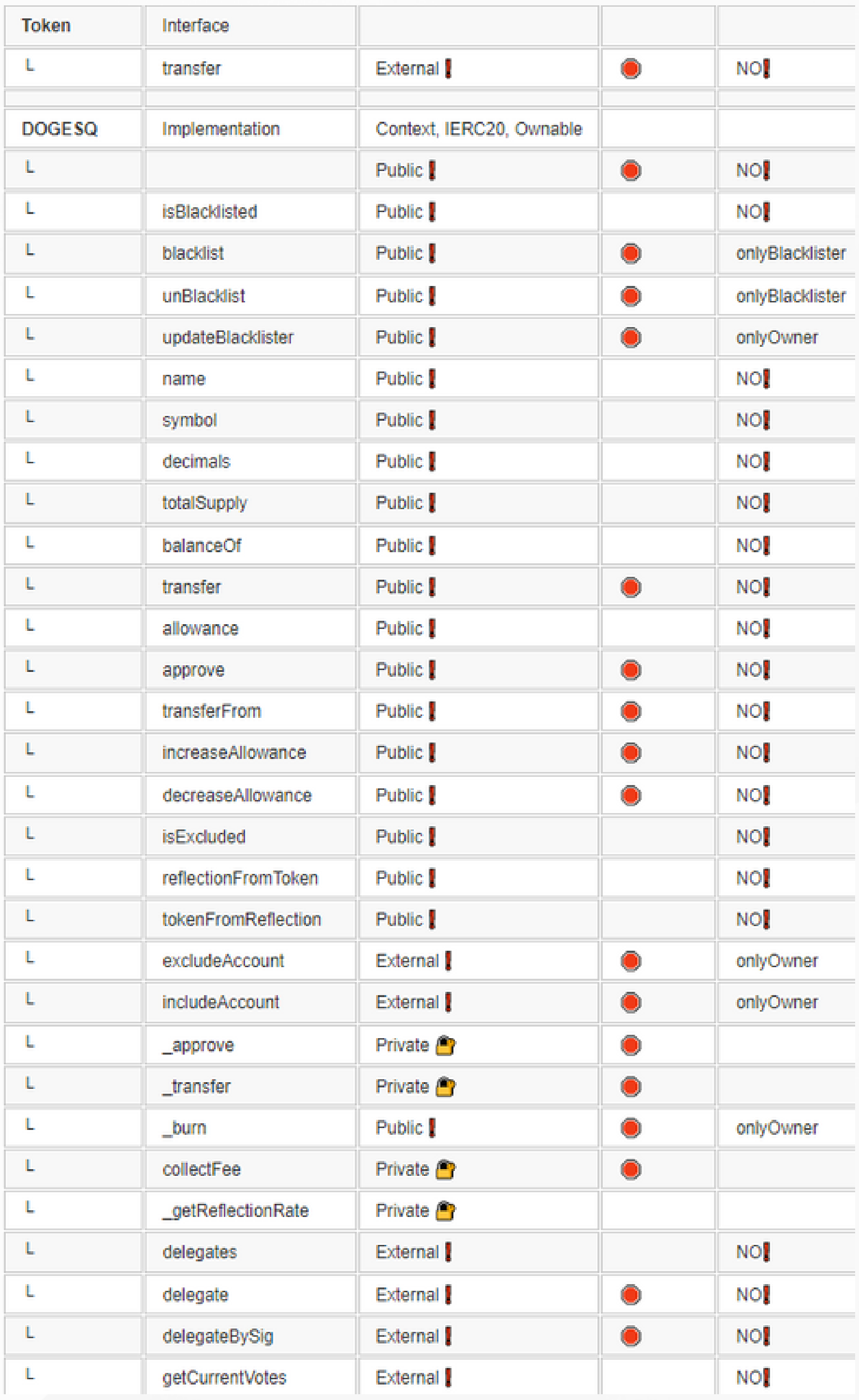


Symbol	Meaning
●	Function can modify state
💵	Function is payable



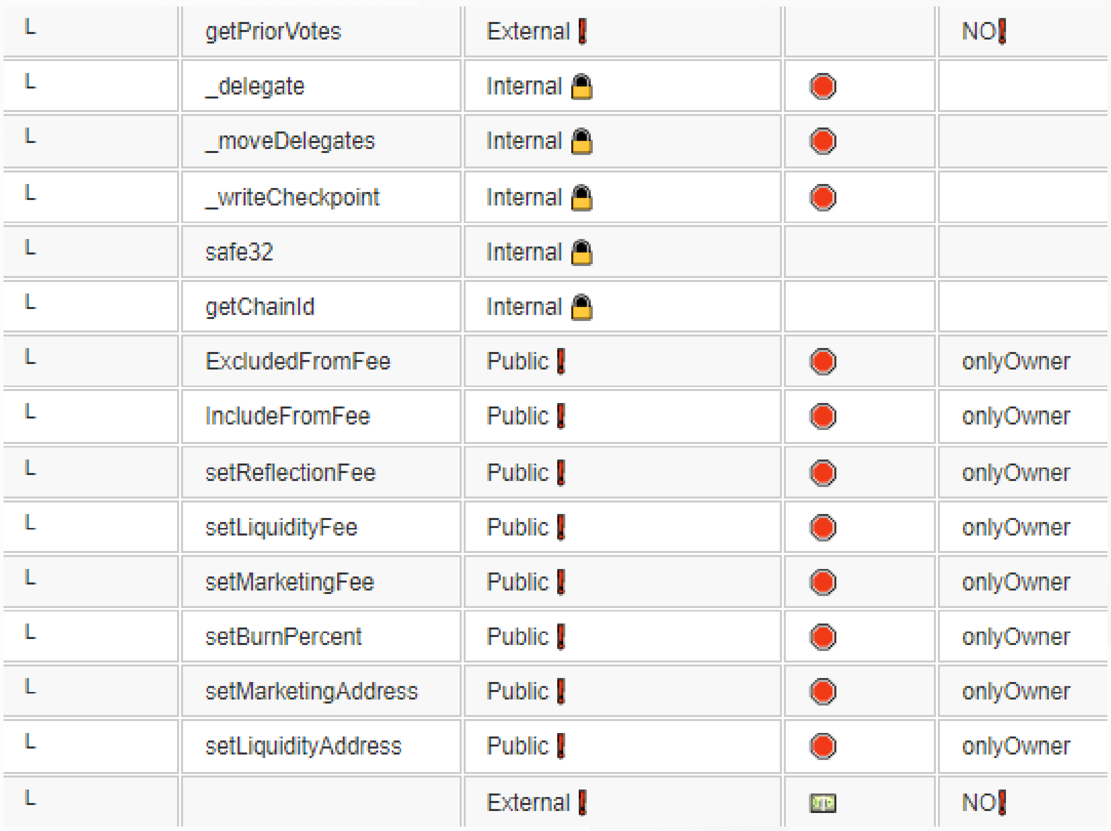
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
IERC20	Interface			
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	●	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	●	NO❗
└	transferFrom	External ❗	●	NO❗
SafeMath	Library			
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
└	mod	Internal 🔒		
Address	Library			
└	isContract	Internal 🔒		
└	sendValue	Internal 🔒	●	
└	functionCall	Internal 🔒	●	
└	functionCall	Internal 🔒	●	
└	functionCallWithValue	Internal 🔒	●	
└	functionCallWithValue	Internal 🔒	●	
└	_functionCallWithValue	Private 🔐	●	
Ownable	Implementation	Context		
└		Public ❗	●	NO❗
└	owner	Public ❗		NO❗
└	renounceOwnership	Public ❗	●	onlyOwner
└	transferOwnership	Public ❗	●	onlyOwner

Automated Analysis



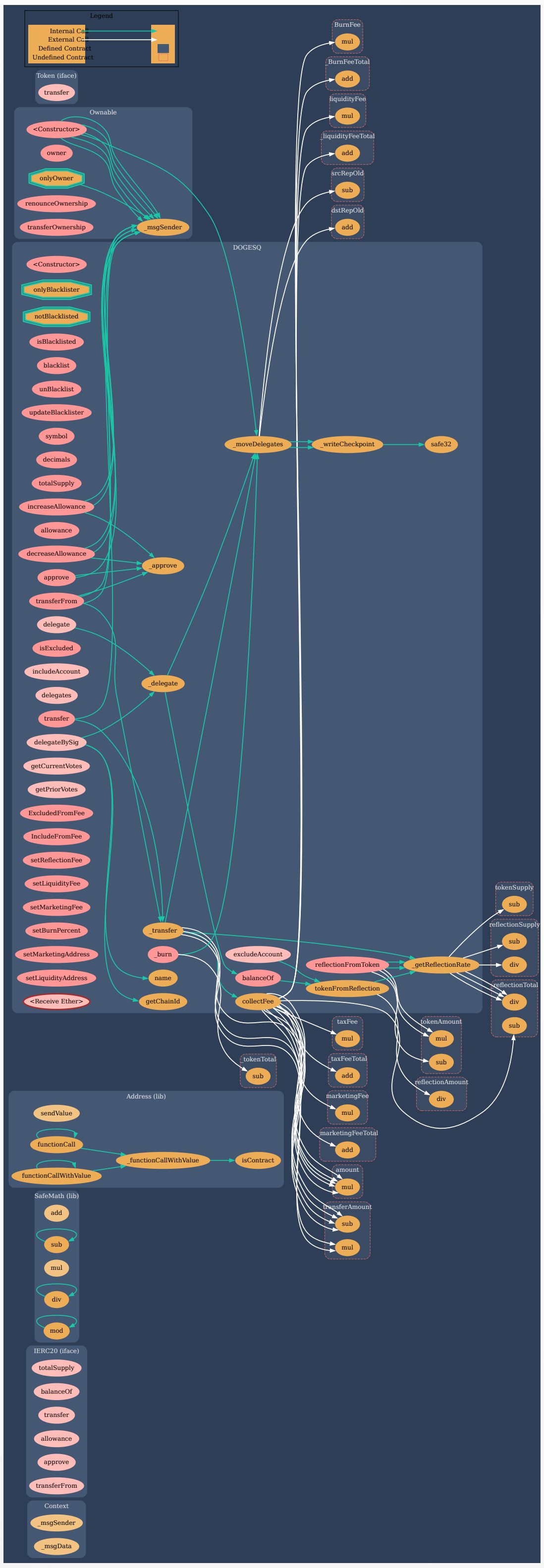
Token	Interface			
└	transfer	External ❗	🔴	NO❗
DOGESQ	Implementation	Context, IERC20, Ownable		
└		Public ❗	🔴	NO❗
└	isBlacklisted	Public ❗		NO❗
└	blacklist	Public ❗	🔴	onlyBlacklister
└	unBlacklist	Public ❗	🔴	onlyBlacklister
└	updateBlacklister	Public ❗	🔴	onlyOwner
└	name	Public ❗		NO❗
└	symbol	Public ❗		NO❗
└	decimals	Public ❗		NO❗
└	totalSupply	Public ❗		NO❗
└	balanceOf	Public ❗		NO❗
└	transfer	Public ❗	🔴	NO❗
└	allowance	Public ❗		NO❗
└	approve	Public ❗	🔴	NO❗
└	transferFrom	Public ❗	🔴	NO❗
└	increaseAllowance	Public ❗	🔴	NO❗
└	decreaseAllowance	Public ❗	🔴	NO❗
└	isExcluded	Public ❗		NO❗
└	reflectionFromToken	Public ❗		NO❗
└	tokenFromReflection	Public ❗		NO❗
└	excludeAccount	External ❗	🔴	onlyOwner
└	includeAccount	External ❗	🔴	onlyOwner
└	_approve	Private 🔐	🔴	
└	_transfer	Private 🔐	🔴	
└	_burn	Public ❗	🔴	onlyOwner
└	collectFee	Private 🔐	🔴	
└	_getReflectionRate	Private 🔐		
└	delegates	External ❗		NO❗
└	delegate	External ❗	🔴	NO❗
└	delegateBySig	External ❗	🔴	NO❗
└	getCurrentVotes	External ❗		NO❗

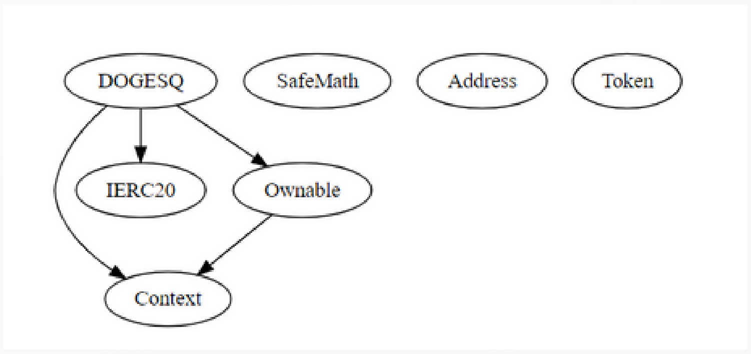
Automated Analysis

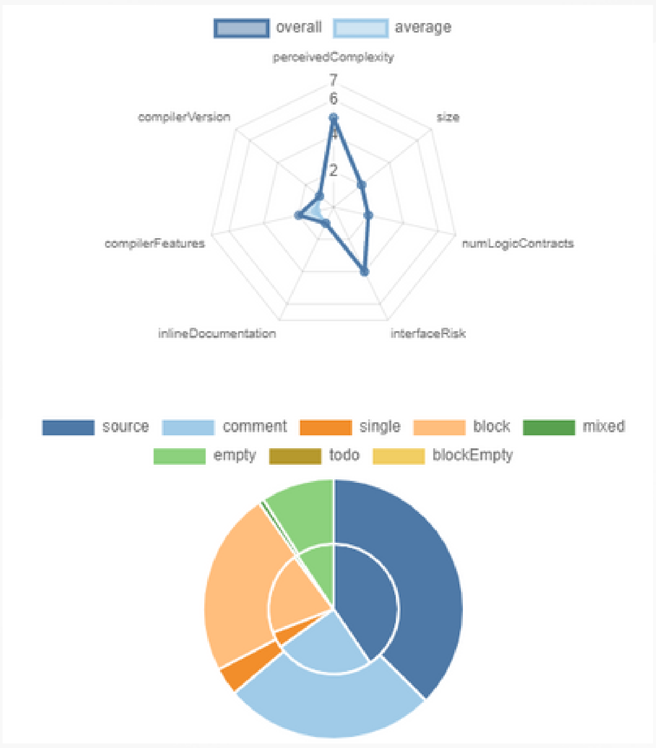


L	getPriorVotes	External ❗		NO❗
L	_delegate	Internal 🔒	🛑	
L	_moveDelegates	Internal 🔒	🛑	
L	_writeCheckpoint	Internal 🔒	🛑	
L	safe32	Internal 🔒		
L	getChainId	Internal 🔒		
L	ExcludedFromFee	Public ❗	🛑	onlyOwner
L	IncludeFromFee	Public ❗	🛑	onlyOwner
L	setReflectionFee	Public ❗	🛑	onlyOwner
L	setLiquidityFee	Public ❗	🛑	onlyOwner
L	setMarketingFee	Public ❗	🛑	onlyOwner
L	setBurnPercent	Public ❗	🛑	onlyOwner
L	setMarketingAddress	Public ❗	🛑	onlyOwner
L	setLiquidityAddress	Public ❗	🛑	onlyOwner
L		External ❗	💵	NO❗

Inheritance Graph

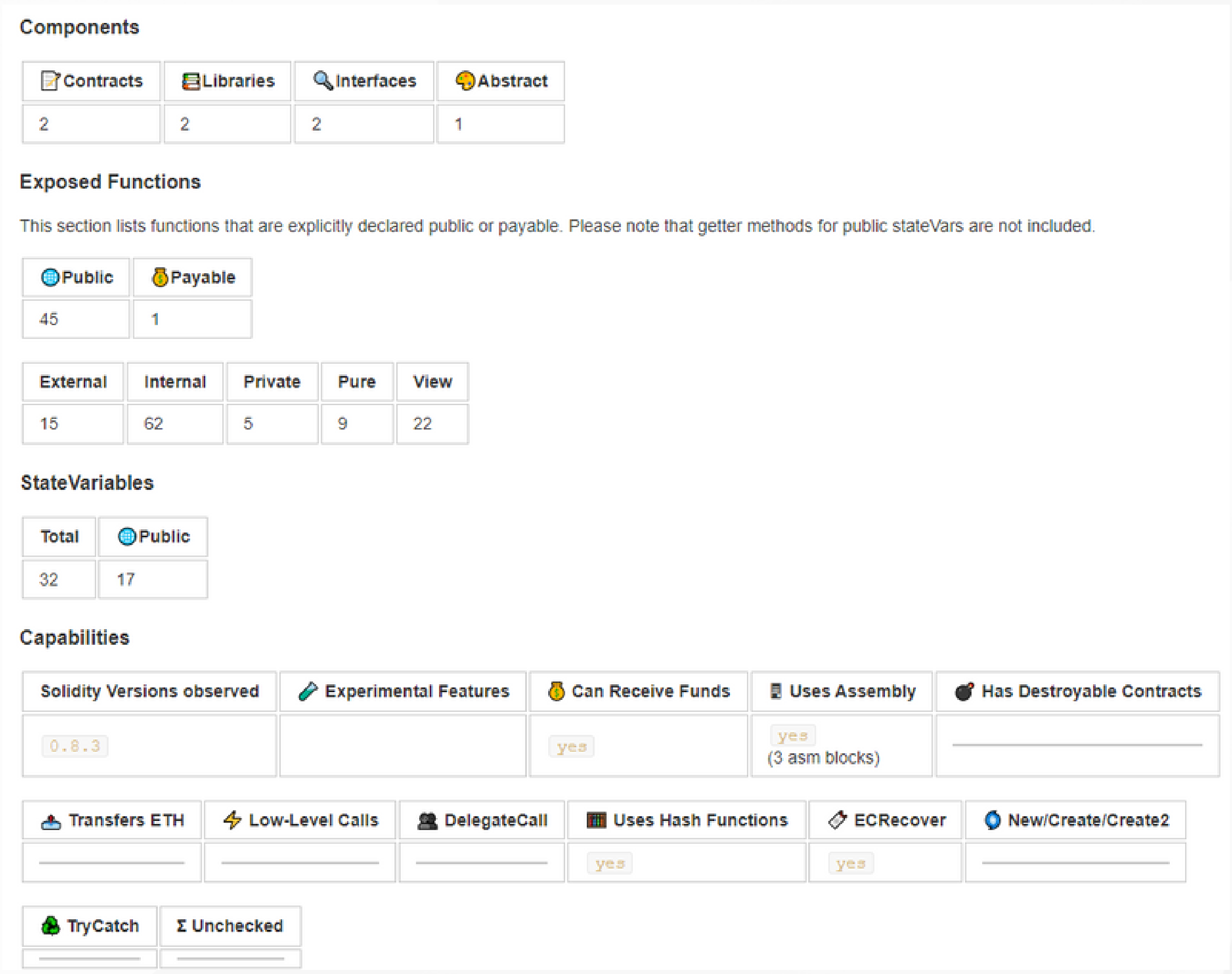






Contract Summary

Logic Contracts	Interfaces	Lines	nLines	nSLOC	Comment Lines	Complex. Score	Capabilities
5	2	1077	961	527	394	410	🖥💰🧮🏷🌀
5	**2**	**1077**	**961**	**527**	**394**	**410**	🖥💰🧮🏷🌀



Components

Contracts	Libraries	Interfaces	Abstract
2	2	2	1

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
45	1

External	Internal	Private	Pure	View
15	62	5	9	22

StateVariables

Total	Public
32	17

Capabilities

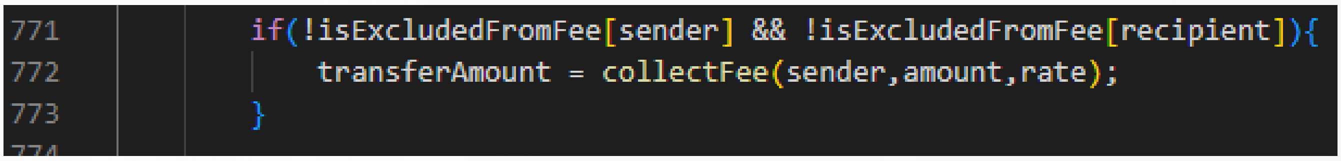
Solidity Versions observed	Experimental Features	Can Receive Funds	Uses Assembly	Has Destroyable Contracts
`0.8.3`		yes	yes (3 asm blocks)	

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

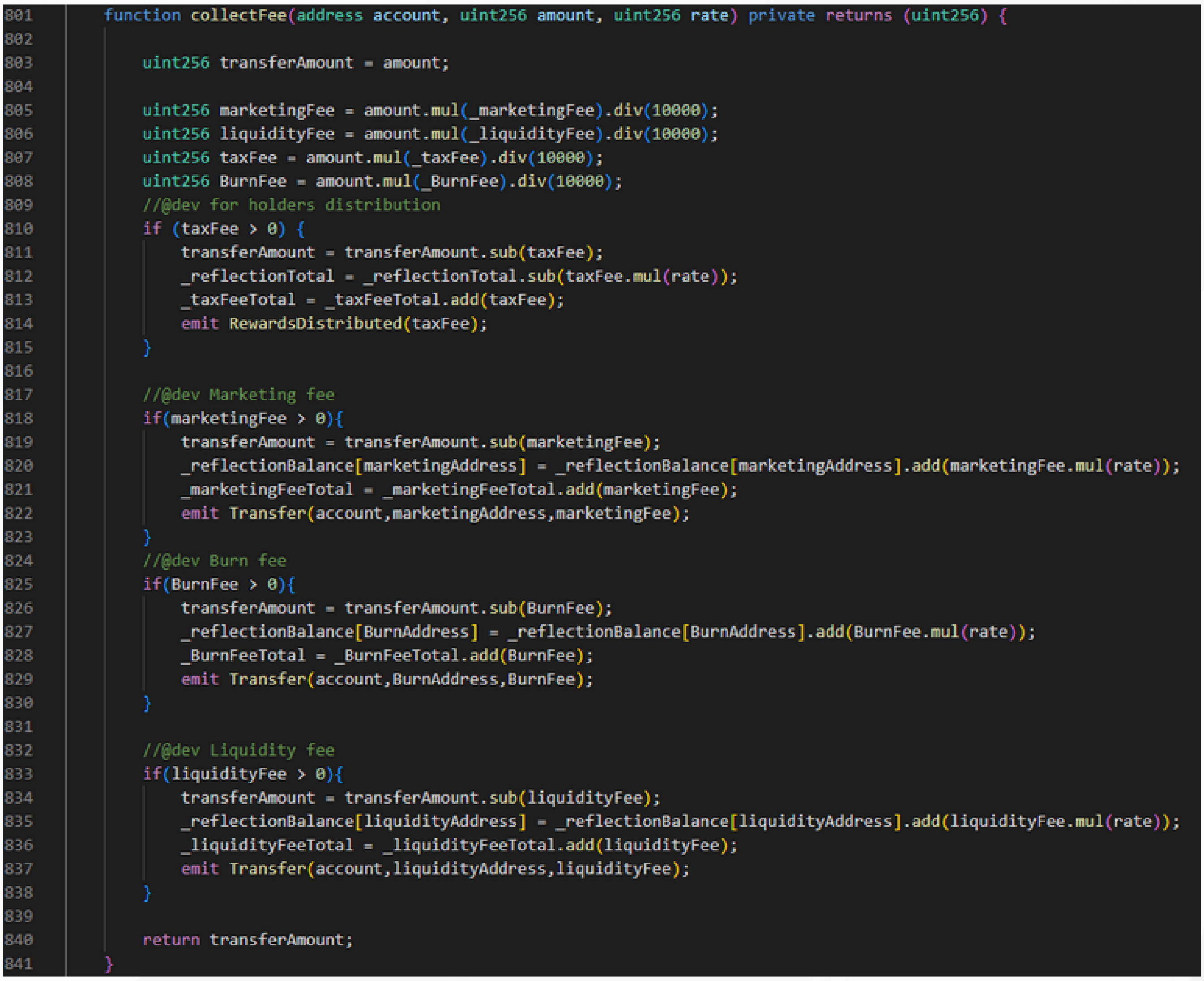
TryCatch	Σ Unchecked

Manual Review

The owner can pause trading



```
        if(!isExcludedFromFee[sender] && !isExcludedFromFee[recipient]){
            transferAmount = collectFee(sender,amount,rate);
        }
```



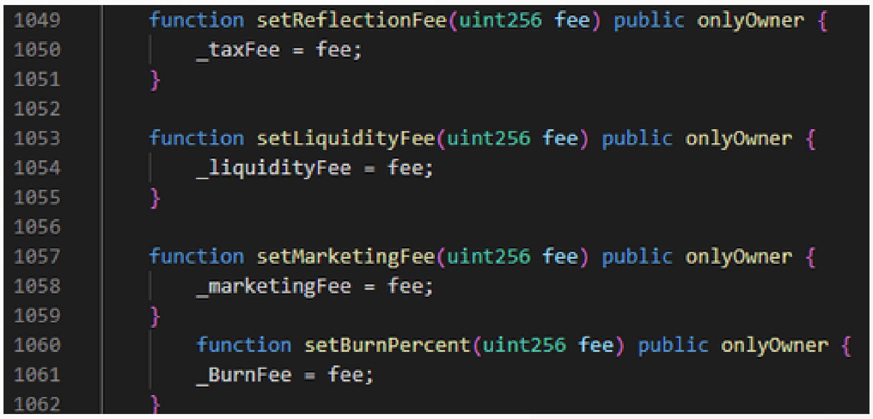
```
    function collectFee(address account, uint256 amount, uint256 rate) private returns (uint256) {

        uint256 transferAmount = amount;

        uint256 marketingFee = amount.mul(_marketingFee).div(10000);
        uint256 liquidityFee = amount.mul(_liquidityFee).div(10000);
        uint256 taxFee = amount.mul(_taxFee).div(10000);
        uint256 BurnFee = amount.mul(_BurnFee).div(10000);
        //@dev for holders distribution
        if (taxFee > 0) {
            transferAmount = transferAmount.sub(taxFee);
            _reflectionTotal = _reflectionTotal.sub(taxFee.mul(rate));
            _taxFeeTotal = _taxFeeTotal.add(taxFee);
            emit RewardsDistributed(taxFee);
        }

        //@dev Marketing fee
        if(marketingFee > 0){
            transferAmount = transferAmount.sub(marketingFee);
            _reflectionBalance[marketingAddress] = _reflectionBalance[marketingAddress].add(marketingFee.mul(rate));
            _marketingFeeTotal = _marketingFeeTotal.add(marketingFee);
            emit Transfer(account,marketingAddress,marketingFee);
        }
        //@dev Burn fee
        if(BurnFee > 0){
            transferAmount = transferAmount.sub(BurnFee);
            _reflectionBalance[BurnAddress] = _reflectionBalance[BurnAddress].add(BurnFee.mul(rate));
            _BurnFeeTotal = _BurnFeeTotal.add(BurnFee);
            emit Transfer(account,BurnAddress,BurnFee);
        }

        //@dev Liquidity fee
        if(liquidityFee > 0){
            transferAmount = transferAmount.sub(liquidityFee);
            _reflectionBalance[liquidityAddress] = _reflectionBalance[liquidityAddress].add(liquidityFee.mul(rate));
            _liquidityFeeTotal = _liquidityFeeTotal.add(liquidityFee);
            emit Transfer(account,liquidityAddress,liquidityFee);
        }

        return transferAmount;
    }
```

Recommendation

Privileged roles can be granted the stop transactions. The owner may take advantage of by setting the fees to high percantage value. The contract could check not allowing setting fees high percantage values put **require** and set reasonable amount.

Manual Review

The owner can set fees up to 100%

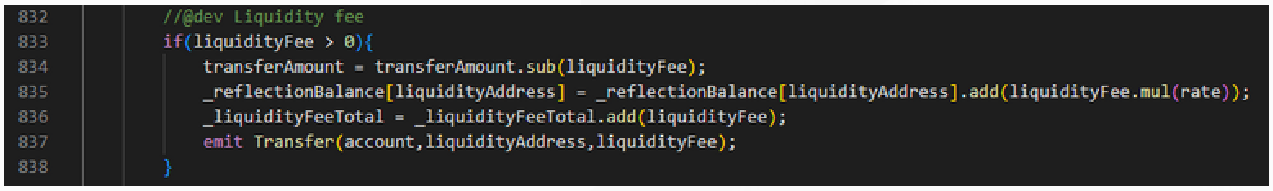


```
    function setReflectionFee(uint256 fee) public onlyOwner {
        _taxFee = fee;
    }

    function setLiquidityFee(uint256 fee) public onlyOwner {
        _liquidityFee = fee;
    }

    function setMarketingFee(uint256 fee) public onlyOwner {
        _marketingFee = fee;
    }
        function setBurnPercent(uint256 fee) public onlyOwner {
        _BurnFee = fee;
    }
```

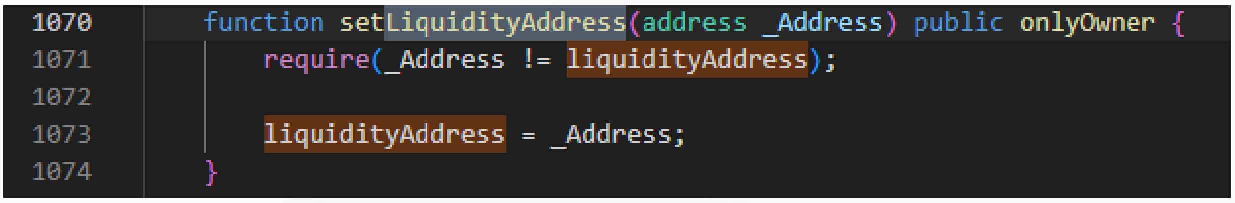
Recommendation

These functions should be provided arbitrary limits, e.g., put a **require** check that allows maximum limit etc.

Auto liquidity is going to an externally owned account



```
        //@dev Liquidity fee
        if(liquidityFee > 0){
            transferAmount = transferAmount.sub(liquidityFee);
            _reflectionBalance[liquidityAddress] = _reflectionBalance[liquidityAddress].add(liquidityFee.mul(rate));
            _liquidityFeeTotal = _liquidityFeeTotal.add(liquidityFee);
            emit Transfer(account,liquidityAddress,liquidityFee);
        }
```



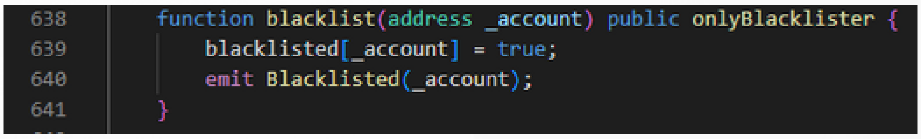
```
    function setLiquidityAddress(address _Address) public onlyOwner {
        require(_Address != liquidityAddress);

        liquidityAddress = _Address;
    }
```

Recommendation

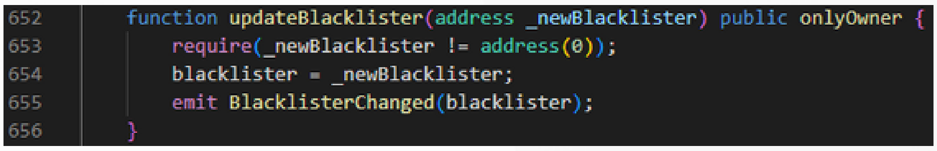
Authorizing privileged roles to externally-owned-account (EOA) is dangerous.
Send LP tokens to dead address or unreachable address.

Manual Review

The owner can set a blacklist any account.



```
function blacklist(address _account) public onlyBlacklister {
    blacklisted[_account] = true;
    emit Blacklisted(_account);
}
```

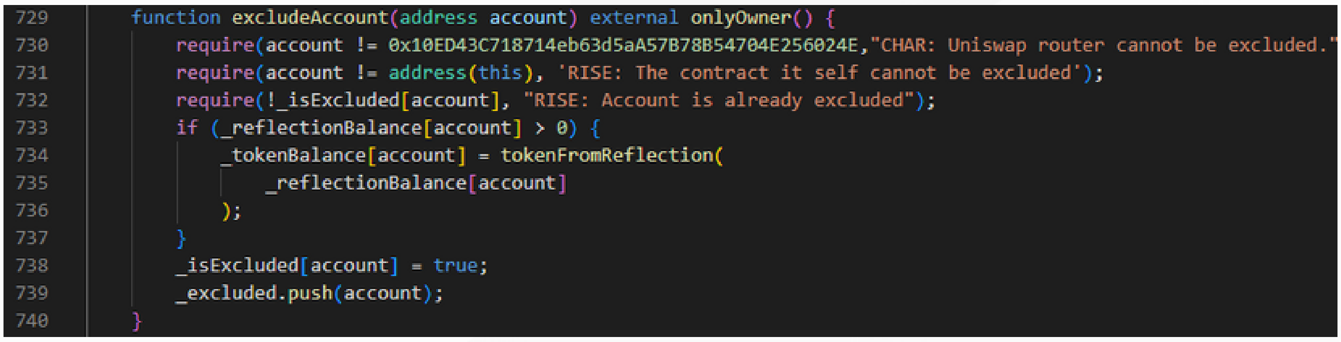


```
function updateBlacklister(address _newBlacklister) public onlyOwner {
    require(_newBlacklister != address(0));
    blacklister = _newBlacklister;
    emit BlacklisterChanged(blacklister);
}
```

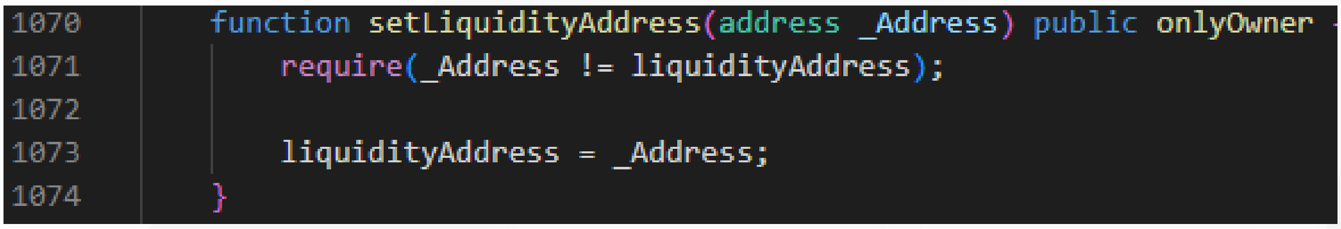
Recommendation

Authorizing privileged roles to add an account to black list and pause trade for account. These cause can affect decentralization. Remove blacklist function

The owner can exclude accounts from fees



```
function excludeAccount(address account) external onlyOwner() {
    require(account != 0x10ED43C718714eb63d5aA57B78B54704E256024E,"CHAR: Uniswap router cannot be excluded."
    require(account != address(this), 'RISE: The contract it self cannot be excluded');
    require(!_isExcluded[account], "RISE: Account is already excluded");
    if (_reflectionBalance[account] > 0) {
        _tokenBalance[account] = tokenFromReflection(
            _reflectionBalance[account]
        );
    }
    _isExcluded[account] = true;
    _excluded.push(account);
}
```



```
function setLiquidityAddress(address _Address) public onlyOwner
    require(_Address != liquidityAddress);

    liquidityAddress = _Address;
}
```

Recommendation

Authorizing privileged roles to exclude accounts from fees. These cause can affect decentralization. If owner change the liqudity address by by calling the setLiquidityAddress excludeAccount function is different from initial value of the liqudityAddress. Should expect logic and carefully check again address.

Manual Review

Public Function could be Declared External

```
setLiquidityAddress
setMarketingAddress
setBurnPercent
setMarketingFee
setLiquidityFee
setReflectionFee
IncludeFromFee
ExcludedFromFee
_burn
```

Recommendation

Public functions that are never called by the contract should be declared external to save gas. Use external attribute for functions never called from the contract.

AUDIT REPORT

SecureWise

SMART CONTRACT AUDIT

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

SecureWise