

# SKELETON ECOSYSTEM

SMART CONTRACT AUDIT



**Tomoko**  
**MOKO**  
**BEP20**

0x6489ce9E6201ca83243f58c578C9cE28268A743C



## Table of Contents

Table of Contents	1
Disclaimer	2
Overview	3
Creation/Audit Date	3
Verified Socials	3
Contract Functions Analysis	4
Contract Safety and Weakness	7
Detected Vulnerability Description	11
Contract Flow Graph	14
Contract Interaction Graph	15
Inheritance Graph	16
Contract Descriptions	17
Audit Scope	21

## 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% safety 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	TOMOKOTOKEN
Ticker/Symbol	MOKO
Blockchain	Binance Smart Chain BEP20
Contract Address	0x6489ce9E6201ca83243f58c578C9cE28268A743C
Creator Address	0x9237d53C70aa1bCE8459D3c51fC038423096313b
Current Owner Address	0x0000000000000000000000000000000000000000
Contract Explorer	<a href="https://bscscan.com/address/0x6489ce9e6201ca83243f58c578c9ce28268a743c#code">https://bscscan.com/address/0x6489ce9e6201ca83243f58c578c9ce28268a743c#code</a>
Compiler Version	v0.8.19+commit.7dd6d404
License	Unlicense
Optimisation	Yes with 200 Runs
Total Supply	420,000,000,000,000,000 MOKO
Decimals	9




## Creation/Audit




Contract Deployed	27.12.2023
Audit Created	04.01.2024
Audit Update	V 1.0

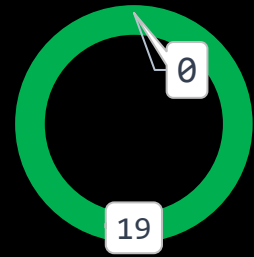
## Verified Socials

Website	<a href="https://mokobsc.com/">https://mokobsc.com/</a>
Telegram	<a href="https://t.me/Watamote_S2">https://t.me/Watamote_S2</a>
Twitter (X)	<a href="https://x.com/Moko_BSC">https://x.com/Moko_BSC</a>

## Contract Function Analysis

 Pass
  Attention Item
  Risky Item

 Pass  
 Attention  
 Risky



Contract Verified		The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership		0x0000000000000000000000000000000000000000000000000000000000000000 previous: 0x9d1EA54d001B6B5c21b976dC3dC01B09044e4e78 (transferred from creator)
Buy Tax	5 %	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. Fee can be set!
Sell Tax	5 %	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. Fee can be set!
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
Liquidity Status		Liquidity Locker status 02.01.2024: 100% LP Tokens Burned
Trading Disable Functions	 	Trading suspendable function found, but contract is renounced, this function can not be triggered.  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!
Mint Function		No Mint Function detected  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		<p>No Balance Modifier function found.</p> <p>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.</p>
Blacklist Function		<p>No Blacklist Setting function found.</p> <p>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.</p>
Whitelist Function	 	<p>Whitelist Setting function found, but contract is renounced, this function can not be triggered.</p> <p>If there is a function for this Developer can set zero fee or no max wallet size for addresses (for example team wallets can trade without fee. Can cause farming)</p>
Hidden Owner Analysis		<p>No hidden or multi owner</p> <p>For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned.</p>
Retrieve Ownership Function		<p>No functions found which can retrieve ownership of the contract.</p> <p>If this function exists, it is possible for the project owner to regain ownership even after relinquishing it. Also known as fake renounce.</p>
Self Destruct Function		<p>No Self Destruct function found.</p> <p>If this function exists and is triggered, the contract will be destroyed, all functions will be unavailable, and all related assets will be erased.</p>
Specific Tax Changing Function		<p>No Specific Tax Changing Functions found.</p> <p>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!</p>
Trading Cooldown Function		<p>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.</p>
Max Transaction and Holding Modify Function		<p>No Max Transaction and Holding Modify function found.</p> <p>If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot</p>
Transaction Limiting Function		<p>No Transaction Limiter Function Found.</p> <p>The number of overall token transactions may be limited (honeypot risk)</p>

## Details of Risk - Attention Items



Function risks removed, based on the ownership of the contract got renounced and no option found to regain ownership.

(Ownership tranferred from creator to 0x9d1EA54d001B6B5c21b976dC3dC01B09044e4e78 to 0x0000000000000000000000000000000000000000)

Transaction Receipt Event Logs

282

Address0x6489ce9e6201ca83243f58c578c9ce28268a743c

NameOwnershipTransferred (index\_topic\_1 address previousOwner, index\_topic\_2 address newOwner) View Source

Topics

00x8be0079c531659141344cd1fd0a4f28419497f9722a3daafe3b4186f6b6457e0

1: previousOwnerDec → 0x9d1EA54d001B6B5c21b976dC3dC01B09044e4e78

2: newOwnerDec → 0x0000000000000000000000000000000000000000

Data0x

The following functions are listed for informational purposes.



**Whitelist (Set excluded)** Contract is renounced, this function can not be triggered.

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)

```

280      ftrace | funcSig
281      function setNoFeeWallet(address account!, bool enabled!) public onlyOwner {
282          noFee[account!] = enabled!;
283      }

```



**Trading Suspensible Functions.** Contract is renounced, this function can not be triggered.

```

441      /
442
443      ftrace | funcSig
444      function enableTrading() external onlyOwner {
445          require(!isTradingEnabled, "Trading already enabled");
446          isTradingEnabled = true;
447          emit _enableTrading();
448      }

```

## Contract Security

Total Findings: 4

■ High 0

■ Medium 0

■ Low 2

■ Info 2



■ **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: (0)**

■ **Low severity issues: (2)**

- Missing Events
- Front Running attack approval

■ **Informational severity issues: (2)**

- Public Functions Should be Declared External
- State Variables Should be Declared Constant

## 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	Passed	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	Passed	Passed	Passed
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	low	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.



No high or medium severity issues found

## Approve of front running attack (2 Items)

Item: 1	Location:	Line 220-248	Severity:	 Low
---------	-----------	--------------	-----------	-----------------------------------------------------------------------------------------

<b>Function</b>	<p>The () method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account.</p> <p>This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function.</p> <p>The function can be front-run by abusing the _approve function.</p>
<b>Remediation</b>	<ol style="list-style-type: none"> <li>1. Introduce mechanisms that limit the maximum acceptable gas price for transactions. This can help prevent front-runners from drastically increasing the gas fees to prioritize their transactions.</li> <li>2. Use transaction taxes to prevent against front-run attack</li> </ol>

```

220 constructor () {
221     _noFee[msg.sender] = true;
222
223     if (block.chainid == 56) {
224         swapRouter = IRouter02(0x10ED43C718714eb63d5aA57B78854704E256024E);
225     } else if (block.chainid == 97) {
226         swapRouter = IRouter02(0x099D1c33F9fC3444f8101754aBC46c52416550D1);
227     } else if (block.chainid == 1 || block.chainid == 4 || block.chainid == 3) {
228         swapRouter = IRouter02(0x7a250d5630B4cF539739dF2C5dAcB4c659F2488D);
229     } else if (block.chainid == 42161) {
230         swapRouter = IRouter02(0x1b02dA8Cb0d097eB8D57A175b88c7D8b47997506);
231     } else if (block.chainid == 5) {
232         swapRouter = IRouter02(0x7a250d5630B4cF539739dF2C5dAcB4c659F2488D);
233     } else {
234         revert("Chain not valid");
235     }
236     liquidityAdd[msg.sender] = true;
237     balance[msg.sender] = _totalSupply;
238     emit Transfer(address(0), msg.sender, _totalSupply);
239
240     require(buyAllocation + sellAllocation + liquidityAllocation == 100, "Fredy: Must equals to 100%");
241
242     lpPair = IFactoryV2(swapRouter.factory()).createPair(swapRouter.WETH(), address(this));
243     isLpPair[lpPair] = true;
244     _approve(msg.sender, address(swapRouter), type(uint256).max);
245     _approve(address(this), address(swapRouter), type(uint256).max);
246
247
248 }
249
  
```

Item: 2	Location: Line 257-260	Severity: <span style="color: green;">■</span> Low
---------	------------------------	----------------------------------------------------

<b>Function</b>	<p>The approve() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver when they try to withdraw certain tokens from the sender's account. Meanwhile, if the sender decides to change the amount and sends another approve transaction, the receiver can notice this transaction before it's mined and can extract tokens from both the transactions, therefore, ending up with tokens from both the transactions. This is a front-running attack affecting the ERC20 Approve function.</p> <p>The function approve can be front-run by abusing the _approve function.</p>
<b>Remediation</b>	<ol style="list-style-type: none"> <li>3. Introduce mechanisms that limit the maximum acceptable gas price for transactions. This can help prevent front-runners from drastically increasing the gas fees to prioritize their transactions.</li> <li>4. Use transaction taxes to prevent against front-run attack</li> </ol>

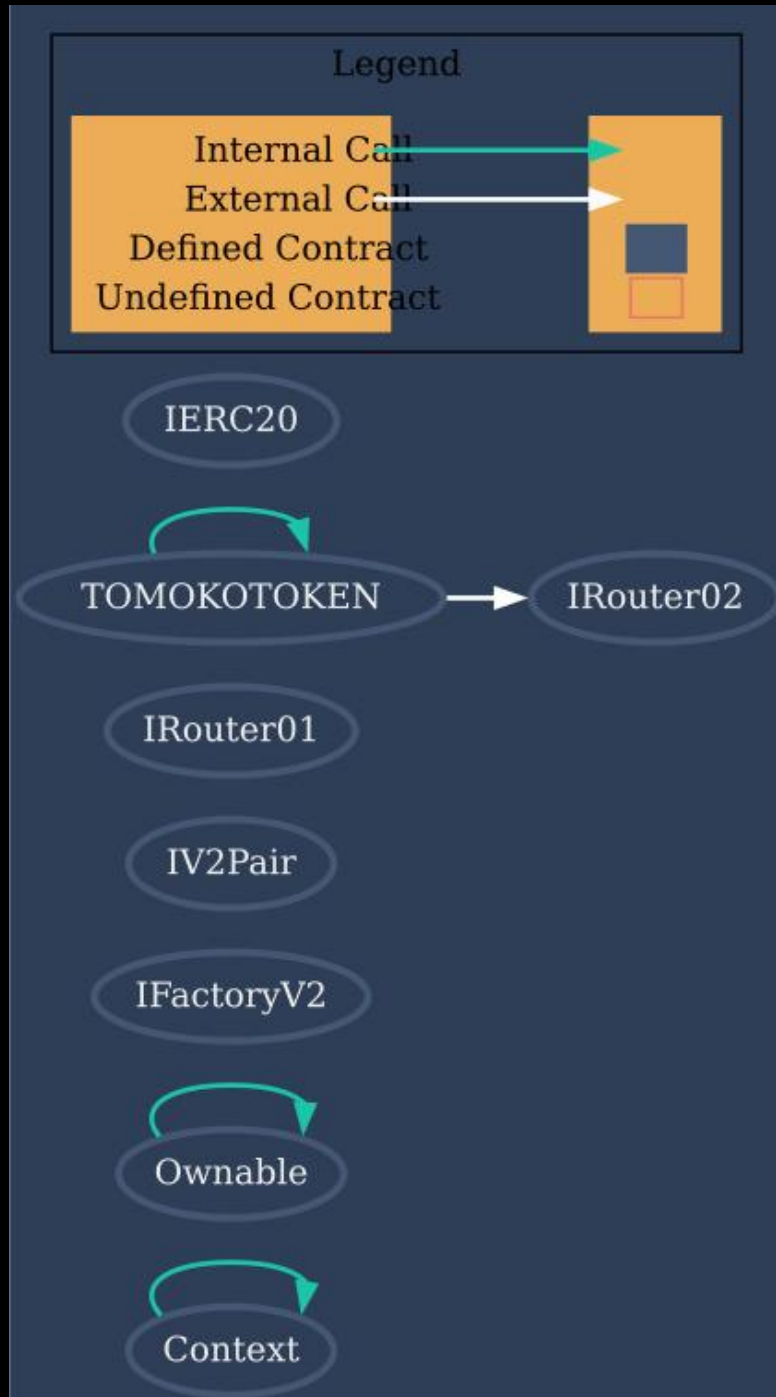
257	fttrace   funcSig
258	function approve(address spender!, uint256 amount!) external override returns (bool) {
259	_approve(msg.sender, spender!, amount!);
260	return true;
261	}

The diagram illustrates the dependencies of the TOMOKOTO token contract. It is organized into several main sections:

- Legend:** Defines the color coding for nodes: Internal Contract (red), External Contract (yellow), Defined Contract (green), and Undefined Contract (blue).
- IERC20 (iface):** A list of standard ERC20 functions including `totalSupply`, `decimals`, `symbol`, `name`, `getOwner`, `balanceOf`, `transfer`, `allowance`, `approve`, and `transferFrom`.
- IRouter01 (iface):** Functions for token swapping, including `factory`, `WETH`, `addLiquidityETH`, `addLiquidity`, `swapExactETHForTokens`, `getAmountsOut`, and `getAmountsIn`.
- IV2Pair (iface):** Functions for managing liquidity pairs, including `factory`, `getReserves`, and `sync`.
- IFactoryV2 (iface):** Functions for creating pairs, including `getPair` and `createPair`.
- Ownable:** Functions for managing ownership, including `renounceOwnership`, `transferOwnership`, `onlyOwner`, and `<Constructor>`.
- Context:** Functions for handling messages, including `<Constructor>`, `msgSender`, `msgData`, and `payable`.
- TOMOKOTO token:** The main contract, which includes functions like `approve`, `isLimitedAddress`, `balanceOf`, `internalSwap`, `swapAndLiquify`, `takeTaxes`, `canSwap`, `is_sell`, `is_buy`, `type`, `transfer`, `transferFrom`, `isNoFeeWallet`, `setNoFeeWallet`, `changeLpPair`, `toggleCanSwapFees`, `changeWallets`, `setPresaleAddress`, and `enableTrading`.

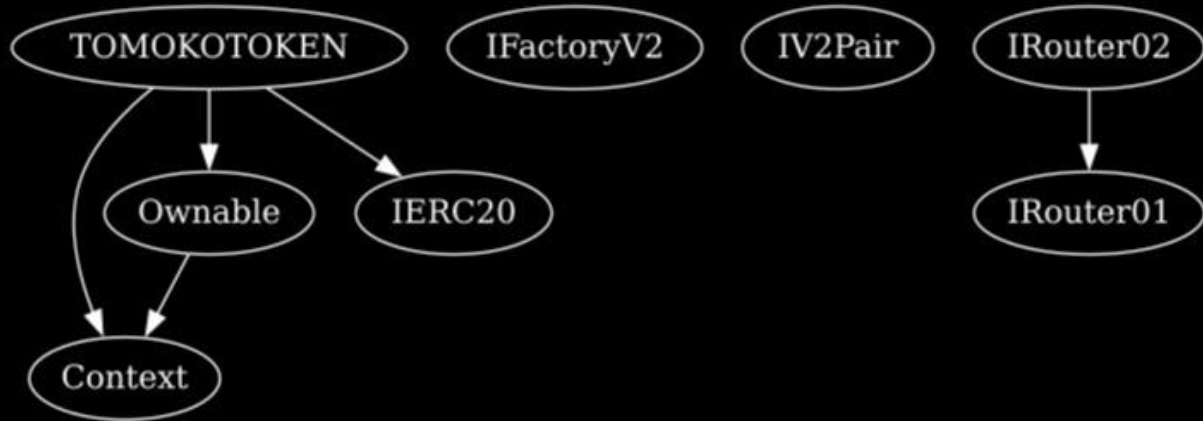
Arrows indicate the flow of dependencies, showing how the TOMOKOTO token contract relies on these external interfaces and internal functions.

## Contract Interaction Graph

















































































































## Inheritance Graph






















## Contract Functions

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
<b>Context</b>	Implementation			
L		Public 		NO 
L	_msgSender	Internal 		
L	_msgData	Internal 		
<b>Ownable</b>	Implementation	Context		
L		Public 		NO 
L	owner	Public 		NO 
L	renounceOwnership	Public 		onlyOwner
L	transferOwnership	Public 		onlyOwner
L	_setOwner	Private 		
<b>IFactoryV2</b>	Interface			
L	getPair	External 		NO 
L	createPair	External 		NO 
<b>IV2Pair</b>	Interface			
L	factory	External 		NO 
L	getReserves	External 		NO 
L	sync	External 		NO 

Contract	Type	Bases		
<b>IRouter01</b>	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 
<b>IRouter02</b>	Interface	IRouter01		
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External 		NO 
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External 		NO 
L	swapExactTokensForTokensSupportingFeeOnTransferTokens	External 		NO 
L	swapExactTokensForTokens	External 		NO 
<b>IERC20</b>	Interface			
L	totalSupply	External 		NO 
L	decimals	External 		NO 

Contract	Type	Bases		
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 
<b>TOMOKOTOKE N</b>	Implementation	Context, Ownable, IERC20		
L	totalSupply	External 		NO 
L	decimals	External 		NO 
L	symbol	External 		NO 
L	name	External 		NO 
L	getOwner	External 		NO 
L	allowance	External 		NO 
L	balanceOf	Public 		NO 
L		Public 		NO 
L		External 		NO 
L	transfer	Public 		NO 
L	approve	External 		NO 
L	_approve	Internal 		

Contract	Type	Bases		
L	transferFrom	External !		NO!
L	isNoFeeWallet	External !		NO!
L	setNoFeeWallet	Public !		onlyOwner
L	isLimitedAddresses	Internal 		
L	is_buy	Internal 		
L	is_sell	Internal 		
L	canSwap	Internal 		
L	changeLpPair	External !		onlyOwner
L	toggleCanSwap Fees	External !		onlyOwner
L	_transfer	Internal 		
L	changeWallets	External !		onlyOwner
L	takeTaxes	Internal 		
L	swapAndLiquify	Internal 		inSwapFlag
L	internalSwap	Internal 		inSwapFlag
L	setPresaleAddresses	External !		onlyOwner
L	enableTrading	External !		onlyOwner



Function  
can modify  
state



Function  
is payable

## 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 vulnerabilities in the code. Findings getting reported and improvements getting suggested.

### Automatic and Manual Review

We are using automated tools to scan functions and weaknesses 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>

