



RugFreeCoins Audit



ROTTOCOIN Staking Contract Smart Contract Security Audit

October 17th, 2022

Contents

Audit details	1
Disclaimer	2
Background	3
Roadmap	4
Contract details	6
Contract code function details	7
Contract description table	9
Security issue checking status	13
Owner privileges	18
Audit conclusion	19

Audit details



Audited project
ROTTCOIN Staking Contract



Contract Address
0x0a7A17f5FE82d6308F6D7aa7AaCf11640444B8bd



Client contact
ROTTCOIN Team



Blockchain
Binance smart chain



Project website
<https://rottcoin.com/>

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

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 Rugfreecoins and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Rugfreecoins) owe no duty of care towards you or any other person, nor does Rugfreecoins 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 Rugfreecoins 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, Rugfreecoins hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Rugfreecoins, 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. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

Background

Rugfreecoins was commissioned by the ROTTCOIN Team to perform an audit of the smart contract.

<https://bscscan.com/address/0x0a7a17f5fe82d6308f6d7aa7aacf11640444b8bd>

The focus of this audit is to verify that the smart contract is secure, resilient, and working according to the specifications.

The information in this report should be used to understand the risk exposure of the smart contract, project feasibility, and long-term sustainability, and as a guide to improving the smart contract's security posture by remediating the identified issues.

.

Roadmap

2022 Q3

- Team recruitment
- Build a social channel
- Website Launch
- Community growth and development
- Whitepaper
- Audit & KYC - Safu
- Pinksale Presale
- Marketing Campaign
- Pancake swap launch

2022 Q4

- Influencer Marketing Push
- 5.000+ Holders
- CoinGecko, CoinmarketCap Listing
- Community events Dextools & Poocoin Banner Ads
- NFT Marketplace
- RottWallet

2023 Q1

- Professional marketing campaign
- Articles on influential sites
- Influencers on Twitter, Youtube, Tiktok
- AMA with large crypto community
- NFTs Collection Release
- CEX Exchanges Listing
- RottSwap launch
- Rottchain Release more to come

Tokenomics

- **APR:** 100%
- **Lock periods:** 7 days or 15 days
- **Early withdrawal penalty:** 15% fee

Contract details

Token contract details for 17th of October 2022

Contract address	0x0a7A17f5FE82d6308F6D7aa7AaCf11640444B8bd
Development wallet	0x639f72d9ce1f2bcc2024eab4ba99fa3b98c430af
Contract deployer address	0x211FE4807B1a076Fb93aeF7DB0B00A4413306601
Contract's current owner address	0x211fe4807b1a076fb93aef7db0b00a4413306601

Contract code function details








No	Category	Item	Result
1	Coding conventions	BRC20 Token standards	pass
		compile errors	pass
		Compiler version security	pass
		visibility specifiers	pass
		Gas consumption	pass
		SafeMath features	pass
		Fallback usage	pass
		tx.origin usage	pass
		deprecated items	pass
		Redundant code	pass
		Overriding variables	pass
2	Function call audit	Authorization of function call	pass
		Low level function (call/delegate call) security	pass
		Returned value security	pass
		Self-destruct function security	pass
3	Business security	Access control of owners	
		Business logics	pass
		Business implementations	pass
4	Integer overflow/underflow		pass
5	Reentrancy		pass
6	Exceptional reachable state		pass
7	Transaction ordering dependence		pass
8	Block properties dependence		pass
9	Pseudo random number generator (PRNG)		pass
10	DoS (Denial of Service)		pass
11	Token vesting implementation		pass
12	Fake deposit		pass














13	Event security		pass
----	----------------	--	------

Contract description table



The below table represents the summary of the contracts and methods in the token contract. We scanned the whole contract and listed down all the Interfaces, functions, and implementations with their visibility and mutability.

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
SafeMath	Library			
L	add	Internal 		
L	sub	Internal 		
L	sub	Internal 		
L	mul	Internal 		
L	div	Internal 		
L	div	Internal 		
L	mod	Internal 		
L	mod	Internal 		
EnumerableSet	Library			
L	_add	Private 		
L	_remove	Private 		
L	_contains	Private 		
L	_length	Private 		
L	_at	Private 		

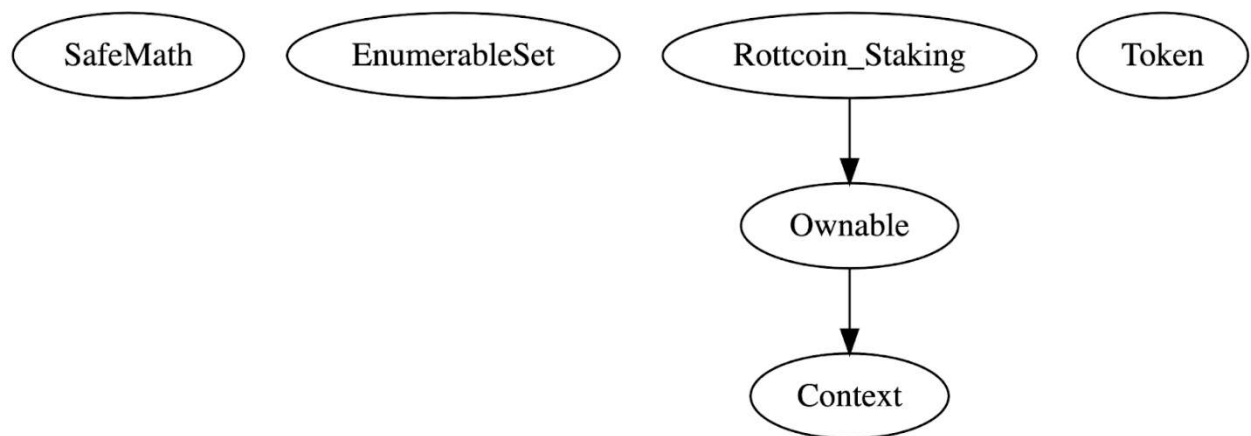
L	add	Internal 🔒		
L	remove	Internal 🔒		
L	contains	Internal 🔒		
L	length	Internal 🔒		
L	at	Internal 🔒		
L	add	Internal 🔒		
L	remove	Internal 🔒		
L	contains	Internal 🔒		
L	length	Internal 🔒		
L	at	Internal 🔒		
L	add	Internal 🔒		
L	remove	Internal 🔒		
L	contains	Internal 🔒		
L	length	Internal 🔒		
L	at	Internal 🔒		
Context	Implementation			
L		Public !		NO!
L	_msgSender	Internal 🔒		
L	_msgData	Internal 🔒		

Ownable	Implementation	Context		
L		Public !		NO!
L	owner	Public !		NO!
L	renounceOwnership	Public !		onlyOwner
L	transferOwnership	Public !		onlyOwner
Token	Interface			
L	transferFrom	External !		NO!
L	transfer	External !		NO!
L	balanceOf	External !		NO!
Rottcoin_Staking	Implementation	Ownable		
L	updateAccount	Private 		
L	getPendingDivs	Public !		NO!
L	getNumberOfHolders	Public !		NO!
L	deposit	Public !		NO!
L	withdraw	Public !		NO!
L	claimDivs	Public !		NO!
L	setDevaddress	Public !		onlyOwner
L	transferAnyERC20Tokens	Public !		onlyOwner

Legend

Symbol	Meaning
	Function can modify state
	Function is payable

Inheritance Hierarchy



Security issue checking status

❖ High severity issues

No validation when depositing tokens

Informed and Fixed

```
ftrace | funcSig
function deposit(uint256 amountToStake↑, uint256 _timeperiod↑) public {
    Token(RottToken).transferFrom(msg.sender, address(this), amountToStake↑);
    timeperiod[msg.sender] = _timeperiod↑;
    updateAccount(msg.sender);
    stakingTime[msg.sender] = block.timestamp;
    depositedTokens[msg.sender] = depositedTokens[msg.sender].add(
        amountToStake↑
    );
    totalstaked = totalstaked.add(amountToStake↑);
    if (!holders.contains(msg.sender)) {
        holders.add(msg.sender);
    }
}
```

As per the code there are only two staking times, 7 days and 15 days but in deposit function doesn't have a validation to validate _timeperiod and because the way they handle the withdraw function if someone stakes tokens other than 7 days or 15 days they cannot unstake their tokens again and it will keep staked forever.

```

if (timeperiod[msg.sender] == 7) {
    if (_lastClaimedTime >= MinimumWithdrawTime) {
        require(
            Token(RottToken).transfer(msg.sender, amountToWithdraw↑),
            "Could not transfer tokens."
        );
    }

    if (_lastClaimedTime < MinimumWithdrawTime) {
        uint256 WithdrawFee = amountToWithdraw↑.div(1e2).mul(15);
        uint256 amountAfterFee = amountToWithdraw↑.sub(WithdrawFee);
        require(
            Token(RottToken).transfer(msg.sender, amountAfterFee),
            "Could not transfer tokens."
        );
        require(
            Token(RottToken).transfer(devAddress, WithdrawFee),
            "Could not transfer tokens."
        );
    }
}

if (timeperiod[msg.sender] == 15) {
    if (_lastClaimedTime >= MinimumWithdrawTime2) {
        require(
            Token(RottToken).transfer(msg.sender, amountToWithdraw↑),
            "Could not transfer tokens."
        );
    }

    if (_lastClaimedTime < MinimumWithdrawTime2) {
        uint256 WithdrawFee = amountToWithdraw↑.div(1e2).mul(15);
        uint256 amountAfterFee = amountToWithdraw↑.sub(WithdrawFee);
        require(
            Token(RottToken).transfer(msg.sender, amountAfterFee),
            "Could not transfer tokens."
        );
        require(
            Token(RottToken).transfer(devAddress, WithdrawFee),
            "Could not transfer tokens."
        );
    }
}
}

```

Possibility of Reentrant Attack

Informed and Fixed

```
fttrace | funcSig
function withdraw(uint256 amountToWithdraw↑) public {
    require(
        depositedTokens[msg.sender] >= amountToWithdraw↑,
        "Invalid amount to withdraw"
    );

    uint256 _lastClaimedTime = block.timestamp.sub(stakingTime[msg.sender]);
    require(
        _lastClaimedTime > cliffTime,
        "You recently staked, please wait before withdrawing."
    );

    if (timeperiod[msg.sender] == 7) {
        if (_lastClaimedTime >= MinimumWithdrawTime) {
            require(
                Token(RottToken).transfer(msg.sender, amountToWithdraw↑),
                "Could not transfer tokens."
            );
        }

        if (_lastClaimedTime < MinimumWithdrawTime) {
            uint256 WithdrawFee = amountToWithdraw↑.div(1e2).mul(15);
            uint256 amountAfterFee = amountToWithdraw↑.sub(WithdrawFee);
            require(
                Token(RottToken).transfer(msg.sender, amountAfterFee),
                "Could not transfer tokens."
            );
            require(
                Token(RottToken).transfer(devAddress, WithdrawFee),
                "Could not transfer tokens."
            );
        }
    }

    if (timeperiod[msg.sender] == 15) {
        if (_lastClaimedTime >= MinimumWithdrawTime2) {
            require(
                Token(RottToken).transfer(msg.sender, amountToWithdraw↑),
                "Could not transfer tokens."
            );
        }

        if (_lastClaimedTime < MinimumWithdrawTime2) {
            uint256 WithdrawFee = amountToWithdraw↑.div(1e2).mul(15);
            uint256 amountAfterFee = amountToWithdraw↑.sub(WithdrawFee);
            require(
                Token(RottToken).transfer(msg.sender, amountAfterFee),
                "Could not transfer tokens."
            );
            require(
                Token(RottToken).transfer(devAddress, WithdrawFee),
                "Could not transfer tokens."
            );
        }
    }

    updateAccount(msg.sender);

    depositedTokens[msg.sender] = depositedTokens[msg.sender].sub(
        amountToWithdraw↑
    );
    totalstaked = totalstaked.sub(amountToWithdraw↑);
    if (holders.contains(msg.sender) && depositedTokens[msg.sender] == 0) {
        holders.remove(msg.sender);
    }
}
```

In withdraw function withdrawal amount will reduce after sending the tokens so using the Reentrant attack hacker can withdraw staked token multiple times, so should do the calculation before the transfer function or should use Reentrant Guard.

```

ftrace | funcSig
function updateAccount(address account↑) private {
    uint256 pendingDivs = getPendingDivs(account↑);
    uint256 conbalance = Token(RottToken).balanceOf(address(this));
    uint256 sur = conbalance.sub(totalstaked);
    if (sur >= pendingDivs) {
        if (pendingDivs != 0) {
            Token(RottToken).transfer(account↑, pendingDivs);
            totalEarnedTokens[account↑] = totalEarnedTokens[account↑].add(
                pendingDivs
            );
            totalClaimedRewards = totalClaimedRewards.add(pendingDivs);
            emit RewardsTransferred(account↑, pendingDivs);
        }
    }
    lastClaimedTime[account↑] = block.timestamp;
}

```

Reentrant Attacks can also happen with updateAccount function also, please refer to the recommendation to withdraw function

❖ Medium severity issues

No medium severity issues found

❖ Low severity issues

Unnecceary variable

Informed and Fixed

```

uint256 _lastClaimedTime = block.timestamp.sub(stakingTime[msg.sender]);
require(
    _lastClaimedTime > cliffTime,
    "You recently staked, please wait before withdrawing."
);
if (timeperiod[msg.sender] == 7) {

```

The clifftime variable value is set to 0 by default and it cannot change, since it has a 0 value no need for that variable, and require function.

❖ Centralization Risk

Owner can change the reward interval value without any limitation, users will get a very low amount of reward if the owner changes that reward interval to a very high value, since the owner can change the reward rate no need to change the reward interval and can remove this.

Informed and Fixed

```
// function to allow admin to set reward interval..  
ftrace | funcSig  
function setRewardInterval(uint256 _rewardInterval↑) public onlyOwner {  
    rewardInterval = _rewardInterval↑;  
}
```

Owner can change the minimum withdrawal time of 7 days staking

Owner can change this value up to 30 days, so if someone stakes tokens for 7 days and if the owner changes it to 30 days then he cannot unstake tokens before 30 days without penalty.

Informed and Fixed

```
// function to allow admin to set minimum withdraw time..  
ftrace | funcSig  
function setMinimumWithdrawTime(uint256 _sec↑) public onlyOwner {  
    require(_sec↑ <= 30 days);  
    MinimumWithdrawTime = _sec↑;  
}
```

Owner can change reward rate without any limitation

Informed and Fixed

The owner can change reward rate without any limitation and can set this value to 0 if the owner set this to 0 no one will receive any rewards.

```
// function to allow admin to set reward rate..  
ftrace | funcSig  
function setRewardRate(uint256 _rewardRate↑) public onlyOwner {  
    rewardRate = _rewardRate↑;  
}
```

Owner privileges

- ❖ The Owner can change the dev address

```
// function to allow admin to set dev address..
ftrace | funcSig
function setDevaddress(address _devAadd↑) public onlyOwner {
    devAddress = _devAadd↑;
}
```

- ❖ The owner can get any BEP20 tokens from the contract (cannot get native tokens)

```
ftrace | funcSig
function transferAnyERC20Tokens(
    address _tokenAddress↑,
    address _to↑,
    uint256 _amount↑
) public onlyOwner {
    require(
        _tokenAddress↑ != RottToken,
        "You can't transfer ROTTCOIN token."
    );

    Token(_tokenAddress↑).transfer(_to↑, _amount↑);
}
```

- ❖ The owner can transfer and renounce ownership

```
ftrace | funcSig
function renounceOwnership() public virtual onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
    _owner = address(0);
}

ftrace | funcSig
function transferOwnership(address newOwner↑) public virtual onlyOwner {
    require(
        newOwner↑ != address(0),
        "Ownable: new owner is the zero address"
    );
    emit OwnershipTransferred(_owner, newOwner↑);
    _owner = newOwner↑;
}
```

Audit conclusion

RugFreeCoins team has performed in-depth testings, line-by-line manual code review, and automated audit of the smart contract. The smart contract was analyzed mainly for common smart contract vulnerabilities, exploits, manipulations, and hacks. According to the smart contract audit.

Smart contract functional Status: **PASS**

Number of risk issues: **0**

Solidity code functional issue level: **PASS**

Number of owner privileges: **3**

Centralization risk correlated to the active owner: **LOW**

Smart contract active ownership: **YES**