RugFreeCoins Audit

Pepe 2.0 Token
Smart Contract Security Audit

July 20th ,2023

Overview

✅ No mint function found, the owner cannot mint tokens after initial deployment.

✅ The owner can’t set a max transaction limit

✅ The owner can’t pause trading once it’s enabled

❌ The owner must enable trade for the holders, if trading remains disabled, no one would be able to buy and sell.

✅ The owner can't change fees over 20%.

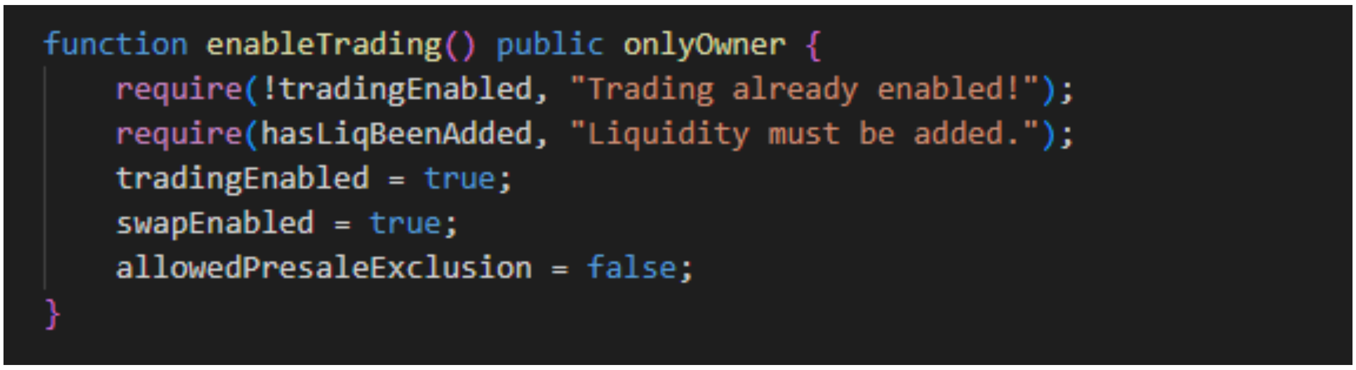
✅ The owner can't blacklist wallets.

✅ The owner can’t set a max wallet limit

✅ The owner can’t claim the contract’s balance of its own token.

- **High severity issues**

The owner must enable trade for the holders, if trading remains disabled, no one would be able to buy and sell.



```
function enableTrading() public onlyOwner {
    require(!tradingEnabled, "Trading already enabled!");
    require(hasLiqBeenAdded, "Liquidity must be added.");
    tradingEnabled = true;
    swapEnabled = true;
    allowedPresaleExclusion = false;
}
```

Contents

Overview ii

Audit details 1

Disclaimer 2

Background 3

Target market and the concept 5

Potential to grow with score points 6

Total Points 6

Contract details 7

Contract code function details 8

Contract description table 10

Security issue checking status 16

Owner privileges 18

Audit conclusion 22

Audit details

Audited project
Pepe 2.0 Token

Contract Address
0x79a94bccc0e4d2f76fdf6a33084e669e174a351c

Client contact
Pepe 2.0 Token Team

Blockchain
Binance Smart chain

Project website
https://pepe20.co/

Disclaimer

Background

Rugfreecoins was commissioned by the Pepe 2.0 Token Team to perform an audit of the smart contract.

https://bscscan.com/address/0x79a94bccc0e4d2f76fdf6a33084e669e174a351c#code

This audit focuses on verifying 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.

Tokenomics

3% tax when buying & selling (20/07/2023)

- 3% of trade goes to the dev wallet in BNB

Target market and the concept

Target market

- Anyone who’s interested in the Crypto space with long-term investment plans.
- Anyone who’s ready to earn a passive income by holding tokens.
- Anyone who’s interested in trading tokens.
- Anyone who’s interested in taking part in the Pepe 2.0 token ecosystem.
- Anyone who’s interested in taking part in the future plans of Pepe 2.0 Token.
- Anyone who’s interested in making financial transactions with any other party using Pepe 2.0 Token as the currency.

Potential to grow with score points

1.	Project efficiency	8/10
2.	Project uniqueness	7/10
3	Information quality	8/10
4	Service quality	8/10
5	System quality	8/10
6	Impact on the community	8/10
7	Impact on the business	9/10
8	Preparing for the future	8/10
9	Smart contract security	9/10
10	Smart contract functionality assessment	8/10
Total Points		**8.1/10**

Contract details

Token contract details for 20th of July 2023

Contract name	Pepe 2.0
Contract address	0x79a94bccC0E4D2f76FDf6a33084E669E174a351c
Token supply	420,000,000,000,000,000
Token ticker	Pepe2.0
Decimals	6
Token holders	1
Transaction count	1
Contract deployer address	0x344376E2CB9502aFf8206d6817a278A3847646DB
Contract ‘s current owner address	0x344376e2cb9502aff8206d6817a278a3847646db

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	Low Issue
		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
		Selfdestruct function security	pass
3	Business security & centralization	Access control of owners	High Issue
		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 🔒		
IERC20	**Interface**			
L	totalSupply	External ❗		NO ❗
L	decimals	External ❗		NO ❗
L	symbol	External ❗		NO ❗

L	name	External ❗		NO ❗
L	getOwner	External ❗		NO ❗
L	balanceOf	External ❗		NO ❗
L	transfer	External ❗	🛑	NO ❗
L	allowance	External ❗		NO ❗
L	approve	External ❗	🛑	NO ❗
L	transferFrom	External ❗	🛑	NO ❗
IFactoryV2	**Interface**			
L	getPair	External ❗		NO ❗
L	createPair	External ❗	🛑	NO ❗
IV2Pair	**Interface**			
L	factory	External ❗		NO ❗
L	getReserves	External ❗		NO ❗
L	sync	External ❗	🛑	NO ❗
IRouter01	**Interface**			

L	factory	External ❗		NO ❗
L	WETH	External ❗		NO ❗
L	addLiquidityETH	External ❗	💵	NO ❗
L	addLiquidity	External ❗	🛑	NO ❗
L	swapExactETHForTokens	External ❗	💵	NO ❗
L	getAmountsOut	External ❗		NO ❗
L	getAmountsIn	External ❗		NO ❗
IRouter02	**Interface**	**IRouter01**		
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO ❗
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO ❗
L	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🛑	NO ❗
L	swapExactTokensForTokens	External ❗	🛑	NO ❗
Pepe20Token	**Implementation**	**IERC20**		
L		Public ❗	💵	NO ❗
L		External ❗	💵	NO ❗

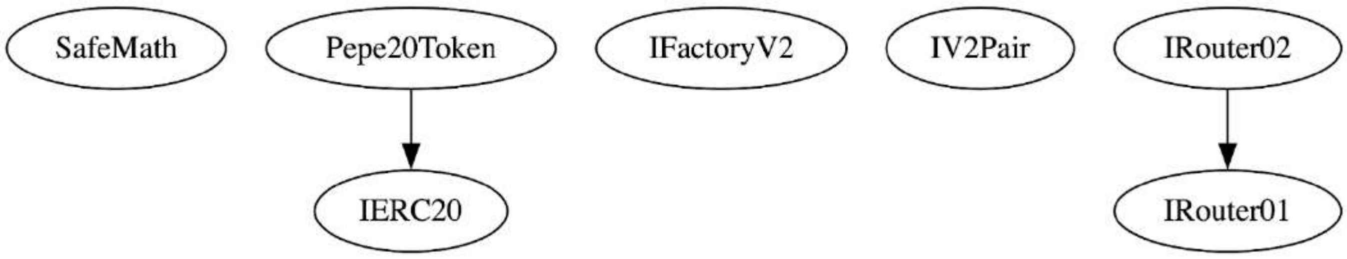
L	totalSupply	External ❗		NO ❗
L	decimals	External ❗		NO ❗
L	symbol	External ❗		NO ❗
L	name	External ❗		NO ❗
L	getOwner	External ❗		NO ❗
L	allowance	Public ❗		NO ❗
L	balanceOf	Public ❗		NO ❗
L	transfer	Public ❗	🛑	NO ❗
L	approve	External ❗	🛑	NO ❗
L	_approve	Internal 🔒	🛑	
L	approveContractContingency	External ❗	🛑	onlyOwner
L	transferFrom	External ❗	🛑	NO ❗
L	_hasLimits	Internal 🔒		
L	_transfer	Internal 🔒	🛑	
L	finalizeTransfer	Internal 🔒	🛑	
L	_transferAmount	Internal 🔒	🛑	

└	_checkLiquidityAdd	Internal 🔒	🛑	
└	swapTokensForEth	Private 🔐	🛑	lockTheSwap
└	sendETHToFee	Private 🔐	🛑	
└	manualswap	External ❗	🛑	NO ❗
└	manualsend	External ❗	🛑	NO ❗
└	transferOwner	External ❗	🛑	onlyOwner
└	renounceOwnership	External ❗	🛑	onlyOwner
└	excludePresaleAddresses	External ❗	🛑	onlyOwner
└	setDevAddress	External ❗	🛑	onlyOwner
└	excludeMultipleAccountsFromFees	Public ❗	🛑	onlyOwner
└	enableTrading	Public ❗	🛑	onlyOwner
└	setFee	Public ❗	🛑	onlyOwner
└	updateSwapEnabled	External ❗	🛑	onlyOwner

Legend

Symbol	Meaning
🛑	Function can modify state
💵	Function is payable

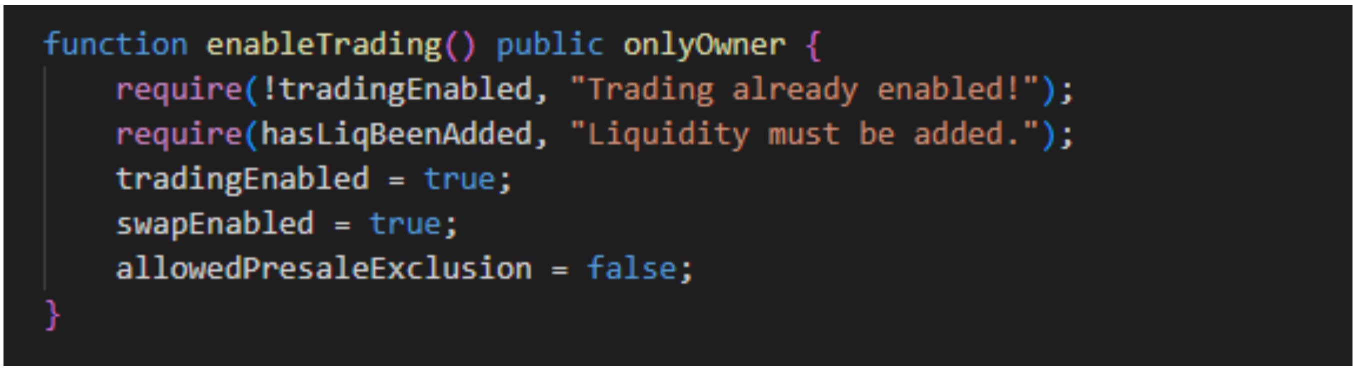
Inheritance Hierarchy



Security issue checking status

❖ **High severity issues**

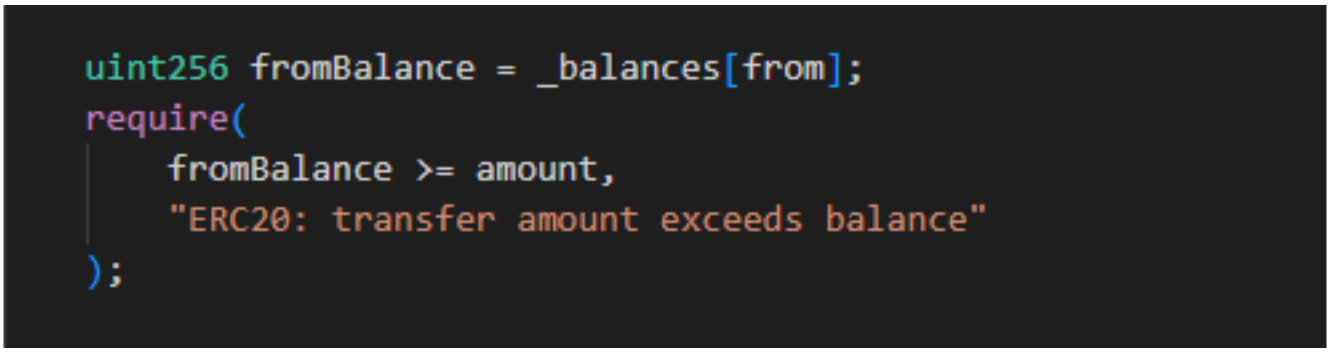
The owner must enable trade for the holders, if trading remains disabled, no one would be able to buy and sell.



```
function enableTrading() public onlyOwner {
    require(!tradingEnabled, "Trading already enabled!");
    require(hasLiqBeenAdded, "Liquidity must be added.");
    tradingEnabled = true;
    swapEnabled = true;
    allowedPresaleExclusion = false;
}
```

❖ **Medium severity issues**

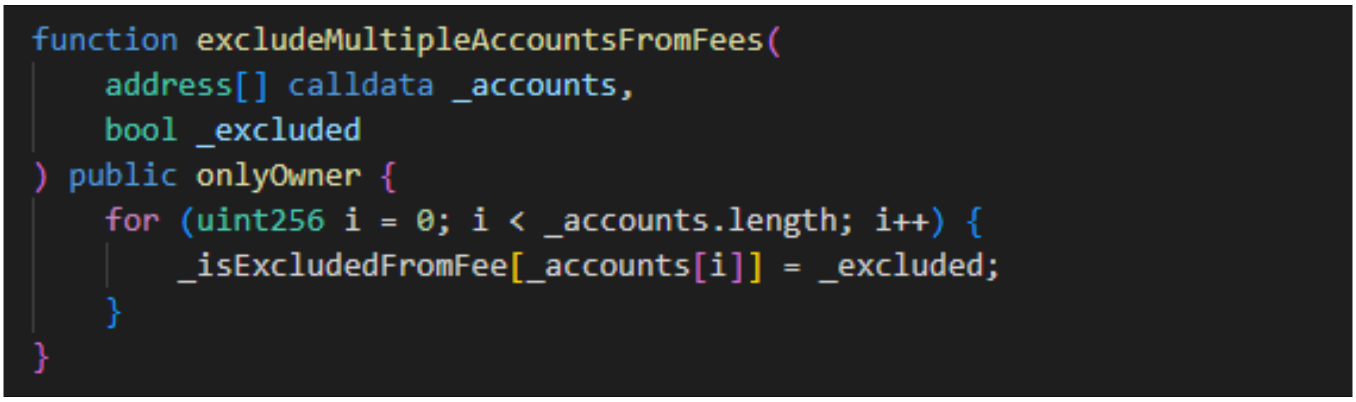
it's checking senders' token availability after doing swaps since swaps are external transactions if the user does not have the required balance then the transaction will fail and the user will have to pay gas fees it's better to check the required balance in the beginning before performing any other transactions



```
uint256 fromBalance = _balances[from];
require(
    fromBalance >= amount,
    "ERC20: transfer amount exceeds balance"
);
```

❖ Low severity issues

When excluding multiple wallets from fees owner can pass any numbers wallets at a time if the owner passes a large number of wallets at a time this might fail due to exceeding block gas limit better to add max wallet limit in here



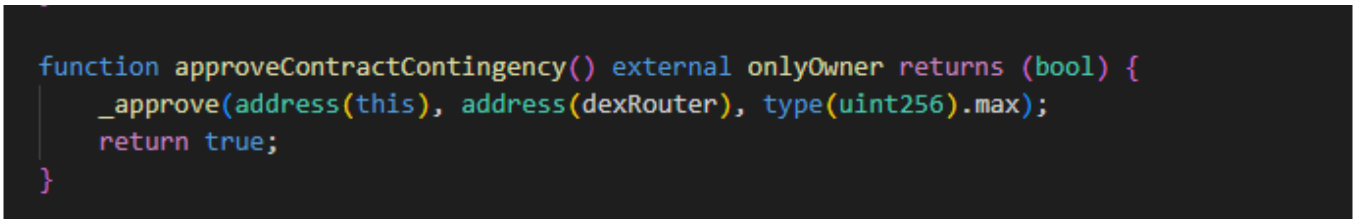
```
function excludeMultipleAccountsFromFees(
    address[] calldata _accounts,
    bool _excluded
) public onlyOwner {
    for (uint256 i = 0; i < _accounts.length; i++) {
        _isExcludedFromFee[_accounts[i]] = _excluded;
    }
}
```

❖ Centralization Risk

No Centralization issues found

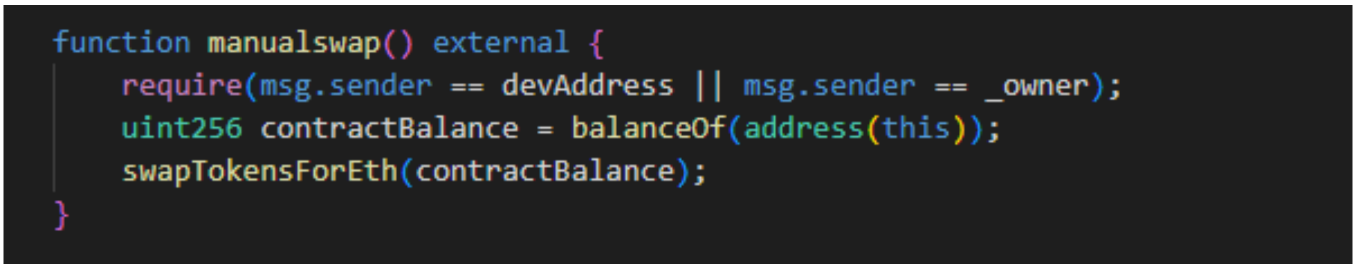
Owner privileges

❖ Owner can give max approval to contract to spend tokens in contract



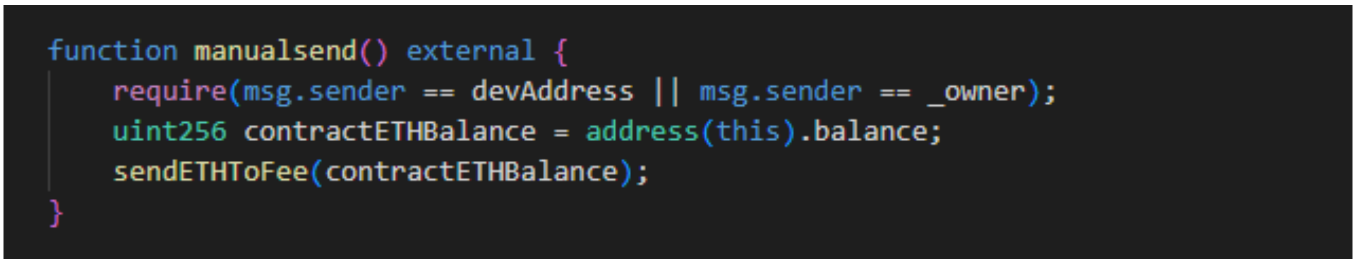
```
function approveContractContingency() external onlyOwner returns (bool) {
    _approve(address(this), address(dexRouter), type(uint256).max);
    return true;
}
```

❖ Owner and dev wallet can manually swap contract tokens before swap point met



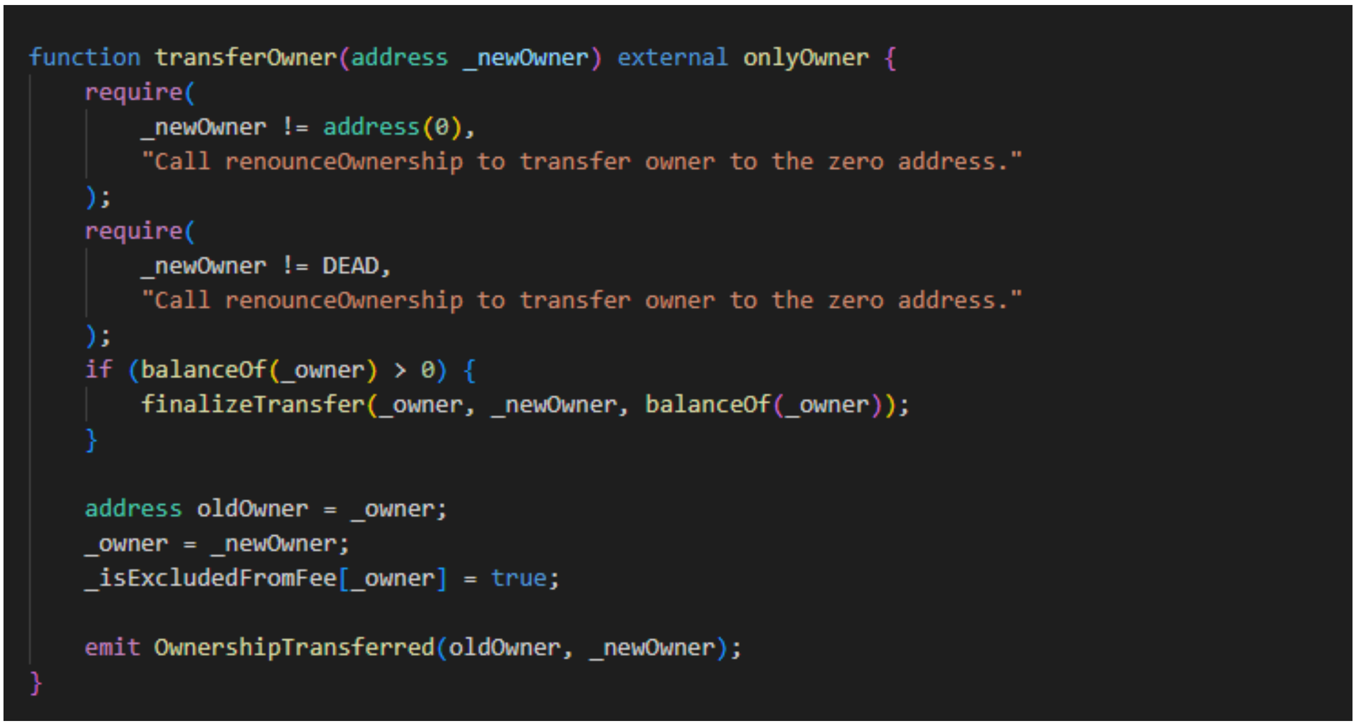
```
function manualswap() external {
    require(msg.sender == devAddress || msg.sender == _owner);
    uint256 contractBalance = balanceOf(address(this));
    swapTokensForEth(contractBalance);
}
```

❖ Owner can dev wallet can transfer contract BNB balance to dev wallet



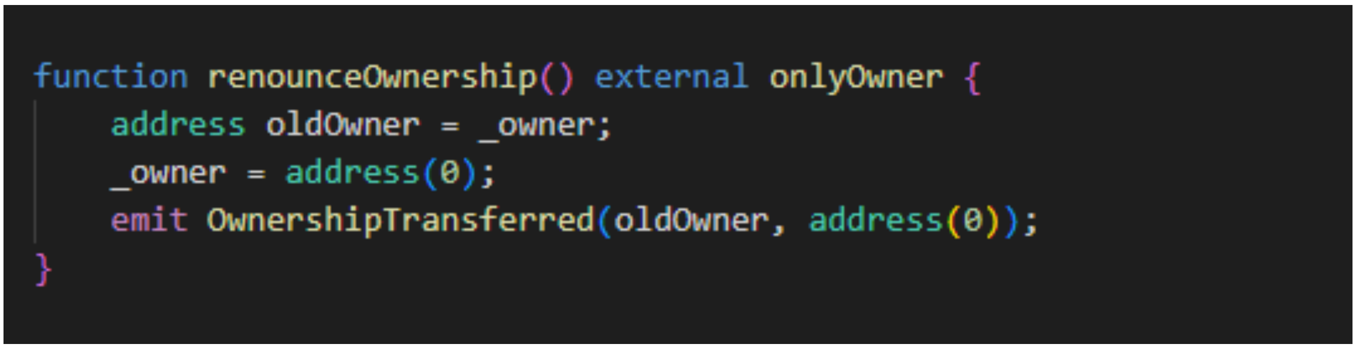
```
function manualsend() external {
    require(msg.sender == devAddress || msg.sender == _owner);
    uint256 contractETHBalance = address(this).balance;
    sendETHToFee(contractETHBalance);
}
```

❖ Owner can transfer ownership



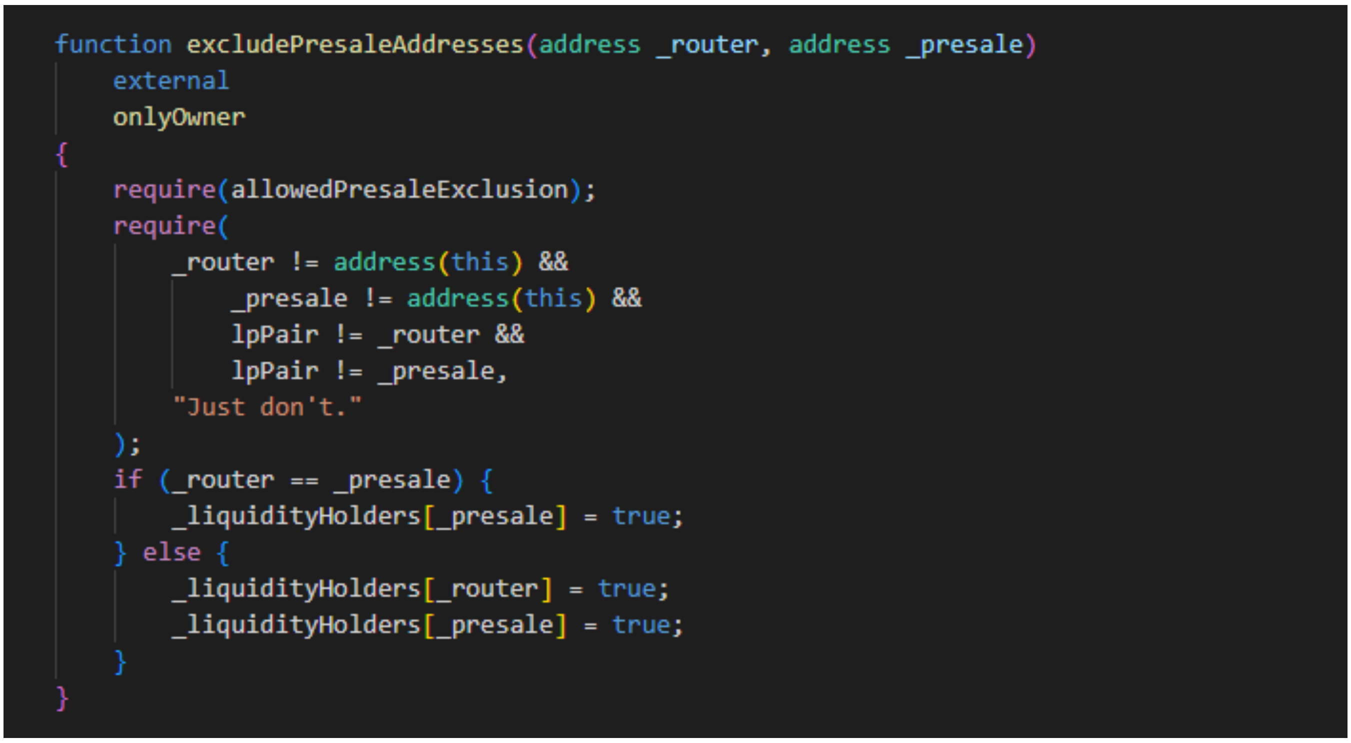
```
function transferOwner(address _newOwner) external onlyOwner {
    require(
        _newOwner != address(0),
        "Call renounceOwnership to transfer owner to the zero address."
    );
    require(
        _newOwner != DEAD,
        "Call renounceOwnership to transfer owner to the zero address."
    );
    if (balanceOf(_owner) > 0) {
        finalizeTransfer(_owner, _newOwner, balanceOf(_owner));
    }

    address oldOwner = _owner;
    _owner = _newOwner;
    _isExcludedFromFee[_owner] = true;

    emit OwnershipTransferred(oldOwner, _newOwner);
}
```

❖ Owner can renounce the ownership



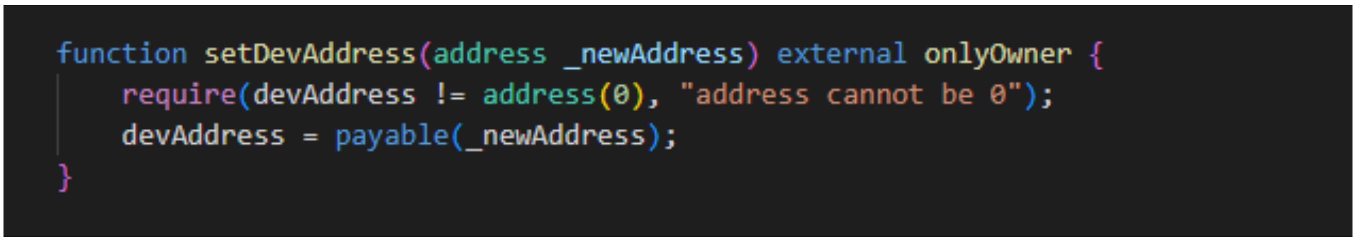
```
function renounceOwnership() external onlyOwner {
    address oldOwner = _owner;
    _owner = address(0);
    emit OwnershipTransferred(oldOwner, address(0));
}
```

- ❖ Owner can exclude presale address (excluded wallets can do transactions before adding lp and enabling trading)



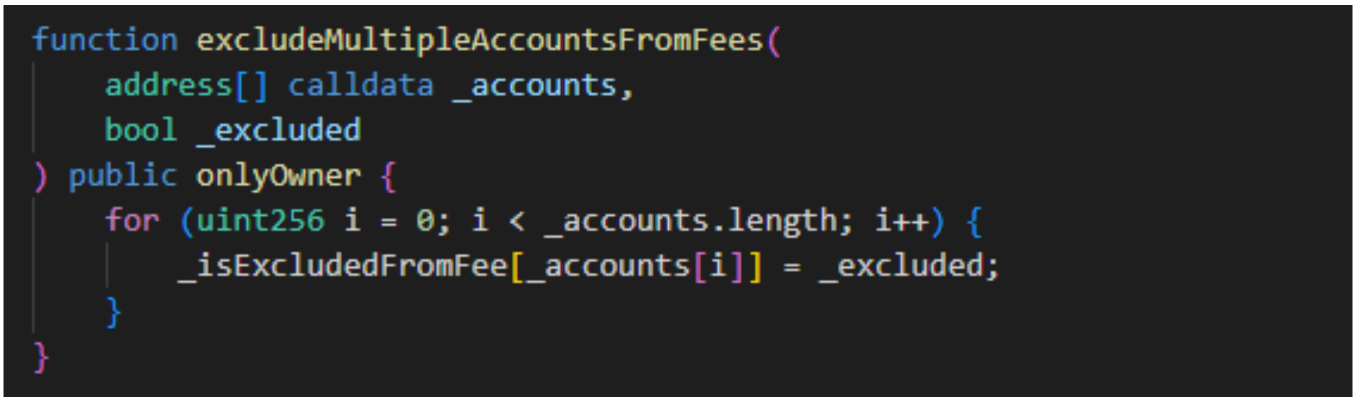
```
function excludePresaleAddresses(address _router, address _presale)
    external
    onlyOwner
{
    require(allowedPresaleExclusion);
    require(
        _router != address(this) &&
            _presale != address(this) &&
            lpPair != _router &&
            lpPair != _presale,
        "Just don't."
    );
    if (_router == _presale) {
        _liquidityHolders[_presale] = true;
    } else {
        _liquidityHolders[_router] = true;
        _liquidityHolders[_presale] = true;
    }
}
```

- ❖ Owner can change the dev wallet



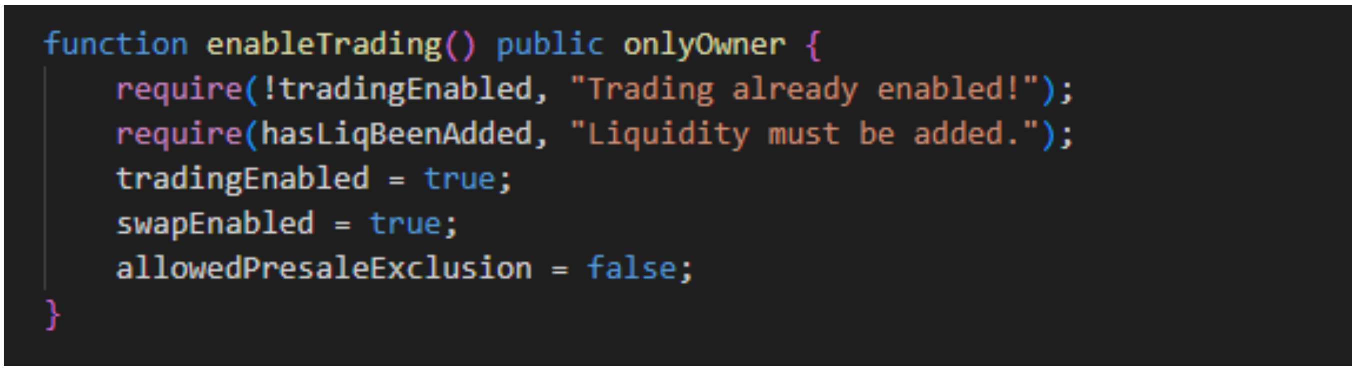
```
function setDevAddress(address _newAddress) external onlyOwner {
    require(devAddress != address(0), "address cannot be 0");
    devAddress = payable(_newAddress);
}
```

- ❖ Owner can exclude/include multiple wallets from fees



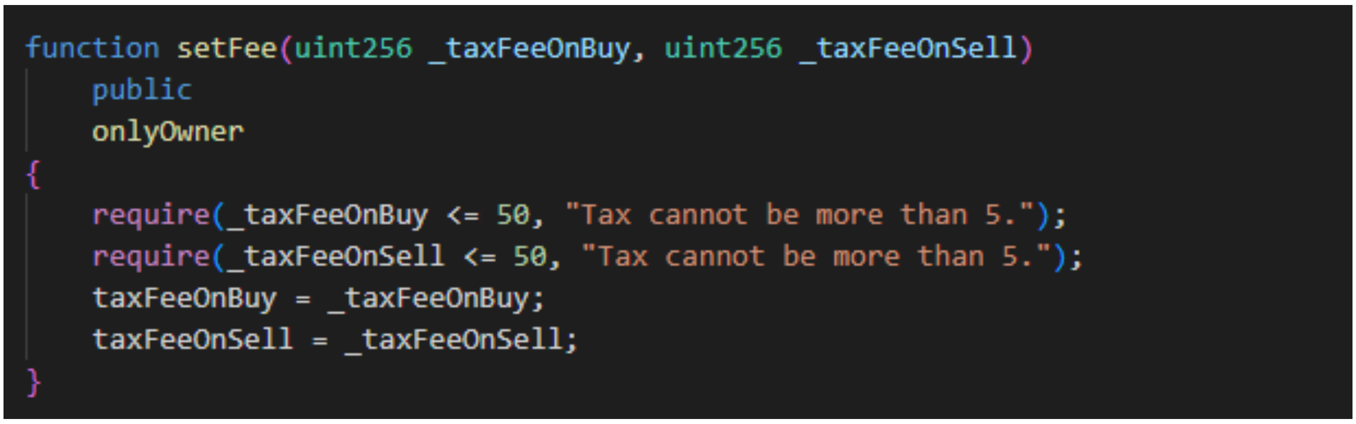
```
function excludeMultipleAccountsFromFees(
    address[] calldata _accounts,
    bool _excluded
) public onlyOwner {
    for (uint256 i = 0; i < _accounts.length; i++) {
        _isExcludedFromFee[_accounts[i]] = _excluded;
    }
}
```

❖ Owner can enable trading, once enabled can not disable again



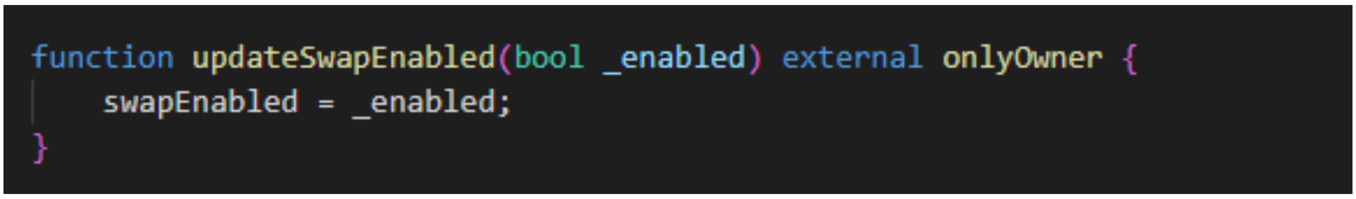
```
function enableTrading() public onlyOwner {
    require(!tradingEnabled, "Trading already enabled!");
    require(hasLiqBeenAdded, "Liquidity must be added.");
    tradingEnabled = true;
    swapEnabled = true;
    allowedPresaleExclusion = false;
}
```

❖ Owner can change buy and sell fees each fees maximum upto 5%



```
function setFee(uint256 _taxFeeOnBuy, uint256 _taxFeeOnSell)
    public
    onlyOwner
{
    require(_taxFeeOnBuy <= 50, "Tax cannot be more than 5.");
    require(_taxFeeOnSell <= 50, "Tax cannot be more than 5.");
    taxFeeOnBuy = _taxFeeOnBuy;
    taxFeeOnSell = _taxFeeOnSell;
}
```

❖ Owner can enable/disable swapping



```
function updateSwapEnabled(bool _enabled) external onlyOwner {
    swapEnabled = _enabled;
}
```

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: **3**

Solidity code functional issue level: **PASS**

Number of owner privileges: **11**

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

Smart contract active ownership: **ACTIVE**