RD AUDITORS

DORUM, Smart Contract, Code Review and Security Analysis Report

Customer: DORUM
Prepared on: 29th April 2022
Platform: Binance Smart Chain
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer

This document may contain confidential information about its systems and intellectual property of the customer as well as information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or it can be disclosed publicly after all vulnerabilities are fixed - upon the decision of the customer.

Document

Name	Smart Contract Code Review and Security Analysis Report of DORUM
Platform	BSC / Solidity
File	DORUM.sol
MD5 hash	8AFACF1199DAC4D86EC4548FB37BAD01
SHA256 hash	C0D55D5A6A43C749172C92CBFBC2EF17BE8A72E89888A25C1CDC8019DB1A7234
Date	29/04/2022

Introduction

RD Auditors (Consultant) were contracted by DORUM (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contract and its code review conducted between 26th - 29th April 2022.

This contract consists of one file.

Project Scope

The scope of the project is a smart contract. We have scanned this smart contract for commonly known and more specific vulnerabilities, below are those considered (the full list includes but is not limited to):

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Byte array vulnerabilities
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Malicious libraries
- Compiler version not fixed
- Unchecked external call - Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, the customer’s solidity smart contract is now **Well-Secured.**

You are Here
Insecure
Poorly Secured
Secure
Well-Secured

Automated checks are with smartDec, Mythril, Slither and remix IDE. All issues were performed by our team, which included the analysis of code functionality, the manual audit found during automated analysis were manually reviewed and applicable vulnerabilities are presented in the audit overview section. The general overview is presented in the AS-IS section and all issues found are located in the audit overview section.

We found the following;

Total Issues	0
🟥 Critical	0
🟧 High	0
🟨 Medium	0
🟩 Low	0
🟦 Very Low	0

Code Quality

The libraries within this smart contract are part of a 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 to a specific address and its properties/methods can be reused many times by other contracts.

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

Documentation

We were given the DORUM code as a link.

https://www.bscscan.com/address/0x1B98f4B4BF431B540bc766C16a5B70e392fcc9aB#code

The hash of that file is mentioned in the table. As mentioned above, It's recommended to write comments in the smart contract code, so anyone can quickly understand the programming flow as well as complex code logic.

Comments are very helpful in understanding the overall architecture of the protocol. It also provides a clear overview of the system components, including helpful details, like the lifetime of the background script.

Use of Dependencies

As per our observation, the libraries are used in this smart contract infrastructure. Those were based on well known industry standard open source projects and even core code blocks that are written well and systematically.

AS-IS Overview

DORUM.sol

File And Function Level Report

File:	DORUM.sol
Contract:	Dorum
Inherit:	Context, IERC20, Ownable
Observation:	Passed
Test Report:	Passed

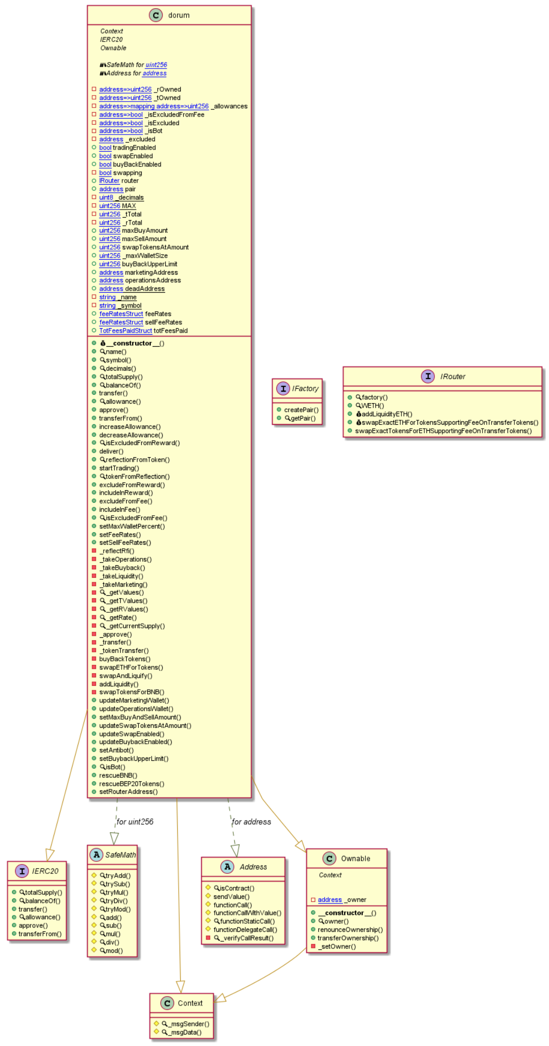
Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	name	read	Passed	All Passed	No Issue	Passed
2	symbol	read	Passed	All Passed	No Issue	Passed
3	decimals	read	Passed	All Passed	No Issue	Passed
4	totalSupply	read	Passed	All Passed	No Issue	Passed
5	balanceOf	read	Passed	All Passed	No Issue	Passed
6	transfer	write	Passed	All Passed	No Issue	Passed
7	allowance	read	Passed	All Passed	No Issue	Passed
8	approve	write	Passed	All Passed	No Issue	Passed
9	transferFrom	write	Passed	All Passed	No Issue	Passed
10	increaseAllowance	write	Passed	All Passed	No Issue	Passed
11	decreaseAllowance	write	Passed	All Passed	No Issue	Passed

12	isExcludedFromReward	read	Passed	All Passed	No Issue	Passed
13	deliver	write	Passed	All Passed	No Issue	Passed
14	reflectionFromToken	write	Passed	All Passed	No Issue	Passed
15	startTrading	write	Passed	All Passed	No Issue	Passed
16	tokenFromReflection	write	Passed	All Passed	No Issue	Passed
17	excludeFromReward	write	Passed	All Passed	No Issue	Passed
18	includeInReward	write	Passed	All Passed	No Issue	Passed
19	excludeFromFee	write	Passed	All Passed	No Issue	Passed
20	includeInFee	write	Passed	All Passed	No Issue	Passed
21	isExcludedFromFee	write	Passed	All Passed	No Issue	Passed
22	setMaxWalletPercent	write	Passed	All Passed	No Issue	Passed
23	setFeeRates	write	Passed	All Passed	No Issue	Passed
24	setSellFeeRates	write	Passed	All Passed	No Issue	Passed
25	_reflectRfi	write	Passed	All Passed	No Issue	Passed
26	_takeOperations	write	Passed	All Passed	No Issue	Passed
27	_takeBuyback	write	Passed	All Passed	No Issue	Passed
28	_takeLiquidity	write	Passed	All Passed	No Issue	Passed
29	_takeMarketing	write	Passed	All Passed	No Issue	Passed
30	_getValues	read	Passed	All Passed	No Issue	Passed
31	_getTValues	read	Passed	All Passed	No Issue	Passed
32	_getRValues	read	Passed	All Passed	No Issue	Passed

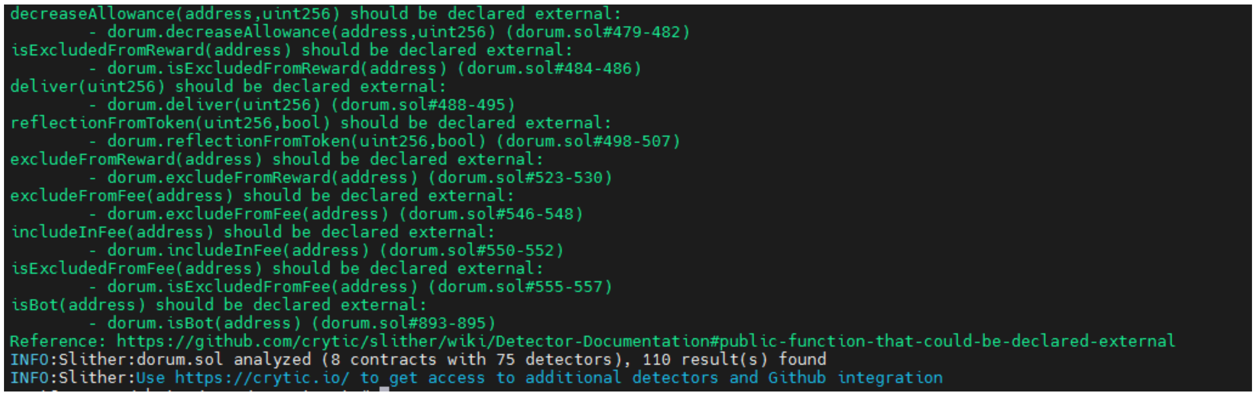
33	_getRate	read	Passed	All Passed	No Issue	Passed
34	_getCurrentSupply	read	Passed	All Passed	No Issue	Passed
35	_approve	write	Passed	All Passed	No Issue	Passed
36	_transfer	write	Passed	All Passed	No Issue	Passed
37	_tokenTransfer	write	Passed	All Passed	No Issue	Passed
38	buyBackTokens	write	Passed	All Passed	No Issue	Passed
39	swapETHForTokens	write	Passed	All Passed	No Issue	Passed
40	swapAndLiquify	write	Passed	All Passed	No Issue	Passed
41	addLiquidity	write	Passed	All Passed	No Issue	Passed
42	swapTokensForBNB	write	Passed	All Passed	No Issue	Passed
43	updateMarketingWallet	write	Passed	All Passed	No Issue	Passed
44	updateOperationsWallet	write	Passed	All Passed	No Issue	Passed
45	setMaxBuyAndSellAmount	write	Passed	All Passed	No Issue	Passed
46	updateSwapTokensAtAmount	write	Passed	All Passed	No Issue	Passed
47	updateSwapEnabled	write	Passed	All Passed	No Issue	Passed
48	updateBuybackEnabled	write	Passed	All Passed	No Issue	Passed
49	setAntibot	write	Passed	All Passed	No Issue	Passed
50	setBuybackUpperLimit	write	Passed	All Passed	No Issue	Passed
51	isBot	read	Passed	All Passed	No Issue	Passed
52	rescueBNB	write	Passed	All Passed	No Issue	Passed

53	rescueBEP20Tokens	write	Passed	All Passed	No Issue	Passed
54	setRouterAddress	write	Passed	All Passed	No Issue	Passed

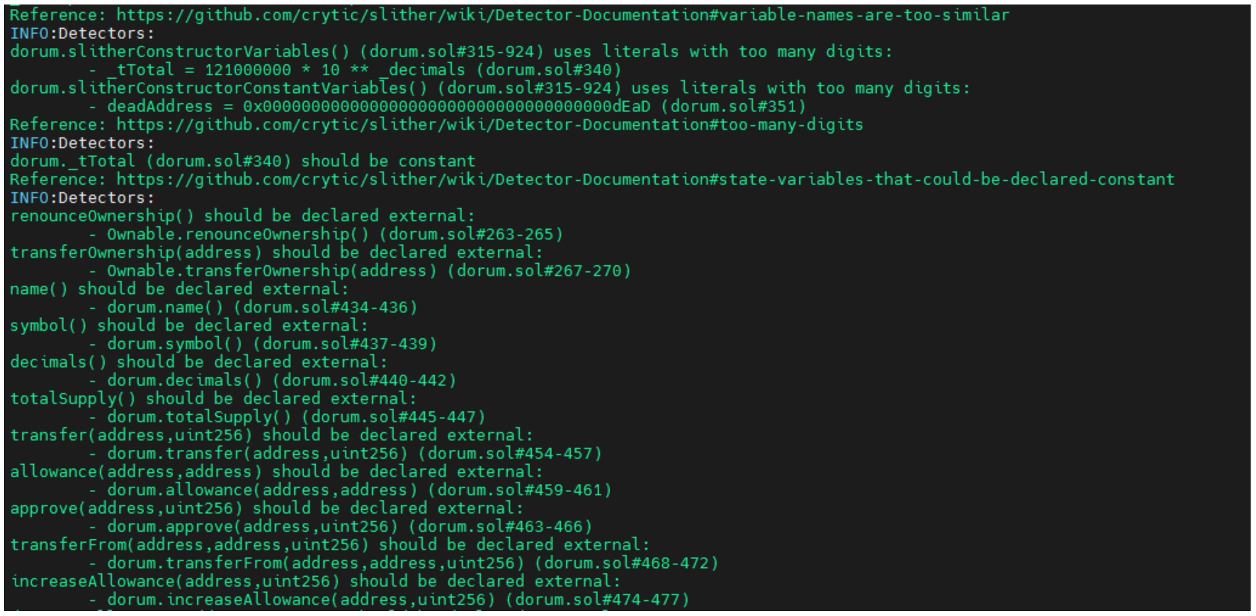
Code Flow Diagram - DORUM



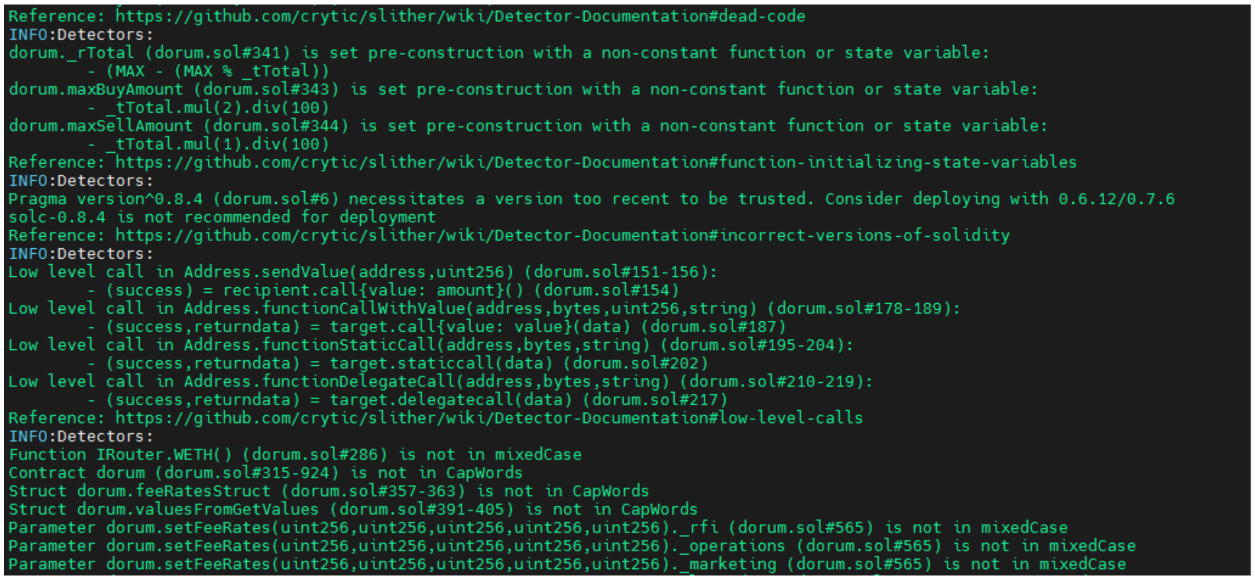
Code Flow Diagram - Slither Results Log



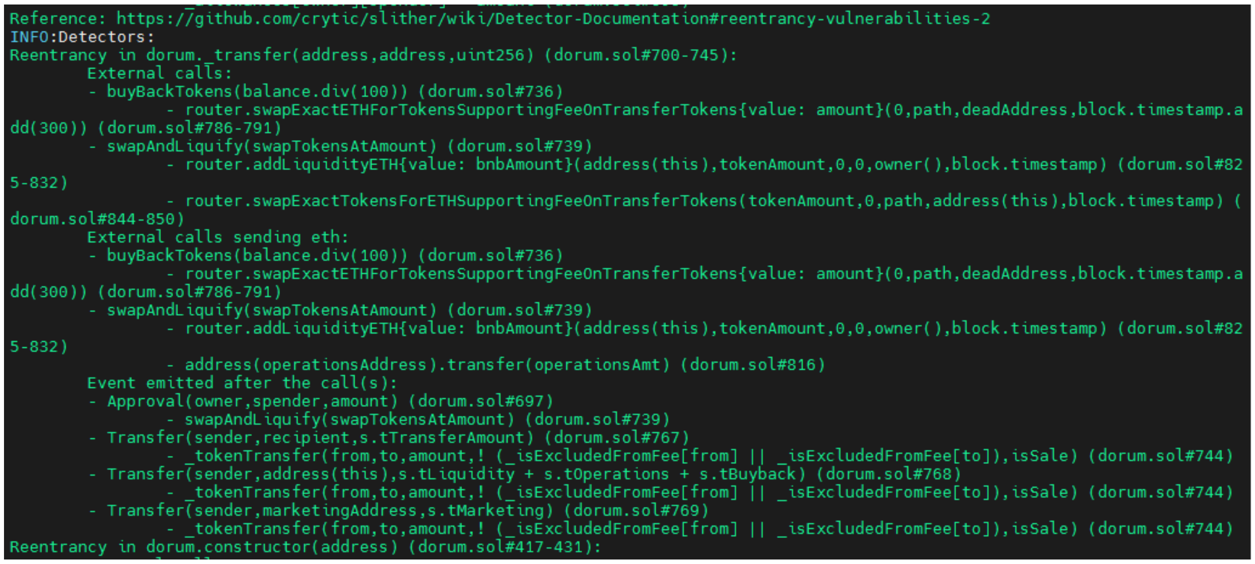
```
decreaseAllowance(address,uint256) should be declared external:
        - dorum.decreaseAllowance(address,uint256) (dorum.sol#479-482)
isExcludedFromReward(address) should be declared external:
        - dorum.isExcludedFromReward(address) (dorum.sol#484-486)
deliver(uint256) should be declared external:
        - dorum.deliver(uint256) (dorum.sol#488-495)
reflectionFromToken(uint256,bool) should be declared external:
        - dorum.reflectionFromToken(uint256,bool) (dorum.sol#498-507)
excludeFromReward(address) should be declared external:
        - dorum.excludeFromReward(address) (dorum.sol#523-530)
excludeFromFee(address) should be declared external:
        - dorum.excludeFromFee(address) (dorum.sol#546-548)
includeInFee(address) should be declared external:
        - dorum.includeInFee(address) (dorum.sol#550-552)
isExcludedFromFee(address) should be declared external:
        - dorum.isExcludedFromFee(address) (dorum.sol#555-557)
isBot(address) should be declared external:
        - dorum.isBot(address) (dorum.sol#893-895)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:dorum.sol analyzed (8 contracts with 75 detectors), 110 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```



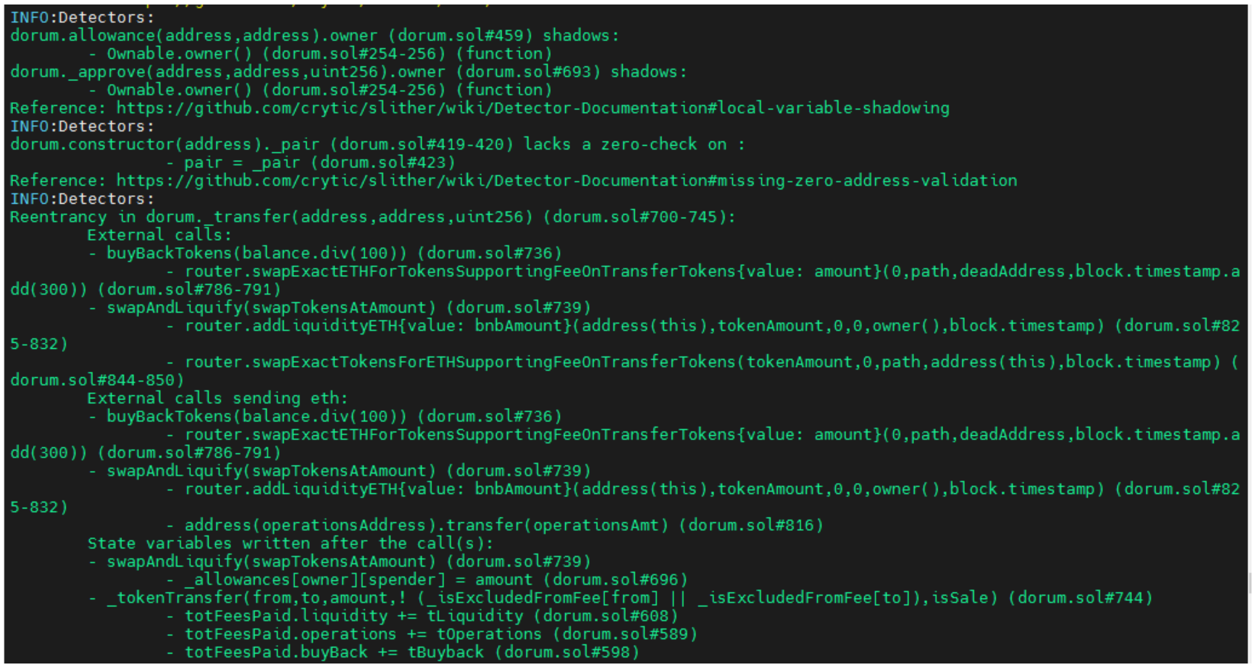
```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
dorum.slitherConstructorVariables() (dorum.sol#315-924) uses literals with too many digits:
        - _tTotal = 121000000 * 10 ** _decimals (dorum.sol#340)
dorum.slitherConstructorConstantVariables() (dorum.sol#315-924) uses literals with too many digits:
        - deadAddress = 0x000000000000000000000000000000000000dEaD (dorum.sol#351)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
dorum._tTotal (dorum.sol#340) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (dorum.sol#263-265)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (dorum.sol#267-270)
name() should be declared external:
        - dorum.name() (dorum.sol#434-436)
symbol() should be declared external:
        - dorum.symbol() (dorum.sol#437-439)
decimals() should be declared external:
        - dorum.decimals() (dorum.sol#440-442)
totalSupply() should be declared external:
        - dorum.totalSupply() (dorum.sol#445-447)
transfer(address,uint256) should be declared external:
        - dorum.transfer(address,uint256) (dorum.sol#454-457)
allowance(address,address) should be declared external:
        - dorum.allowance(address,address) (dorum.sol#459-461)
approve(address,uint256) should be declared external:
        - dorum.approve(address,uint256) (dorum.sol#463-466)
transferFrom(address,address,uint256) should be declared external:
        - dorum.transferFrom(address,address,uint256) (dorum.sol#468-472)
increaseAllowance(address,uint256) should be declared external:
        - dorum.increaseAllowance(address,uint256) (dorum.sol#474-477)
```



```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
dorum._rTotal (dorum.sol#341) is set pre-construction with a non-constant function or state variable:
        - (MAX - (MAX % _tTotal))
dorum.maxBuyAmount (dorum.sol#343) is set pre-construction with a non-constant function or state variable:
        - _tTotal.mul(2).div(100)
dorum.maxSellAmount (dorum.sol#344) is set pre-construction with a non-constant function or state variable:
        - _tTotal.mul(1).div(100)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#function-initializing-state-variables
INFO:Detectors:
Pragma version^0.8.4 (dorum.sol#6) necessitates a version too recent to be trusted. Consider deploying with 0.6.12/0.7.6
solc-0.8.4 is not recommended for deployment
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#incorrect-versions-of-solidity
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (dorum.sol#151-156):
        - (success) = recipient.call{value: amount}() (dorum.sol#154)
Low level call in Address.functionCallWithValue(address,bytes,uint256,string) (dorum.sol#178-189):
        - (success,returndata) = target.call{value: value}(data) (dorum.sol#187)
Low level call in Address.functionStaticCall(address,bytes,string) (dorum.sol#195-204):
        - (success,returndata) = target.staticcall(data) (dorum.sol#202)
Low level call in Address.functionDelegateCall(address,bytes,string) (dorum.sol#210-219):
        - (success,returndata) = target.delegatecall(data) (dorum.sol#217)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Function IRouter.WETH() (dorum.sol#286) is not in mixedCase
Contract dorum (dorum.sol#315-924) is not in CapWords
Struct dorum.feeRatesStruct (dorum.sol#357-363) is not in CapWords
Struct dorum.valuesFromGetValues (dorum.sol#391-405) is not in CapWords
Parameter dorum.setFeeRates(uint256,uint256,uint256,uint256,uint256)._rfi (dorum.sol#565) is not in mixedCase
Parameter dorum.setFeeRates(uint256,uint256,uint256,uint256,uint256)._operations (dorum.sol#565) is not in mixedCase
Parameter dorum.setFeeRates(uint256,uint256,uint256,uint256,uint256)._marketing (dorum.sol#565) is not in mixedCase
```

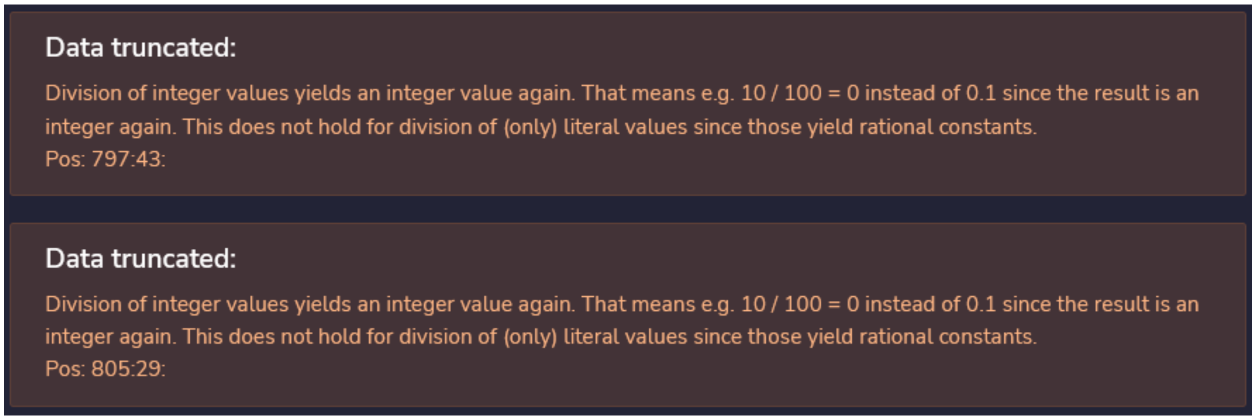


```
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-vulnerabilities-2
INFO:Detectors:
Reentrancy in dorum._transfer(address,address,uint256) (dorum.sol#700-745):
        External calls:
        - buyBackTokens(balance.div(100)) (dorum.sol#736)
                - router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.a
dd(300)) (dorum.sol#786-791)
        - swapAndLiquify(swapTokensAtAmount) (dorum.sol#739)
                - router.addLiquidityETH{value: bnbAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (dorum.sol#82
5-832)
                - router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (
dorum.sol#844-850)
        External calls sending eth:
        - buyBackTokens(balance.div(100)) (dorum.sol#736)
                - router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.a
dd(300)) (dorum.sol#786-791)
        - swapAndLiquify(swapTokensAtAmount) (dorum.sol#739)
                - router.addLiquidityETH{value: bnbAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (dorum.sol#82
5-832)
                - address(operationsAddress).transfer(operationsAmt) (dorum.sol#816)
        Event emitted after the call(s):
        - Approval(owner,spender,amount) (dorum.sol#697)
                - swapAndLiquify(swapTokensAtAmount) (dorum.sol#739)
        - Transfer(sender,recipient,s.tTransferAmount) (dorum.sol#767)
                - _tokenTransfer(from,to,amount,! (_isExcludedFromFee[from] || _isExcludedFromFee[to]),isSale) (dorum.sol#744)
        - Transfer(sender,address(this),s.tLiquidity + s.tOperations + s.tBuyback) (dorum.sol#768)
                - _tokenTransfer(from,to,amount,! (_isExcludedFromFee[from] || _isExcludedFromFee[to]),isSale) (dorum.sol#744)
        - Transfer(sender,marketingAddress,s.tMarketing) (dorum.sol#769)
                - _tokenTransfer(from,to,amount,! (_isExcludedFromFee[from] || _isExcludedFromFee[to]),isSale) (dorum.sol#744)
Reentrancy in dorum.constructor(address) (dorum.sol#417-431):
```



```
INFO:Detectors:
dorum.allowance(address,address).owner (dorum.sol#459) shadows:
        - Ownable.owner() (dorum.sol#254-256) (function)
dorum._approve(address,address,uint256).owner (dorum.sol#693) shadows:
        - Ownable.owner() (dorum.sol#254-256) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
dorum.constructor(address)._pair (dorum.sol#419-420) lacks a zero-check on :
                - pair = _pair (dorum.sol#423)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-validation
INFO:Detectors:
Reentrancy in dorum._transfer(address,address,uint256) (dorum.sol#700-745):
        External calls:
        - buyBackTokens(balance.div(100)) (dorum.sol#736)
                - router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.a
dd(300)) (dorum.sol#786-791)
        - swapAndLiquify(swapTokensAtAmount) (dorum.sol#739)
                - router.addLiquidityETH{value: bnbAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (dorum.sol#82
5-832)
                - router.swapExactTokensForETHSupportingFeeOnTransferTokens(tokenAmount,0,path,address(this),block.timestamp) (
dorum.sol#844-850)
        External calls sending eth:
        - buyBackTokens(balance.div(100)) (dorum.sol#736)
                - router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(0,path,deadAddress,block.timestamp.a
dd(300)) (dorum.sol#786-791)
        - swapAndLiquify(swapTokensAtAmount) (dorum.sol#739)
                - router.addLiquidityETH{value: bnbAmount}(address(this),tokenAmount,0,0,owner(),block.timestamp) (dorum.sol#82
5-832)
                - address(operationsAddress).transfer(operationsAmt) (dorum.sol#816)
        State variables written after the call(s):
        - swapAndLiquify(swapTokensAtAmount) (dorum.sol#739)
                - _allowances[owner][spender] = amount (dorum.sol#696)
        - _tokenTransfer(from,to,amount,! (_isExcludedFromFee[from] || _isExcludedFromFee[to]),isSale) (dorum.sol#744)
                - totFeesPaid.liquidity += tLiquidity (dorum.sol#608)
                - totFeesPaid.operations += tOperations (dorum.sol#589)
                - totFeesPaid.buyBack += tBuyback (dorum.sol#598)
```

Solidity Static Analysis

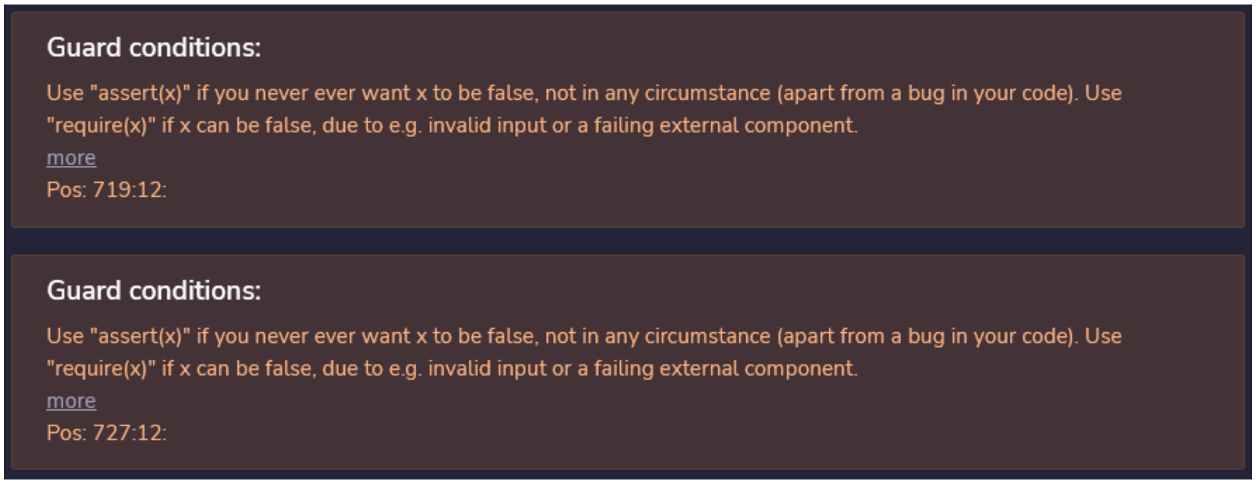


Data truncated:

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: 797:43:

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: 805:29:

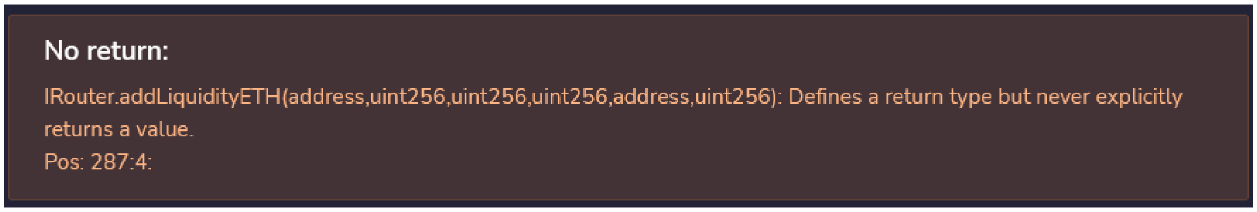


Guard conditions:

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: 719:12:

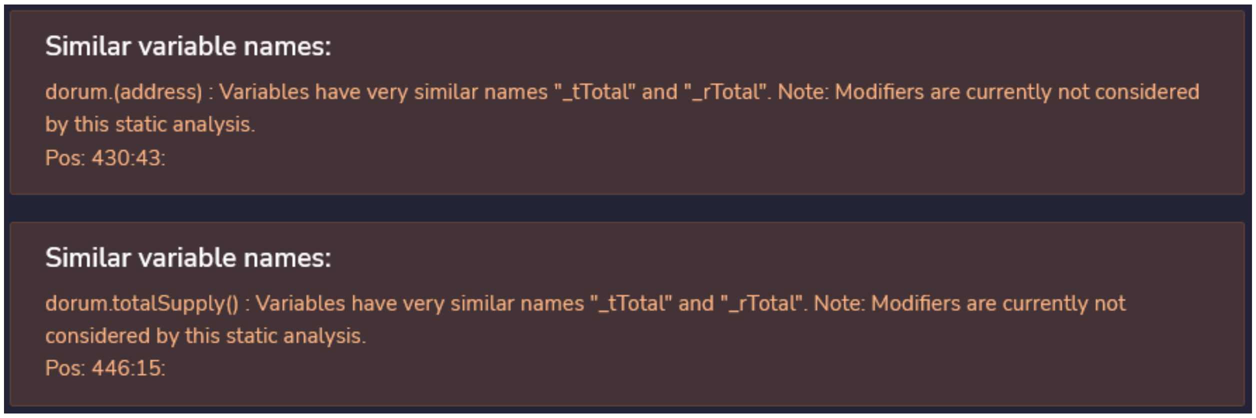
Guard conditions:

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: 727:12:



No return:

IRouter.addLiquidityETH(address,uint256,uint256,uint256,address,uint256): Defines a return type but never explicitly returns a value.
Pos: 287:4:

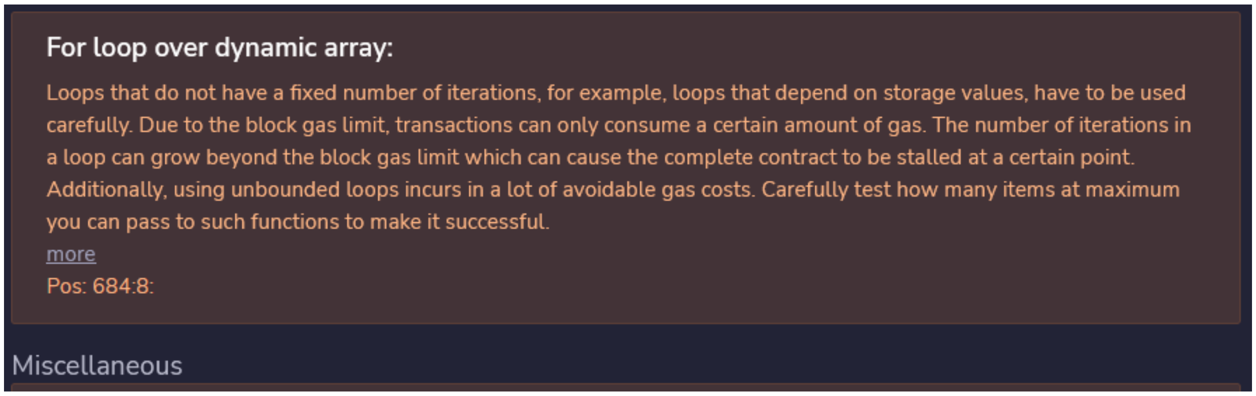


Similar variable names:

dorum.(address) : Variables have very similar names "_tTotal" and "_rTotal". Note: Modifiers are currently not considered by this static analysis.
Pos: 430:43:

Similar variable names:

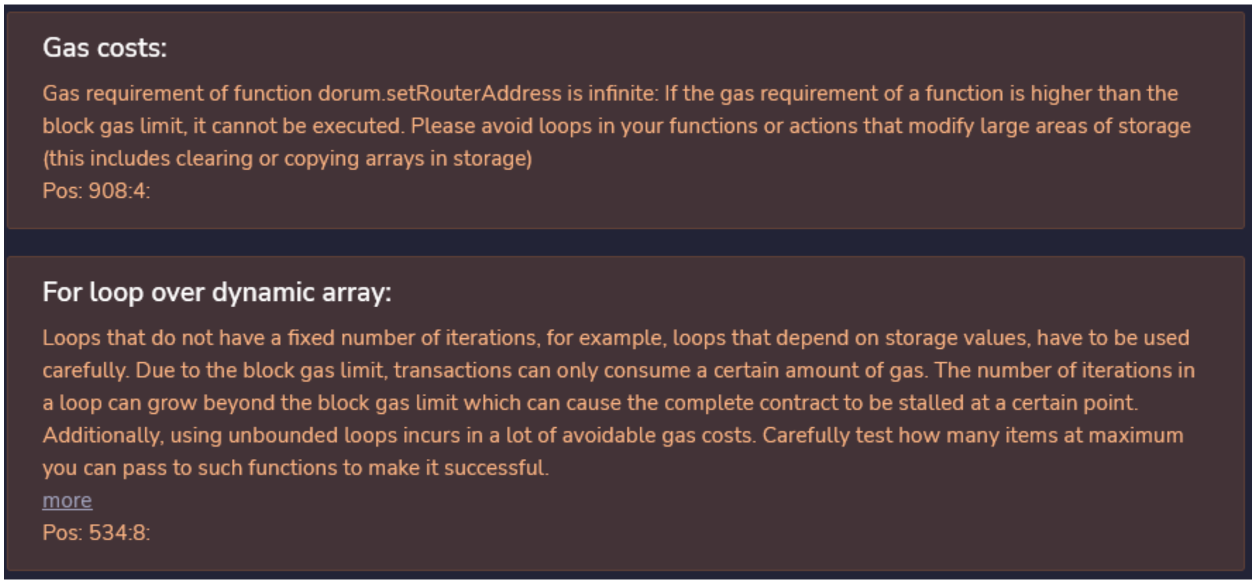
dorum.totalSupply() : Variables have very similar names "_tTotal" and "_rTotal". Note: Modifiers are currently not considered by this static analysis.
Pos: 446:15:



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: 684:8:

Miscellaneous

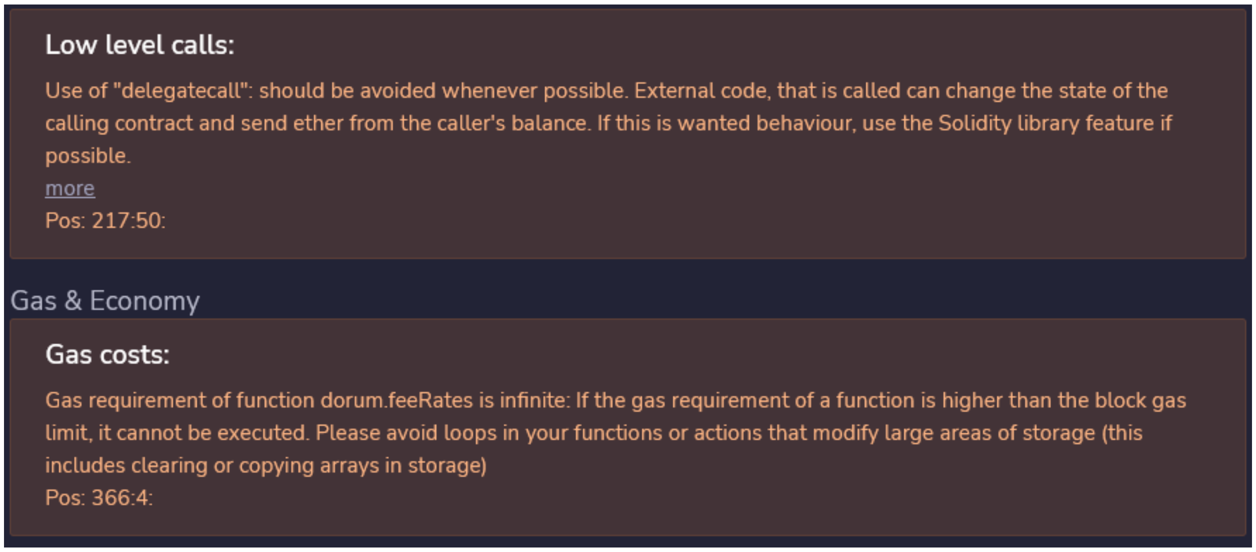


Gas costs:

Gas requirement of function dorum.setRouterAddress 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: 908:4:

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: 534:8:



Low level calls:

Use of "delegatecall": should be avoided whenever possible. External code, that is called can change the state of the calling contract and send ether from the caller's balance. If this is wanted behaviour, use the Solidity library feature if possible.
more
Pos: 217:50:

Gas & Economy

Gas costs:

Gas requirement of function dorum.feeRates 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: 366:4:

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to lost tokens etc.
High	High level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g. public access to crucial functions.
Medium	Medium level vulnerabilities are important to fix; however, they cannot lead to lost tokens.
Low	Low level vulnerabilities are most related to outdated, unused etc. These code snippets cannot have a significant impact on execution.
Lowest Code Style/ Best Practice	Lowest level vulnerabilities, code style violations and information statements cannot affect smart contract execution and can be ignored.

Audit Findings

Critical:

No critical severity vulnerabilities were found.

High:

No high severity vulnerabilities were found.

Medium:

No medium severity vulnerabilities were found.

Low:

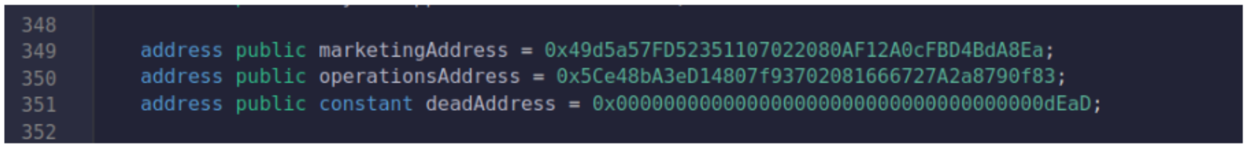
No low severity vulnerabilities were found.

Very Low:

No very low severity vulnerabilities were found.

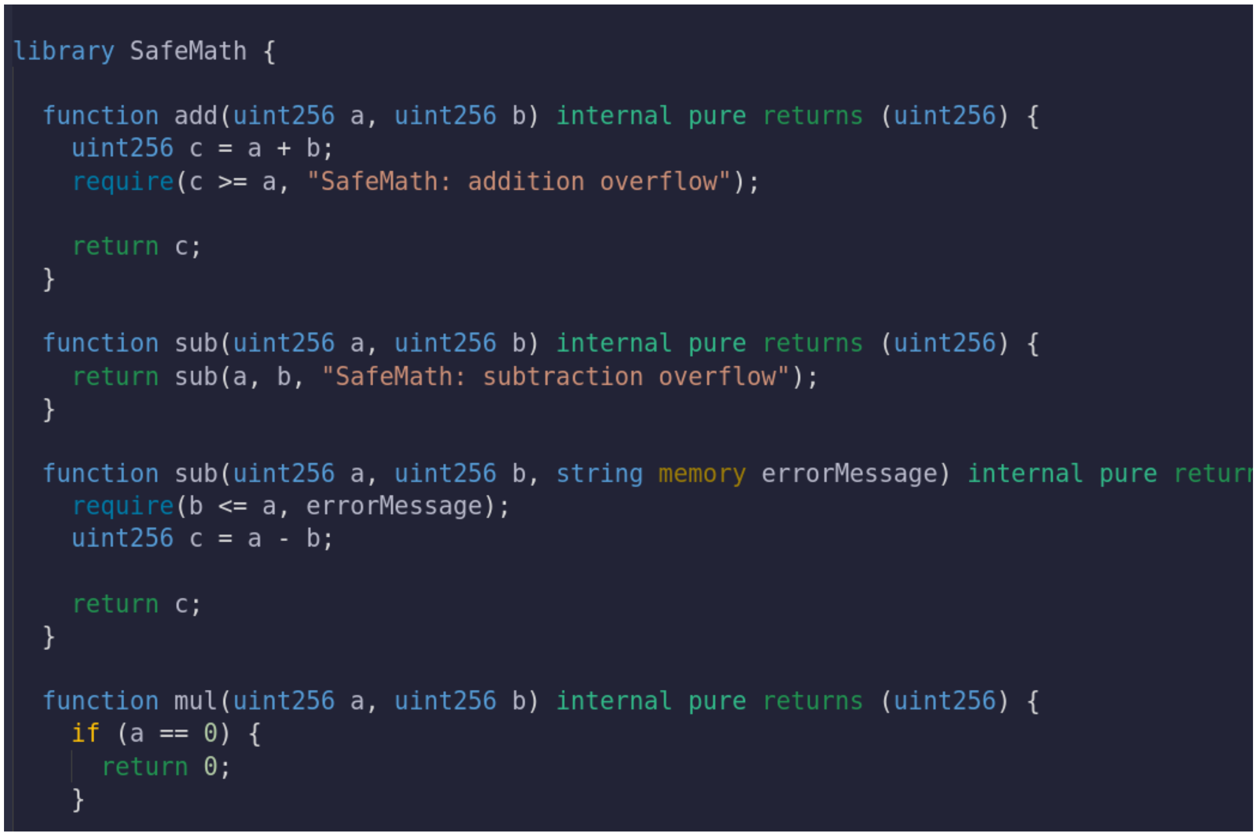
Discussion

1. Please verify these hardcoded addresses before deploying to the mainnet.



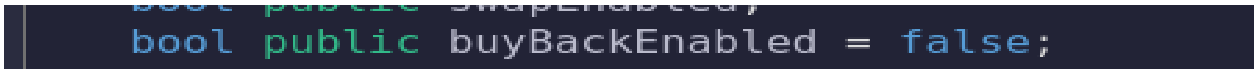
```
    address public marketingAddress = 0x49d5a57FD52351107022080AF12A0cFBD4BdA8Ea;
    address public operationsAddress = 0x5Ce48bA3eD14807f93702081666727A2a8790f83;
    address public constant deadAddress = 0x000000000000000000000000000000000000dEaD;
```

2. Solidity “pragma solidity ^0.8.0” doesn’t require safemath so you can avoid using it.



```
library SafeMath {

  function add(uint256 a, uint256 b) internal pure returns (uint256) {
    uint256 c = a + b;
    require(c >= a, "SafeMath: addition overflow");

    return c;
  }

  function sub(uint256 a, uint256 b) internal pure returns (uint256) {
    return sub(a, b, "SafeMath: subtraction overflow");
  }

  function sub(uint256 a, uint256 b, string memory errorMessage) internal pure retur
    require(b <= a, errorMessage);
    uint256 c = a - b;

    return c;
  }

  function mul(uint256 a, uint256 b) internal pure returns (uint256) {
    if (a == 0) {
      return 0;
    }
```

3. No need to assign false because by default it is false.

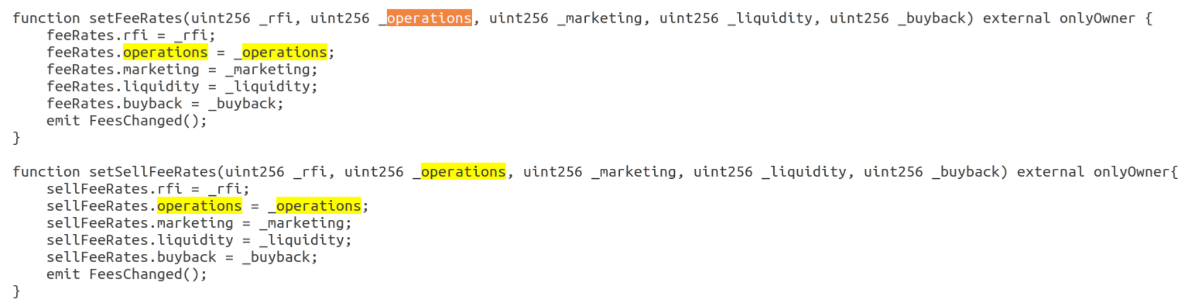


```
    bool public buyBackEnabled = false;
```

4. In this logic if you want to use floating point values like: 0.5, 0.5, 0.3 best practices.

Reference: https://ethereum.stackexchange.com/questions/55701/how-to-do-solidity-percentage-calculation

Note: whenever you change this value please keep in mind your calculation logic.


```
function setFeeRates(uint256 _rfi, uint256 _operations, uint256 _marketing, uint256 _liquidity, uint256 _buyback) external onlyOwner {
    feeRates.rfi = _rfi;
    feeRates.operations = _operations;
    feeRates.marketing = _marketing;
    feeRates.liquidity = _liquidity;
    feeRates.buyback = _buyback;
    emit FeesChanged();
}

function setSellFeeRates(uint256 _rfi, uint256 _operations, uint256 _marketing, uint256 _liquidity, uint256 _buyback) external onlyOwner{
    sellFeeRates.rfi = _rfi;
    sellFeeRates.operations = _operations;
    sellFeeRates.marketing = _marketing;
    sellFeeRates.liquidity = _liquidity;
    sellFeeRates.buyback = _buyback;
    emit FeesChanged();
}
```

Conclusion

We were given a contract file and have used all possible tests based on the given object. So it is ready to go for production. We have used all the latest static tools and manual observations to cover maximum possible test cases to scan everything.

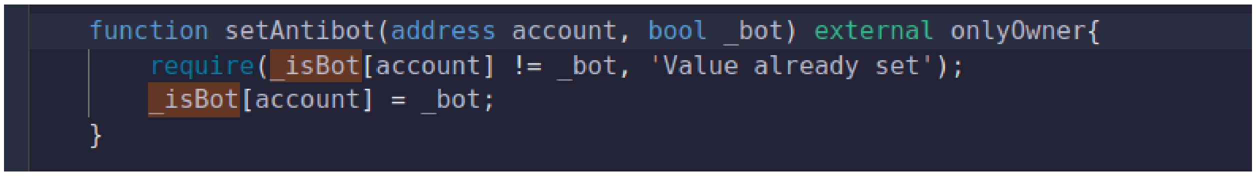
The security state of the reviewed contract is “**Well-Secured**”.

Note For Contract Users

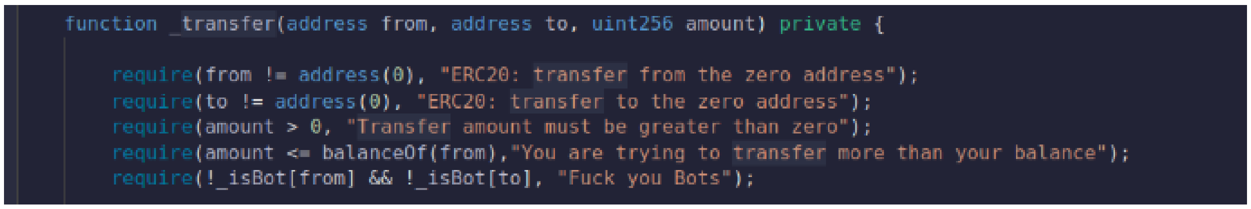
Technical auditing does not guarantee the project's ethical side.

Please do your due diligence before investing. Our audit report is never an investment advice.

The owner has privileges to ban the buying or selling of any wallet.



```
function setAntibot(address account, bool _bot) external onlyOwner{
    require(_isBot[account] != _bot, 'Value already set');
    _isBot[account] = _bot;
}
```



```
function _transfer(address from, address to, uint256 amount) private {

    require(from != address(0), "ERC20: transfer from the zero address");
    require(to != address(0), "ERC20: transfer to the zero address");
    require(amount > 0, "Transfer amount must be greater than zero");
    require(amount <= balanceOf(from),"You are trying to transfer more than your balance");
    require(!_isBot[from] && !_isBot[to], "Fuck you Bots");
```

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 analyse 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 scrutinised 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

RD Auditors Disclaimer

The smart contracts given for audit have been analysed 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).

Because the total number of test cases are unlimited, the audit makes no statements or warranties on the 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 recommend proceeding with several independent audits and a public bug bounty program to ensure security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on the blockchain. 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.

RD
AUDITORS

Email: info@rdauditors.com
Website: www.rdauditors.com