Rats Coin Smart Contract, Code Review and Security Analysis Report

Customer: Rats Coin
Prepared on: 16th March 2022
Platform: BSC
Language: Solidity

rdauditors.com

Table of Contents

Disclaimer	**2**
Document	**3**
Introduction	**4**
Project Scope	**5**
Executive Summary	**6**
Code Quality	**7**
Documentation	**8**
Use of Dependencies	**9**
AS-IS Overview	**10**
Code Flow Diagram - Rats Coin	**13**
Code Flow Diagram - Slither Results Log	**14**
Severity Definitions	**21**
Audit Findings	**22**
Discussion	**23**
Conclusion	**24**
Note For Contract Users	**25**
Our Methodology	**26**
Disclaimers	**28**

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 Rats Coin
Platform	BSC / Solidity
File	RATSCOIN.sol
MD5 hash	9B88631DA1EBD16CA604318BE6A5F02B
SHA256 hash	99337A673F27950D08D81E431C89B17129FC8BA5F968E1E86359499 5E50E4D91
Date	16/03/2022

Introduction

RD Auditors (Consultant) were contracted by Rats Coin (Customer) to conduct a Smart Contracts Code Review and Security Analysis. This report represents the findings of the security assessment of the customer`s smart contracts and its code review conducted between 14th - 16th March 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 **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

Please note within this report safeMath, Address and IERC20 are taken from the popular OpenZeppelin library.

The library within this smart contract is 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 Rats Coin team has not provided scenario and unit test scripts, which would help to determine the integrity of the code in an automated way.

Overall, the code is almost not commented. Commenting provides rich documentation for functions, return variables and more. Use of the Ethereum Natural Language Specification Format (NatSpec) for commenting is recommended.

Documentation

We were given the Rats Coin code as a link:

https://bscscan.com/address/0x57b798d2252557f13a9148a075a72816f2707356#code

The hash of that file is mentioned in the table. As mentioned above, It's recommended to write comments on 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

Rats Coin

File And Function Level Report

Contract: Context

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	_msgSender	read	Passed	All Passed	No Issue	Passed
2	_msgData	read	Passed	All Passed	No Issue	Passed

Contract: Ownable

Inherit: Context

Observation: Passed

Test Report: Passed

Sl.	Function	Type	Observation	Test Report	Conclusion	Score
1	Owner	read	Passed	All Passed	No Issue	Passed
2	renounceOwnership	onlyowner	Passed	All Passed	No Issue	Passed

Contract: Rats Coin

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	isExcluded	read	Passed	All Passed	No Issue	Passed
13	totalFees	read	Passed	All Passed	No Issue	Passed
14	reflect	write	Passed	All Passed	No Issue	Passed
15	reflectionFromToken	read	Passed	All Passed	No Issue	Passed
16	tokenFromReflection	read	Passed	All Passed	No Issue	Passed

17	excludeAccount	onlyowner	Passed	All Passed	No Issue	Passed
18	includeAccount	onlyowner	Infinite loop possibility	Owner must Excluded limited wallets	Passed with client consent	Passed with client consent
19	_approve	Private	Passed	All Passed	No Issue	Passed
20	_transfer	Private	Passed	All Passed	No Issue	Passed
21	_transferStandard	Private	Passed	All Passed	No Issue	Passed
22	_transferToExcluded	Private	Passed	All Passed	No Issue	Passed
23	_transferFromExcluded	Private	Passed	All Passed	No Issue	Passed
24	_transferBothExcluded	Private	Passed	All Passed	No Issue	Passed
25	_reflectFee	Private	Passed	All Passed	No Issue	Passed
26	_getValues	Private	Passed	All Passed	No Issue	Passed
27	_getTValues	Private	Passed	All Passed	No Issue	Passed
28	_getRValues	Private	Passed	All Passed	No Issue	Passed
29	_getRate	Private	Passed	All Passed	No Issue	Passed
30	_getCurrentSupply	Private	Passed	All Passed	No Issue	Passed

Code Flow Diagram - Rats Coin

RATSCOIN
Context
IERC20
Ownable

SafeMath for uint256
Address for address

address=>uint256 _rOwned
address=>uint256 _tOwned
address=>mapping address=>uint256 _allowances
address=>bool _isExcluded
address _excluded
uint256 MAX
uint256 _tTotal
uint256 _rTotal
uint256 _tFeeTotal
string _name
string _symbol
uint8 _decimals

__constructor__()
name()
symbol()
decimals()
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()
increaseAllowance()
decreaseAllowance()
isExcluded()
totalFees()
reflect()
reflectionFromToken()
tokenFromReflection()
excludeAccount()
includeAccount()
_approve()
_transfer()
_transferStandard()
_transferToExcluded()
_transferFromExcluded()
_transferBothExcluded()
_reflectFee()
_getValues()
_getTValues()
_getRValues()
_getRate()
_getCurrentSupply()

for uint256
for address

IERC20
totalSupply()
balanceOf()
transfer()
allowance()
approve()
transferFrom()

SafeMath
add()
sub()
mul()
div()
mod()

Address
isContract()
sendValue()
functionCall()
functionCallWithValue()
_functionCallWithValue()

Ownable
Context
address _owner
__constructor__()
owner()
renounceOwnership()
transferOwnership()

Context
_msgSender()
_msgData()

Code Flow Diagram - Slither Results Log

```
INFO:Detectors:
RATSCOIN.allowance(address,address).owner (RATSCOIN.sol#478) shadows:
        - Ownable.owner() (RATSCOIN.sol#390-392) (function)
RATSCOIN._approve(address,address,uint256).owner (RATSCOIN.sol#559) shadows:
        - Ownable.owner() (RATSCOIN.sol#390-392) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-shadowing
INFO:Detectors:
Address.isContract(address) (RATSCOIN.sol#256-265) uses assembly
        - INLINE ASM (RATSCOIN.sol#263)
Address._functionCallWithValue(address,bytes,uint256,string) (RATSCOIN.sol#349-370) uses assembly
        - INLINE ASM (RATSCOIN.sol#362-365)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#assembly-usage
INFO:Detectors:
Address._functionCallWithValue(address,bytes,uint256,string) (RATSCOIN.sol#349-370) is never used and should be removed
Address.functionCall(address,bytes) (RATSCOIN.sol#309-311) is never used and should be removed
Address.functionCall(address,bytes,string) (RATSCOIN.sol#319-321) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256) (RATSCOIN.sol#334-336) is never used and should be removed
Address.functionCallWithValue(address,bytes,uint256,string) (RATSCOIN.sol#344-347) is never used and should be removed
Address.isContract(address) (RATSCOIN.sol#256-265) is never used and should be removed
Address.sendValue(address,uint256) (RATSCOIN.sol#283-289) is never used and should be removed
Context._msgData() (RATSCOIN.sol#18-21) is never used and should be removed
SafeMath.mod(uint256,uint256) (RATSCOIN.sol#216-218) is never used and should be removed
SafeMath.mod(uint256,uint256,string) (RATSCOIN.sol#232-235) is never used and should be removed
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#dead-code
INFO:Detectors:
Low level call in Address.sendValue(address,uint256) (RATSCOIN.sol#283-289):
        - (success) = recipient.call{value: amount}() (RATSCOIN.sol#287)
Low level call in Address._functionCallWithValue(address,bytes,uint256,string) (RATSCOIN.sol#349-370):
        - (success,returndata) = target.call{value: weiValue}(data) (RATSCOIN.sol#353)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#low-level-calls
INFO:Detectors:
Constant RATSCOIN._tTotal (RATSCOIN.sol#439) is not in UPPER_CASE_WITH_UNDERSCORES
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#conformance-to-solidity-naming-conventions
INFO:Detectors:
Redundant expression "this (RATSCOIN.sol#19)" inContext (RATSCOIN.sol#13-22)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#redundant-statements
```

```
INFO:Detectors:
Variable RATSCOIN._transferToExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#593) is too similar to RATSCOIN._t
ransferStandard(address,address,uint256).tTransferAmount (RATSCOIN.sol#585)
Variable RATSCOIN._transferBothExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#611) is too similar to RATSCOIN.
_transferFromExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#602)
Variable RATSCOIN._transferFromExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#602) is too similar to RATSCOIN.
_transferFromExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#602)
Variable RATSCOIN._transferToExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#593) is too similar to RATSCOIN._t
ransferFromExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#602)
Variable RATSCOIN._getValues(uint256).rTransferAmount (RATSCOIN.sol#628) is too similar to RATSCOIN._transferStandard(address,a
ddress,uint256).tTransferAmount (RATSCOIN.sol#585)
Variable RATSCOIN._transferBothExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#611) is too similar to RATSCOIN.
_getTValues(uint256).tTransferAmount (RATSCOIN.sol#634)
Variable RATSCOIN._getValues(uint256).rTransferAmount (RATSCOIN.sol#628) is too similar to RATSCOIN._getValues(uint256).tTransf
erAmount (RATSCOIN.sol#626)
Variable RATSCOIN._getValues(uint256).rTransferAmount (RATSCOIN.sol#628) is too similar to RATSCOIN._transferBothExcluded(addre
ss,address,uint256).tTransferAmount (RATSCOIN.sol#611)
Variable RATSCOIN._transferStandard(address,address,uint256).rTransferAmount (RATSCOIN.sol#585) is too similar to RATSCOIN._tra
nsferStandard(address,address,uint256).tTransferAmount (RATSCOIN.sol#585)
Variable RATSCOIN._getRValues(uint256,uint256,uint256).rTransferAmount (RATSCOIN.sol#641) is too similar to RATSCOIN._transferS
tandard(address,address,uint256).tTransferAmount (RATSCOIN.sol#585)
Variable RATSCOIN._transferStandard(address,address,uint256).rTransferAmount (RATSCOIN.sol#585) is too similar to RATSCOIN._get
Values(uint256).tTransferAmount (RATSCOIN.sol#626)
Variable RATSCOIN._getRValues(uint256,uint256,uint256).rTransferAmount (RATSCOIN.sol#641) is too similar to RATSCOIN._getValues
(uint256).tTransferAmount (RATSCOIN.sol#626)
Variable RATSCOIN._transferToExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#593) is too similar to RATSCOIN._t
ransferBothExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#611)
Variable RATSCOIN._transferToExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#593) is too similar to RATSCOIN._g
etValues(uint256).tTransferAmount (RATSCOIN.sol#626)
Variable RATSCOIN._transferToExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#593) is too similar to RATSCOIN._g
etTValues(uint256).tTransferAmount (RATSCOIN.sol#634)
Variable RATSCOIN.reflectionFromToken(uint256,bool).rTransferAmount (RATSCOIN.sol#526) is too similar to RATSCOIN._getValues(ui
nt256).tTransferAmount (RATSCOIN.sol#626)
Variable RATSCOIN._getValues(uint256).rTransferAmount (RATSCOIN.sol#628) is too similar to RATSCOIN._transferFromExcluded(addre
ss,address,uint256).tTransferAmount (RATSCOIN.sol#602)
Variable RATSCOIN.reflectionFromToken(uint256,bool).rTransferAmount (RATSCOIN.sol#526) is too similar to RATSCOIN._transferFrom
Excluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#602)
```

```
Variable RATSCOIN._getValues(uint256).rTransferAmount (RATSCOIN.sol#628) is too similar to RATSCOIN._getTValues(uint256).tTrans
ferAmount (RATSCOIN.sol#634)
Variable RATSCOIN._transferBothExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#611) is too similar to RATSCOIN.
_transferStandard(address,address,uint256).tTransferAmount (RATSCOIN.sol#585)
Variable RATSCOIN._transferFromExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#602) is too similar to RATSCOIN.
_transferStandard(address,address,uint256).tTransferAmount (RATSCOIN.sol#585)
Variable RATSCOIN._transferBothExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#611) is too similar to RATSCOIN.
_getValues(uint256).tTransferAmount (RATSCOIN.sol#626)
Variable RATSCOIN._transferFromExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#602) is too similar to RATSCOIN.
_getValues(uint256).tTransferAmount (RATSCOIN.sol#626)
Variable RATSCOIN._transferBothExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#611) is too similar to RATSCOIN.
_transferBothExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#611)
Variable RATSCOIN.reflectionFromToken(uint256,bool).rTransferAmount (RATSCOIN.sol#526) is too similar to RATSCOIN._transferStan
dard(address,address,uint256).tTransferAmount (RATSCOIN.sol#585)
Variable RATSCOIN.reflectionFromToken(uint256,bool).rTransferAmount (RATSCOIN.sol#526) is too similar to RATSCOIN._getTValues(u
int256).tTransferAmount (RATSCOIN.sol#634)
Variable RATSCOIN._getValues(uint256).rTransferAmount (RATSCOIN.sol#628) is too similar to RATSCOIN._transferToExcluded(address
,address,uint256).tTransferAmount (RATSCOIN.sol#593)
Variable RATSCOIN._transferStandard(address,address,uint256).rTransferAmount (RATSCOIN.sol#585) is too similar to RATSCOIN._tra
nsferToExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#593)
Variable RATSCOIN._getRValues(uint256,uint256,uint256).rTransferAmount (RATSCOIN.sol#641) is too similar to RATSCOIN._transferT
oExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#593)
Variable RATSCOIN._transferStandard(address,address,uint256).rTransferAmount (RATSCOIN.sol#585) is too similar to RATSCOIN._tra
nsferBothExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#611)
Variable RATSCOIN._getRValues(uint256,uint256,uint256).rTransferAmount (RATSCOIN.sol#641) is too similar to RATSCOIN._transferB
othExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#611)
Variable RATSCOIN._transferFromExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#602) is too similar to RATSCOIN.
_getTValues(uint256).tTransferAmount (RATSCOIN.sol#634)
Variable RATSCOIN._transferStandard(address,address,uint256).rTransferAmount (RATSCOIN.sol#585) is too similar to RATSCOIN._tra
nsferFromExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#602)
Variable RATSCOIN._transferFromExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#602) is too similar to RATSCOIN.
_transferToExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#593)
Variable RATSCOIN._transferBothExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#611) is too similar to RATSCOIN.
_transferToExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#593)
Variable RATSCOIN._transferFromExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#602) is too similar to RATSCOIN.
_transferBothExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#611)
Variable RATSCOIN._getRValues(uint256,uint256,uint256).rTransferAmount (RATSCOIN.sol#641) is too similar to RATSCOIN._transferF
```

```
Variable RATSCOIN._getRValues(uint256,uint256,uint256).rTransferAmount (RATSCOIN.sol#641) is too similar to RATSCOIN._transferF
romExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#602)
Variable RATSCOIN.reflectionFromToken(uint256,bool).rTransferAmount (RATSCOIN.sol#526) is too similar to RATSCOIN._transferToEx
cluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#593)
Variable RATSCOIN.reflectionFromToken(uint256,bool).rTransferAmount (RATSCOIN.sol#526) is too similar to RATSCOIN._transferBoth
Excluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#611)
Variable RATSCOIN._transferToExcluded(address,address,uint256).rTransferAmount (RATSCOIN.sol#593) is too similar to RATSCOIN._t
ransferToExcluded(address,address,uint256).tTransferAmount (RATSCOIN.sol#593)
Variable RATSCOIN._transferStandard(address,address,uint256).rTransferAmount (RATSCOIN.sol#585) is too similar to RATSCOIN._get
TValues(uint256).tTransferAmount (RATSCOIN.sol#634)
Variable RATSCOIN._getRValues(uint256,uint256,uint256).rTransferAmount (RATSCOIN.sol#641) is too similar to RATSCOIN._getTValue
s(uint256).tTransferAmount (RATSCOIN.sol#634)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#variable-names-are-too-similar
INFO:Detectors:
RATSCOIN.slitherConstructorConstantVariables() (RATSCOIN.sol#427-662) uses literals with too many digits:
        - _tTotal = 1000000000 * 10 ** 6 (RATSCOIN.sol#439)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#too-many-digits
INFO:Detectors:
RATSCOIN._decimals (RATSCOIN.sol#445) should be constant
RATSCOIN._name (RATSCOIN.sol#443) should be constant
RATSCOIN._symbol (RATSCOIN.sol#444) should be constant
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#state-variables-that-could-be-declared-constant
INFO:Detectors:
owner() should be declared external:
        - Ownable.owner() (RATSCOIN.sol#390-392)
renounceOwnership() should be declared external:
        - Ownable.renounceOwnership() (RATSCOIN.sol#409-412)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (RATSCOIN.sol#418-422)
name() should be declared external:
        - RATSCOIN.name() (RATSCOIN.sol#452-454)
symbol() should be declared external:
        - RATSCOIN.symbol() (RATSCOIN.sol#456-458)
decimals() should be declared external:
        - RATSCOIN.decimals() (RATSCOIN.sol#460-462)
totalSupply() should be declared external:
        - RATSCOIN.totalSupply() (RATSCOIN.sol#464-466)
```

```
balanceOf(address) should be declared external:
        - RATSCOIN.balanceOf(address) (RATSCOIN.sol#468-471)
transfer(address,uint256) should be declared external:
        - RATSCOIN.transfer(address,uint256) (RATSCOIN.sol#473-476)
allowance(address,address) should be declared external:
        - RATSCOIN.allowance(address,address) (RATSCOIN.sol#478-480)
approve(address,uint256) should be declared external:
        - RATSCOIN.approve(address,uint256) (RATSCOIN.sol#482-485)
transferFrom(address,address,uint256) should be declared external:
        - RATSCOIN.transferFrom(address,address,uint256) (RATSCOIN.sol#487-491)
increaseAllowance(address,uint256) should be declared external:
        - RATSCOIN.increaseAllowance(address,uint256) (RATSCOIN.sol#493-496)
decreaseAllowance(address,uint256) should be declared external:
        - RATSCOIN.decreaseAllowance(address,uint256) (RATSCOIN.sol#498-501)
isExcluded(address) should be declared external:
        - RATSCOIN.isExcluded(address) (RATSCOIN.sol#503-505)
totalFees() should be declared external:
        - RATSCOIN.totalFees() (RATSCOIN.sol#507-509)
reflect(uint256) should be declared external:
        - RATSCOIN.reflect(uint256) (RATSCOIN.sol#511-518)
reflectionFromToken(uint256,bool) should be declared external:
        - RATSCOIN.reflectionFromToken(uint256,bool) (RATSCOIN.sol#520-529)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:RATSCOIN.sol analyzed (6 contracts with 75 detectors), 83 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```

Solidity Static Analysis

Security

Check-effects-interaction:

Potential violation of Checks-Effects-Interaction pattern in Address._functionCallWithValue(address,bytes,uint256,string): Could potentially lead to re-entrancy vulnerability. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 349:4:

Low level calls:

Use of "call": should be avoided whenever possible. It can lead to unexpected behavior if return value is not handled properly. Please use Direct Calls via specifying the called contract's interface.
more
Pos: 353:50:

Gas & Economy

Gas costs:

Gas requirement of function RATSCOIN.transferOwnership 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: 418:4:

Gas costs:

Gas requirement of function RATSCOIN.approve 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: 482:4:

Gas costs:

Gas requirement of function RATSCOIN.transferFrom 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: 487:4:

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: 548:8:

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: 653:8:

Miscellaneous

Constant/View/Pure functions:

SafeMath.sub(uint256,uint256) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 123:4:

Constant/View/Pure functions:

RATSCOIN.reflectionFromToken(uint256,bool) : Is constant but potentially should not be. Note: Modifiers are currently not considered by this static analysis.
more
Pos: 520:4:

Similar variable names:

RATSCOIN.() : Variables have very similar names "_rOwned" and "_tOwned". Note: Modifiers are currently not considered by this static analysis.
Pos: 448:8:

Similar variable names:

RATSCOIN.() : Variables have very similar names "_tTotal" and "_rTotal". Note: Modifiers are currently not considered by this static analysis.
Pos: 448:32:

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: 532:8:

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: 538:8:

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) Infinite loop possibility

```
650    function _getCurrentSupply() private view returns(uint256, uint256) {
651        uint256 rSupply = _rTotal;
652        uint256 tSupply = _tTotal;
653        for (uint256 i = 0; i < _excluded.length; i++) {
654            if (_rOwned[_excluded[i]] > rSupply || _tOwned[_excluded[i]] > tSupply) return (_rTotal, _tTotal);
655            rSupply = rSupply.sub(_rOwned[_excluded[i]]);
656            tSupply = tSupply.sub(_tOwned[_excluded[i]]);
657        }
```

```
546    function includeAccount(address account) external onlyOwner() {
547        require(_isExcluded[account], "Account is already excluded");
548        for (uint256 i = 0; i < _excluded.length; i++) {
549            if (_excluded[i] == account) {
550                _excluded[i] = _excluded[_excluded.length - 1];
551                _tOwned[account] = 0;
552                _isExcluded[account] = false;
553                _excluded.pop();
554                break;
555            }
556        }
557    }
```

If the _excluded array length is increased, it might hit the gas limit. Another function also would be affected with this is: _getCurrentSupply but it is a view function so it will not create any serious impact.

Solution

Please keep the excluded wallet to a minimum, ideally under 100 wallets.

(2) The address library was not used anywhere. So, it's better to remove it if not used to make code clean.

Conclusion

We were given a contract file and have used all possible tests based on the given object. The contract is written systematically, so it is good 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 now “well-secured”.

Note For Contract Users

There are several owner only functions. Those can be called by the owner's wallet only. So, if the owner's wallet is compromised, then it carries the risk of the contract becoming vulnerable to unexpected fate.

- transferOwnership: Owner can transfer owner to another wallet
- renounceOwnership: Owner can resign and renounce ownership. After this function is called, no other owner functions can be called.
- excludeAccount: Excludes accounts from reflection.
- includeAccount: Includes accounts from reflection.

To make this smart contract fully decentralized, it is recommended to renounce ownership, or send the ownership to a governance smart contract.

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.

Email: info@rdauditors.com

Website: www.rdauditors.com