SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

Smurf Coin
[SCOIN]
BEP 20

0x714eee03e9c7ca0eed4d01790a26354e559263e2

Table of Contents

Global Disclaimer

This document serves as a disclaimer for the crypto smart contract audit conducted by Skeleton Ecosystem. The purpose of the audit was to review the codebase of the smart contracts for potential vulnerabilities and issues. It is important to note the following:

Limited Scope: The audit is based on the code and information available up to the audit completion date. It does not cover external factors, system interactions, or changes made after the audit. The audit itself can not guarantee 100% safaty and can not detect common scam methods like farming and developer sell-out.

No Guarantee of Security: While we have taken reasonable steps to identify vulnerabilities, it is impossible to guarantee the complete absence of security risks or issues. The audit report provides an assessment of the contract’s security as of the audit date.

Continued Development: Smart contracts and blockchain technology are evolving fields. Updates, forks, or changes to the contract post-audit may introduce new risks that were not present during the audit.

Third-party Code: If the smart contract relies on third-party libraries or code, those components were not thoroughly audited unless explicitly stated. Security of these dependencies is the responsibility of their respective developers.

Non-Exhaustive Testing: The audit involved automated analysis, manual review, and testing under controlled conditions. It is possible that certain vulnerabilities or issues may not have been identified.

Risk Evaluation: The audit report includes a risk assessment for identified vulnerabilities. It is recommended that the development team carefully reviews and addresses these risks to mitigate potential exploits.

Not Financial Advice: This audit report is not intended as financial or investment advice. Decisions regarding the use, deployment, or investment in the smart contract should be made based on a comprehensive assessment of the associated risks.

By accessing and using this audit report, you acknowledge and agree to the limitations outlined above. Skeleton Ecosystem and its auditors shall not be held liable for any direct or indirect damages resulting from the use of the audit report or the smart contract itself.

Please consult with legal, technical, and financial professionals before making any decisions related to the smart contract.

Overview

Contract Name	Smurf Coin
Ticker/Simbol	SCOIN
Blockchain	Binance Smart Chain BEP20
Contract Address	0x714eee03e9c7ca0eed4d01790a26354e559263e2
Creator Address	0x8cd964B395ADb091a573ABdF14329f8bde240E68
Current Owner Address	Renounced
Contract Explorer	https://bscscan.com/token/0x714eee03e9c7ca0eed4d01790a26354e559263e2
Compiler Version	v0.8.18+commit.87f61d96
License	MIT
Optimisation	No with 200 Runs
Total Supply	870,202,749,934,997.960191 **SCOIN**
Decimals	9

Creation/Audit

Contract Deployed	20 Sept 2023
Audit Created	20 Sept 2023
Audit Update	V 1.0

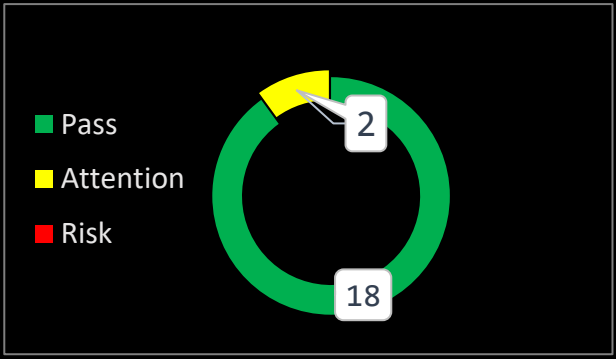
Verified Socials

Website	https://smurfcoin.org/
Telegram	https://t.me/SmurfCoinCommunity
Twitter (X)	https://twitter.com/SmurfCoinGlobal

SMURF COIN BEP20

Contract Function Analysis

Pass Attention Item Risky Item

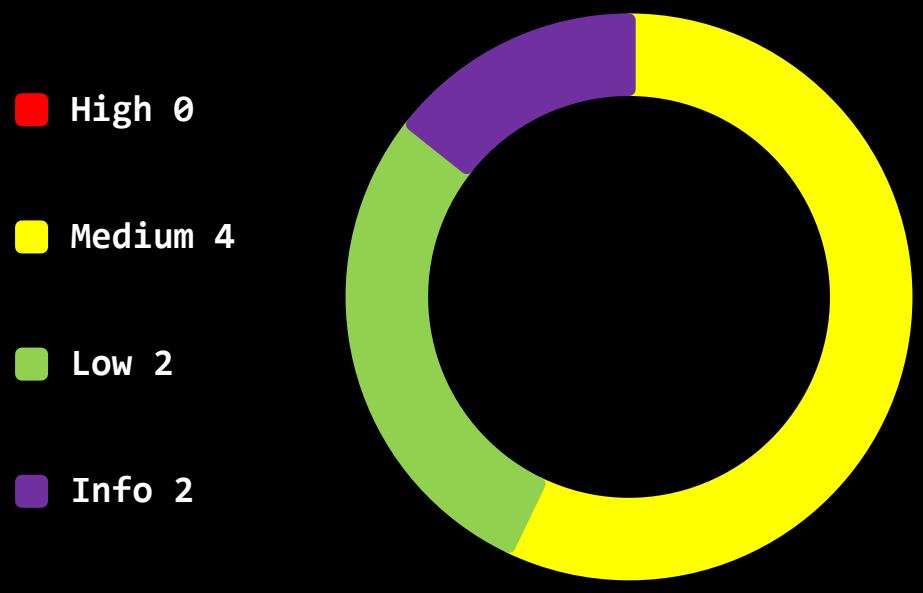


Contract Verified	✓	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership	✓	Renounced
Buy Tax	8 %	Shows the taxes for purchase transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable.
Sell Tax	8 %	Shows the taxes for sell transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable.
Honeypot Analyse	✓	Holder is able to buy and sell. If honeypot: The contract blocks sell transfer from holder wallet. Multiple events may cause honeypot. Trading disabled, extremely high tax
Liqudity Status	✓	LP Lock found: 96% Locked on Mudra Locker for 367 Days on 20.09.2023 3,8% LP Burn found on 20.09.2013
Trading Disable Functions	✓	No Trading suspendable function found. If a suspendable code is included, the token maybe neither be bought or sold (honeypot risk). If contract is renounced this function can't be used
Set Fees function	✓	No Fee Setting function found. The contract owner may contain the authority to modify the transaction tax. If the transaction tax is increased to more than 49%, the tokens may not be able to be traded (honeypot risk).
Proxy Contract	✓	Not a Proxy Contract. The proxy contract means contract owner can modifiy the function of the token and possibly effect the price. The Owner is not the creator but the creator may have authorisation to change functions.
Mint Function	✓ ⚠	Mint function found but contract is Renounced, the function can not be used. Mint function is transparent or non-existent. Hidden mint functions may increase the amount of tokens in circulation and effect the price of the token. Owner can mint new tokens and sell.

Balance Modifier Function	✅	No Balance Modifier function found. If there is a function for this, the contract owner can have the authority to modify the balance of tokens at other addresses. For example revoke the bought tokens from the holders wallet. Common form of scam: You buy the token, but it's disappearing from your wallet.
Blacklist Function	✅⚠️	Blacklist function found but contract renounced. This function can not be used. If there is a blacklist, some addresses may not be able to trade normally. Example: you buy the token and right after your Wallet getting blacklisted. Like so you will be unable to sell. Honeypot Risk.
Whitelist Function	✅⚠️	Whitelist function found but contract renounced. This function can not be used. If there is a function for this Developer can set zero fee or no max wallet size for adresses (for example team wallets can trade without fee. Can cause farming)
Hidden Owner Analysis	✅	No authorised hidden owner found. For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned. Fake renounce.
Retrieve Ownership Function	✅	No functions found which can retrieve ownership of the contract. If this function exists, it is possible for the project owner to regain ownership even after relinquishing it. Also known as fake renounce.
Self Destruct Function	✅	No Self Destruct function found. If this function exists and is triggered, the contract will be destroyed, all functions will be unavailable, and all related assets will be erased.
Specific Tax Changing Function	✅	No Specific Tax Changing Functions found. If it exists, the contract owner may set a very outrageous tax rate for assigned address to block it from trading. Can assign all wallets at once!
Trading Cooldown Function	✅	No Trading Cooldown Function found. If there is a trading cooldown function, the user will not be able to sell the token within a certain time or block after buying. Like a temporary honeypot.
Max Transaction and Holding Modify Function	✅	No Max Transaction and Holding Modify function found. If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot
Transaction Limiting Function	✅	No Transaction Limiter Function Found. The number of overall token transactions may be limited (honeypot risk)

Contract Security

Total Findings: 8



High Severity Issues: High possibility to cause problems, need to be resolved.

Medium Severity Issue: Will likely cause problems, recommended to resolve.

Low Severity Issues: Won’t cause problems, but for improvement purposes could be adjusted.

Informational Severity Issues: Not harmful in any way, information for the developer team.

Contract Security

List of Found Issues

High severity Issues: (0)

Medium severity issues: (4)

- Reentrancy
- Unchecked Transfer
- Unchecked Array Lenght
- Usage of EXTCodesize to check for externally owned accounts

Low severity issues: (2)

- Missing Events
- Use of Floating Pragma

Informational severity issues: (2)

- Hard Coded Address
- Public Functions Should be Declared External

Contract Weakness Classisication

THE SMART CONTRACT WEAKNESS CLASSIFICATION REGISTRY (SWC REGISTRY) IS AN IMPLEMENTATION OF THE WEAKNESS CLASSIFICATION SCHEME PROPOSED IN EIP-1470. IT IS LOOSELY ALIGNED TO THE TERMINOLOGIES AND STRUCTURE USED IN THE COMMON WEAKNESS ENUMERATION (CWE) WHILE OVERLAYING A WIDE RANGE OF WEAKNESS VARIANTS THAT ARE SPECIFIC TO SMART CONTRACTS.

ID	Description	AI	Manual	Result
SWC-100	Function Default Visibility	Passed	Passed	Passed
SWC-101	Integer Overflow and Underflow	Passed	Passed	Passed
SWC-102	Outdated Compiler Version	Passed	Passed	Passed
SWC-103	Floating Pragma	Low	Passed	Passed
SWC-104	Unchecked Call Return Value	Passed	Passed	Passed
SWC-105	Unprotected Ether Withdrawal	Passed	Passed	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed	Passed	Passed
SWC-107	Reentrancy	High	Medium	Medium
SWC-108	State Variable Default Visibility	Passed	Passed	Passed
SWC-109	Uninitialized Storage Pointer	Passed	Passed	Passed
SWC-110	Assert Violation	Passed	Passed	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed	Passed	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed	Passed	Passed
SWC-113	DoS with Failed Call	Passed	Passed	Passed
SWC-114	Transaction Order Dependence	Passed	Passed	Passed
SWC-115	Authorization through tx.origin	Passed	Passed	Passed
SWC-116	Block values as a proxy for time	Passed	Passed	Passed
SWC-117	Signature Malleability	Passed	Passed	Passed
SWC-118	Incorrect Constructor Name	Passed	Passed	Passed
SWC-119	Shadowing State Variables	Passed	Passed	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed	Passed	Passed

SWC-121	Missing Protection against Signature Replay Attacks	**Passed**	**Passed**	**Passed**
SWC-122	Lack of Proper Signature Verification	**Passed**	**Passed**	**Passed**
SWC-123	Requirement Violation	**Passed**	**Passed**	**Passed**
SWC-124	Write to Arbitrary Storage Location	**Passed**	**Passed**	**Passed**
SWC-125	Incorrect Inheritance Order	**Passed**	**Passed**	**Passed**
SWC-126	Insufficient Gas Griefing	**Passed**	**Passed**	**Passed**
SWC-127	Arbitrary Jump with Function Type Variable	**Passed**	**Passed**	**Passed**
SWC-128	DoS With Block Gas Limit	**Passed**	**Passed**	**Passed**
SWC-129	Typographical Error	**Passed**	**Passed**	**Passed**
SWC-130	Right-To-Left-Override control character (U+202E)	**Passed**	**Passed**	**Passed**
SWC-131	Presence of unused variables	**Passed**	**Passed**	**Passed**
SWC-132	Unexpected Ether balance	**Passed**	**Passed**	**Passed**
SWC-133	Hash Collisions With Multiple Variable Length Arguments	**Passed**	**Passed**	**Passed**
SWC-134	Message call with hardcoded gas amount	**Passed**	**Passed**	**Passed**
SWC-135	Code With No Effects	**Passed**	**Passed**	**Passed**
SWC-136	Unencrypted Private Data On-Chain	**Passed**	**Passed**	**Passed**

Detected High and Medium Severity Vulnerability Description

⚠ Unchecked Transfer (1 Items)

Item: 1	Location:	Line 519	Severity:	Medium

Function	Some tokens do not revert the transaction when the transfer or transferFrom fails and returns False. Hence we must check the return value after calling the transfer or transferFrom function.
Remedation	Use OpenZeppelin SafeERC20's safetransfer and safetransferFrom functions.

```
515    function remove_Random_Tokens(address random_Token_Address1, uint256 percent_of_Tokens1) public returns(bool _sent1){
516        require(random_Token_Address1 != address(this), "Can not remove native token");
517        uint256 totalRandom = IERC20(random_Token_Address1).balanceOf(address(this));
518        uint256 removeRandom = totalRandom*percent_of_Tokens1/100;
519        _sent1 = IERC20(random_Token_Address1).transfer(Wallet_Dev, removeRandom);
520    }
```

⚠ Unchecked Array Lenght (1 Items)

Item: 1	Location:	Line 370	Severity:	Medium

Function	Ethereum is a very resource-constrained environment. Prices per computational step are orders of magnitude higher than with centralized providers. Moreover, Ethereum miners impose a limit on the total number of Gas consumed in a block. If array.length is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed. for (uint256 i = 0; i < array.length ; i++) { cosltyFunc(); } This becomes a security issue if an external actor influences array.length. E.g., if an array enumerates all registered addresses, an adversary can register many addresses, causing the problem described above.
Remedation	Either explicitly or just due to normal operation, 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. Therefore, loops with a bigger or unknown number of steps should always be avoided.

```
ftrace | funcSig
369 function bulkAntiBot(address[] memory accounts, bool state) external onlyOwner{
370     for(uint256 i = 0; i < accounts.length; i++){
371         isBot[accounts[i]] = state;
372     }
373 }
374
```

⚠ Reentrancy (1 Items)

Item: 1	Location:	Line 515-520	Severity:	Medium

Function	In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls. This may lead to loss of funds, improper value updates, token loss, etc.
Remedation	It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing must happen before the call.

```
515 function remove_Random_Tokens(address random_Token_Address1, uint256 percent_of_Tokens1) public returns(bool _sent1){
516     require(random_Token_Address1 != address(this), "Can not remove native token");
517     uint256 totalRandom = IERC20(random_Token_Address1).balanceOf(address(this));
518     uint256 removeRandom = totalRandom*percent_of_Tokens1/100;
519     _sent1 = IERC20(random_Token_Address1).transfer(Wallet_Dev, removeRandom);
520 }
```

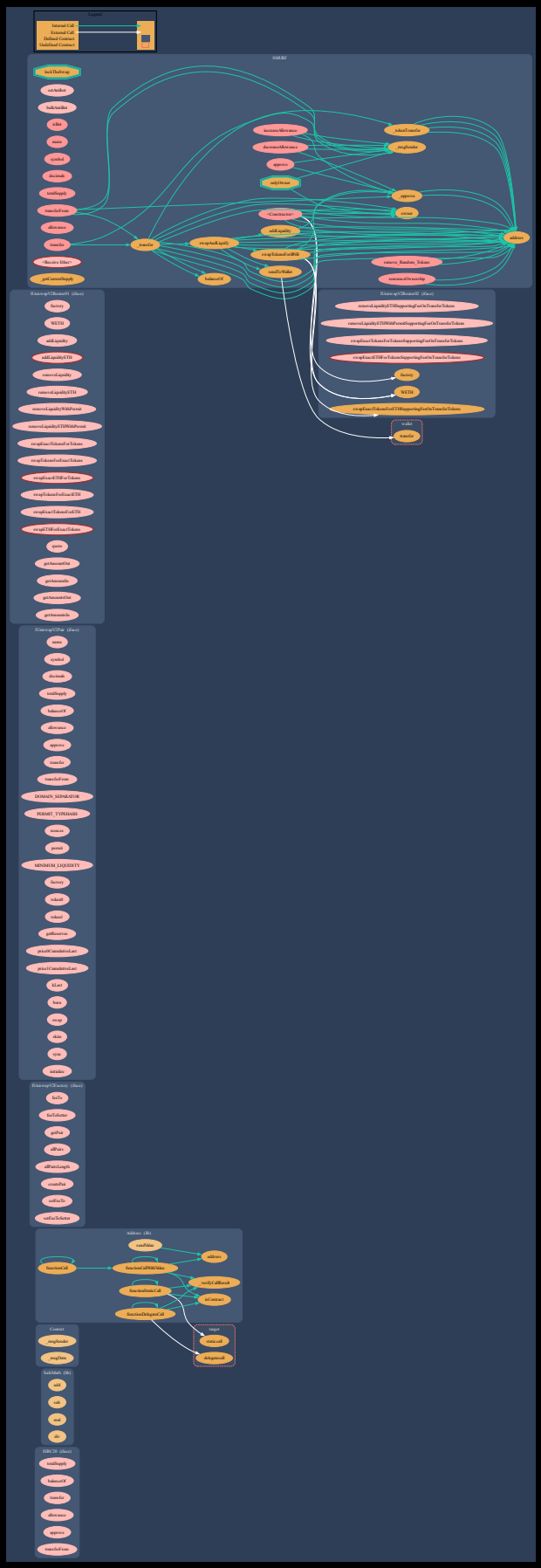
⚠ Usage of EXTCodesize to check for externally owned accounts (1 Items)

Item: 1	Location:	Line 59	Severity:	Medium

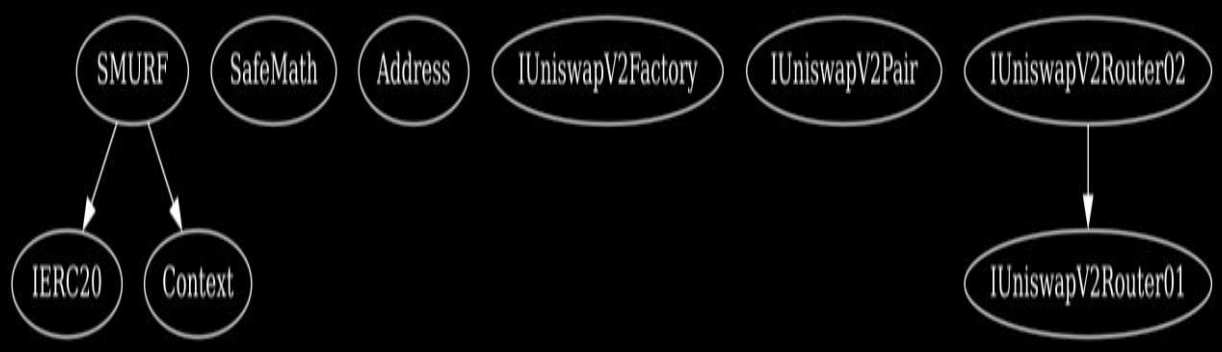
Function	extcodesize is used to check if a contract is an externally owned account or another contract. extcodesize returns 0 for externally owned accounts but there's a specific condition here that when an extcodesize check is made to a contract that is still under construction or when the contract's constructor is running, extcodesize for its address returns zero. This may give erroneous outputs for checking externally owned contracts.
Remedation	It is recommended to manually check and validate at compile-time that the contract/account address being checked inside extcodesize does not return improper values due to the external contract's construction.

```
ftrace | funcSig
57  function isContract(address account) internal view returns (bool) {
58      uint256 size;
59      assembly { size := extcodesize(account) }
60      return size > 0;
61  }
```

Contract Flow Graph



Inheritance Graph



Contract Functions

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
IERC20	Interface			
	totalSupply	External !		NO!
	balanceOf	External !		NO!
	transfer	External !	🛑	NO!
	allowance	External !		NO!
	approve	External !	🛑	NO!
	transferFrom	External !	🛑	NO!
SafeMath	Library			
	add	Internal 🔒		
	sub	Internal 🔒		
	mul	Internal 🔒		
	div	Internal 🔒		
	sub	Internal 🔒		
	div	Internal 🔒		
Context	Implementation			
	_msgSender	Internal 🔒		
	_msgData	Internal 🔒		
Address	Library			
	isContract	Internal 🔒		
	sendValue	Internal 🔒	🛑	
	functionCall	Internal 🔒	🛑	
	functionCall	Internal 🔒	🛑	
	functionCallWithValue	Internal 🔒	🛑	
	functionCallWithValue	Internal 🔒	🛑	
	functionStaticCall	Internal 🔒		
	functionStaticCall	Internal 🔒		
	functionDelegateCall	Internal 🔒	🛑	
	functionDelegateCall	Internal 🔒	🛑	

	_verifyCallResult	Private 🔐		
IUniswapV2Factory	Interface			
	feeTo	External ❗		NO❗
	feeToSetter	External ❗		NO❗
	getPair	External ❗		NO❗
	allPairs	External ❗		NO❗
	allPairsLength	External ❗		NO❗
	createPair	External ❗	🛑	NO❗
	setFeeTo	External ❗	🛑	NO❗
	setFeeToSetter	External ❗	🛑	NO❗
IUniswapV2Pair	Interface			
	name	External ❗		NO❗
	symbol	External ❗		NO❗
	decimals	External ❗		NO❗
	totalSupply	External ❗		NO❗
	balanceOf	External ❗		NO❗
	allowance	External ❗		NO❗
	approve	External ❗	🛑	NO❗
	transfer	External ❗	🛑	NO❗
	transferFrom	External ❗	🛑	NO❗
	DOMAIN_SEPARATOR	External ❗		NO❗
	PERMIT_TYPEHASH	External ❗		NO❗
	nonces	External ❗		NO❗
	permit	External ❗	🛑	NO❗
	MINIMUM_LIQUIDITY	External ❗		NO❗
	factory	External ❗		NO❗
	token0	External ❗		NO❗
	token1	External ❗		NO❗
	getReserves	External ❗		NO❗
	price0CumulativeLast	External ❗		NO❗
	price1CumulativeLast	External ❗		NO❗
	kLast	External ❗		NO❗
	burn	External ❗	🛑	NO❗

	swap	External !	🛑	NO!
	skim	External !	🛑	NO!
	sync	External !	🛑	NO!
	initialize	External !	🛑	NO!
IUniswapV2Router01	Interface			
	factory	External !		NO!
	WETH	External !		NO!
	addLiquidity	External !	🛑	NO!
	addLiquidityETH	External !	💵	NO!
	removeLiquidity	External !	🛑	NO!
	removeLiquidityETH	External !	🛑	NO!
	removeLiquidityWithPermit	External !	🛑	NO!
	removeLiquidityETHWithPermit	External !	🛑	NO!
	swapExactTokensForTokens	External !	🛑	NO!
	swapTokensForExactTokens	External !	🛑	NO!
	swapExactETHForTokens	External !	💵	NO!
	swapTokensForExactETH	External !	🛑	NO!
	swapExactTokensForETH	External !	🛑	NO!
	swapETHForExactTokens	External !	💵	NO!
	quote	External !		NO!
	getAmountOut	External !		NO!
	getAmountIn	External !		NO!
	getAmountsOut	External !		NO!
	getAmountsIn	External !		NO!
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External !	🛑	NO!

	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO❗
	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
SMURF	Implementation	Context, IERC20		
	owner	Public ❗		NO❗
	renounceOwnership	Public ❗	🛑	NO❗
		Public ❗	🛑	NO❗
	setAntibot	External ❗	🛑	onlyOwner
	bulkAntiBot	External ❗	🛑	onlyOwner
	isBot	Public ❗		NO❗
	name	Public ❗		NO❗
	symbol	Public ❗		NO❗
	decimals	Public ❗		NO❗
	totalSupply	Public ❗		NO❗
	balanceOf	Public ❗		NO❗
	transfer	Public ❗	🛑	NO❗
	allowance	Public ❗		NO❗
	approve	Public ❗	🛑	NO❗
	transferFrom	Public ❗	🛑	NO❗
	increaseAllowance	Public ❗	🛑	NO❗
	decreaseAllowance	Public ❗	🛑	NO❗
		External ❗	💵	NO❗
	_getCurrentSupply	Private 🔒		
	_approve	Private 🔒	🛑	
	_transfer	Private 🔒	🛑	

	sendToWallet	Private 🔒	🛑	
	swapAndLiquify	Private 🔒	🛑	lockTheSwap
	swapTokensForBNB	Private 🔒	🛑	
	addLiquidity	Private 🔒	🛑	
	remove_Random_Tokens	Public ❗	🛑	NO❗
	_tokenTransfer	Private 🔒	🛑	

🛑 Function can modify state

💵 Function is payable

Source:

File Name	SHA-1 Hash
c:\Solidity\smurfcoin.sol	3d74665f7cc6b587628fd5e76474efdeeff22e81

Audit Scope

Audit Method.

Our smart contract audit is an extensive methodical examination and analysis of the smart contract's code that is used to interact with the blockchain. Goal: discover errors, issues and security vulnaribilities in the code. Findings getting reported and improvements getting suggested.

Automatic and Manual Review
We are using automated tools to scan functions and weeknesses of the contract. Transfers, integer over-undeflow checks such as all CWE events.

Tools we use:
Visual Studio Code
CWE
SWC
Solidity Scan
SVD

In manual code review our auditor looking at source code and performing line by line examination. This method helps to clarify developer's coding decisions and business logic.

Skeleton Ecosystem

https://skeletonecosystem.com

https://github.com/SkeletonEcosystem/Audits