

# SKELETON ECOSYSTEM

SMART CONTRACT AUDIT



**Garble**  
**GARBLE**  
**BEP20**

0x808a6293098b3ac6686317f8e771438995b5f



## 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	8
Detected Vulnerability Description	12
Contract Flow Graph	15
Contract Interaction Graph	16
Inheritance Graph	17
Contract Descriptions	18
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	GarbleMemeToken
Ticker/Symbol	GARBLE
Blockchain	Binance Smart Chain BEP20
Contract Address	0x808a6293098b3ac6686317f8e771438995b5f19a
Creator Address	0x782Ca967aA2E33b0f183757099c210783384a01C
Current Owner Address	0x00
Contract Explorer	<a href="https://bscscan.com/address/0x808a6293098b3ac6686317f8e771438995b5f19a#code">https://bscscan.com/address/0x808a6293098b3ac6686317f8e771438995b5f19a#code</a>
Compiler Version	v0.8.24+commit.e11b9ed9
License	london EvmVersion, Apache-2.0 <a href="#">license</a>
Optimisation	No with 200 Runs
Total Supply	100,000,000,000,000,000 <b>GARBLE</b>
Decimals	9



## Creation/Audit

Contract Deployed	10.02.2024
Audit Created	11.02.2024
Audit Update	V 1.0

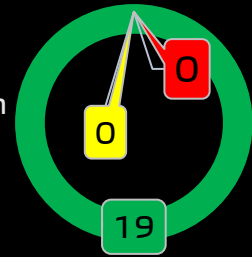
## Verified Socials

Website	<a href="https://garble.lol/">https://garble.lol/</a>
Telegram	<a href="https://t.me/GarbleToken">https://t.me/GarbleToken</a>
Twitter (X)	<a href="https://twitter.com/GarbleToken">https://twitter.com/GarbleToken</a>








## Contract Function Analysis

 Pass
  Attention Item
  Risky Item

■ Pass  
 ■ Attention  
 ■ Risk



Contract Verified		The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership		0x00 Sometimes referred to as the "zero address" or "dead address" and is not owned by anyone.
Buy Tax	1 %	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	1 %	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 status on 11.02.2024 100.00% Pinklock for 324 days.
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.
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. Remove all limits function 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 with authorisation</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>Max Transaction and Holding Modify function found. Remove all limits function triggered</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

### Removing Risk of contract function based on renounced ownership

Transaction Receipt Event Logs

54

**Address** 0x808a6293098b3ac6686317f8e771438995b5f19a  

**Name** OwnershipTransferred (index\_topic\_1 address previousOwner, index\_topic\_2 address newOwner) [View Source](#)

**Topics**

0

0x8be0079c531659141344cd1fd0a4f28419497f9722a3daafe3b4186f6b6457e0

1: previousOwner

Dec ▾

→ 0x782Ca967aA2E33b0f183757099c210783384a01C

2: newOwner

Dec ▾

→ 0x00

**Data** 0x

Following detected contract functions serve as informational purposes about the contract. The owner has no more authorisation to trigger the following functions.

Remove all limits function triggered. Max wallet and transaction = Total Supply

```
Address    0x808a6293098b3ac6686317f8e771438995b5f19a [Q] v  
Name      MaxTxAmountUpdated (uint256 _maxTxAmount) View Source  
Topics    0   0x947f344d56e1e8c70dc492fb94cd4ddddd490cc16aab685f57e47b2ce85cb44cf  
Data      _maxTxAmount : 10000000000000000000000000000000000
```

```

300 function removeAllLimits() external onlyOwner{
301     maxTxAmount = _totalSupply;
302     maxWalletAmount = _totalSupply;
303     emit MaxTxAmountUpdated(_totalSupply);
304 }

```



## Contract Security

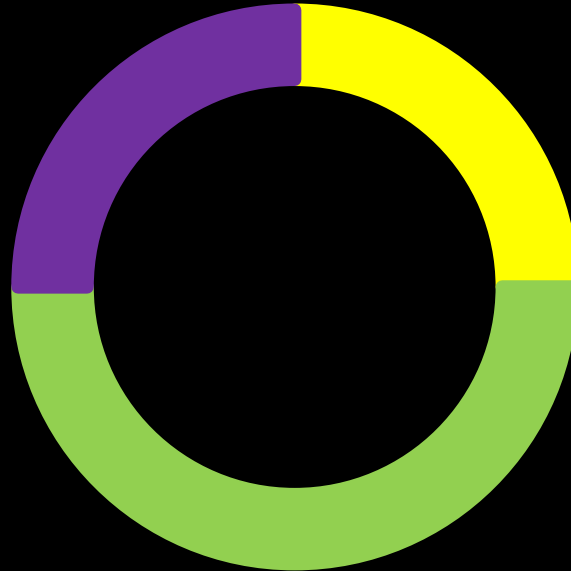
Total Findings: 4

■ High 0

■ Medium 1

■ Low 2

■ Info 1



■ **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: (1)

- Authorization through tx.origin

#### Low severity issues: (2)

- Long number literals
- Approve Front Running Attack (Sandwich Bots)

#### Informational severity issues: (1)

- 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

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	High	Medium	Medium
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.

### ⚠️ Authorization through tx.origin (2 Items)

Item: 1	Location:	Line 243	Severity:	■ Medium
Item: 2	Location:	Line 247	Severity:	■ Medium

<b>Function</b>	In Solidity, tx.origin is a global variable that returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable. For example, if an authorized account calls a malicious contract which triggers it to call the vulnerable contract that passes an authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.
<b>Remediation</b>	tx.origin should not be used for authorization in smart contracts. It does have some legitimate use cases, for example, To prevent external contracts from calling the current contract, you can implement a require of the form require(tx.origin == msg.sender). This prevents intermediate contracts from calling the current contract, thus limiting the contract to regular codeless addresses.

```

242     require(
243         holderLastTransferTimestamp[tx.origin] <
244         block.number,
245         "_transfer:: Transfer Delay enabled. Only one purchase per block allowed."
246     );
247     holderLastTransferTimestamp[tx.origin] = block.number;
  
```

⚠️ Approve of front running attack. Also known as Sandwich Bot attack. (2 Item)

Item: 1	Location:	Line 214-217	Severity:	■ Low
---------	-----------	--------------	-----------	-------

<b>Function</b>	<p>The <code>approve()</code> 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. The function approve can be front-run by abusing the <code>_approve</code> 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>

```

214  function approve(address spender!, uint256 amount!) public override returns (bool) {
215      _approve(_msgSender(), spender!, amount!);
216      return true;
217  }
218

```

Item: 2	Location:	Line 285-297	Severity:	■ Low
---------	-----------	--------------	-----------	-------

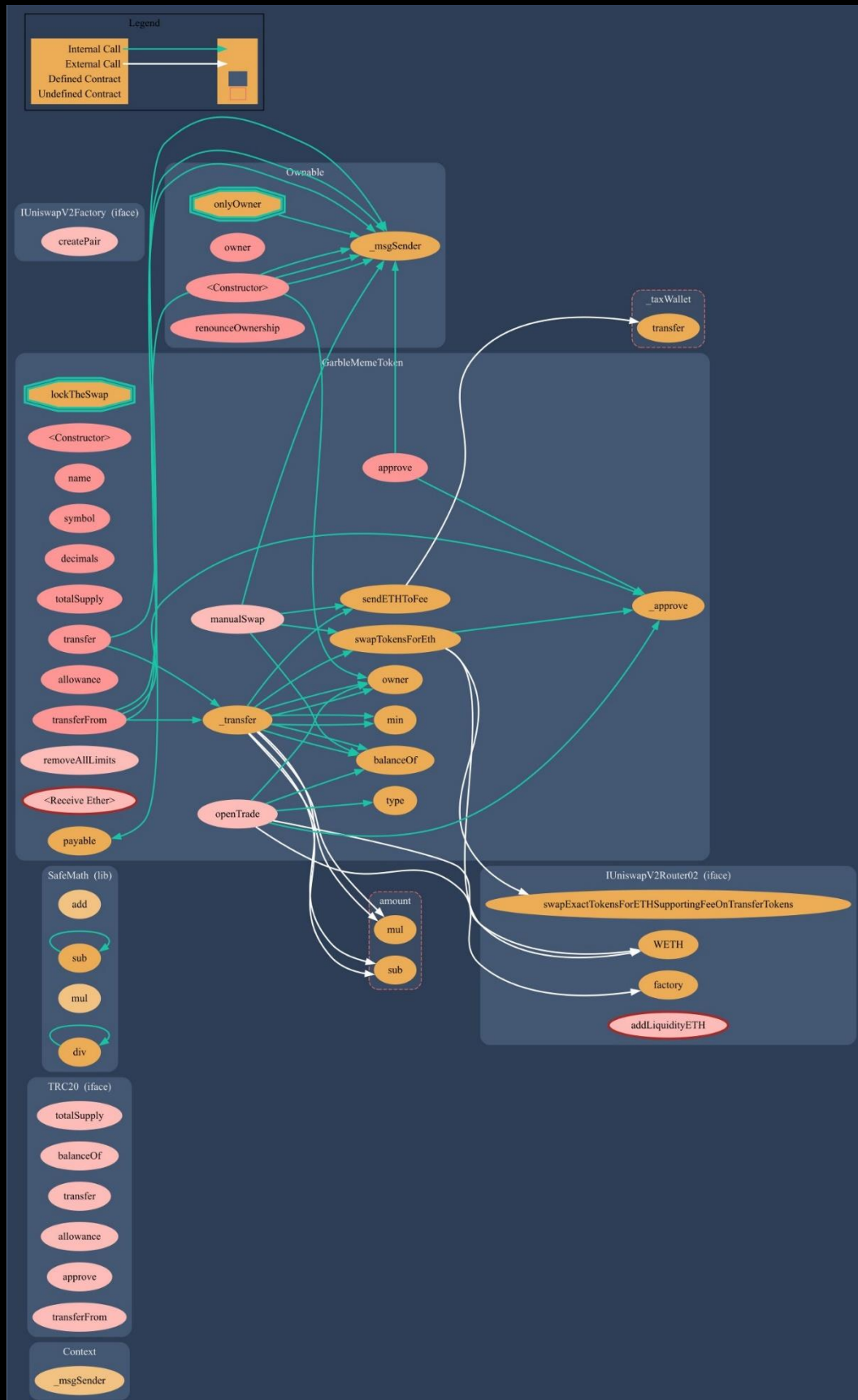
<b>Function</b>	<p>The <code>swapTokensForEth()</code> method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account.</p> <p>This 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. The function <code>swapTokensForEth</code> can be front-run by abusing the <code>_approve</code> 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-runattack</li> </ol>

```

285     function swapTokensForEth(uint256 tokenAmount) private lockTheSwap {
286         address[] memory path = new address[](2);
287         path[0] = address(this);
288         path[1] = uniswapV2Router.WETH();
289         _approve(address(this), address(uniswapV2Router), tokenAmount);
290         uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
291             tokenAmount,
292             0,
293             path,
294             address(this),
295             block.timestamp
296         );
297     }
298

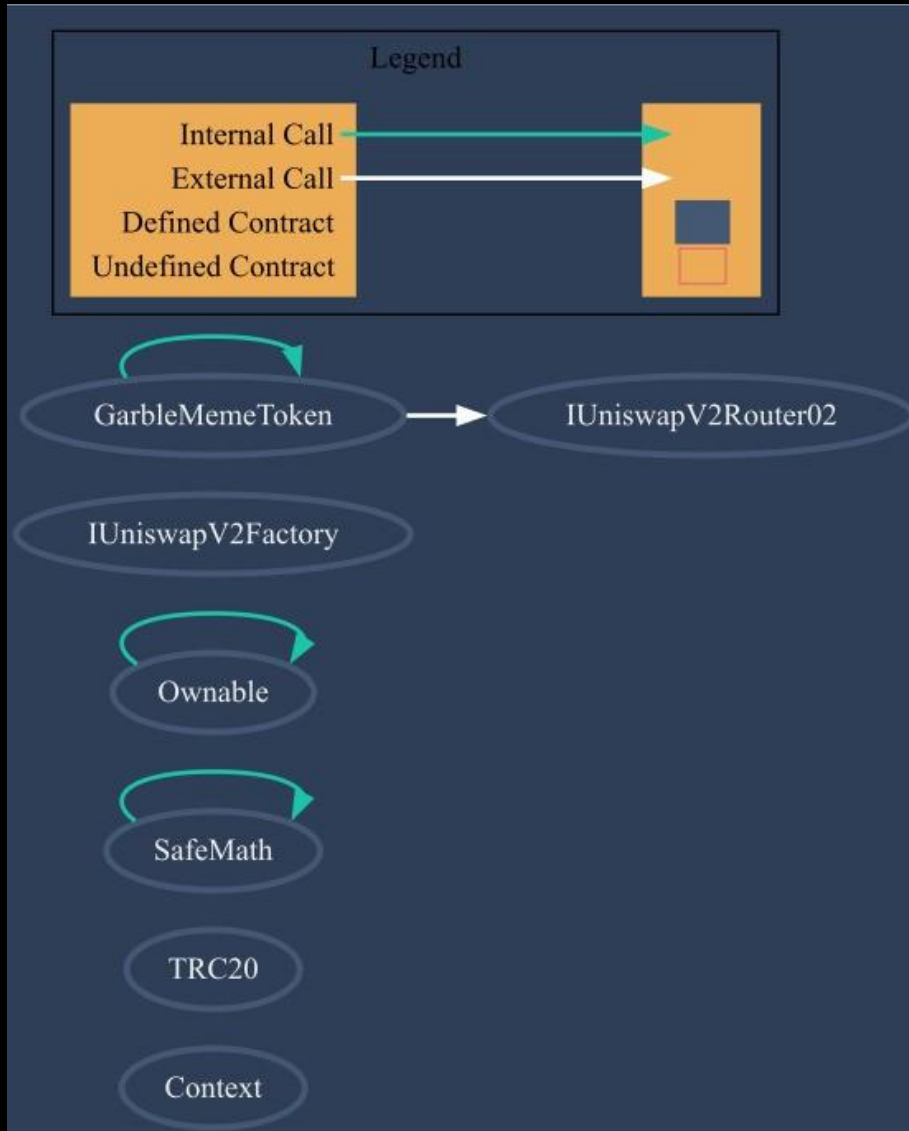
```

## Contract Flow Graph

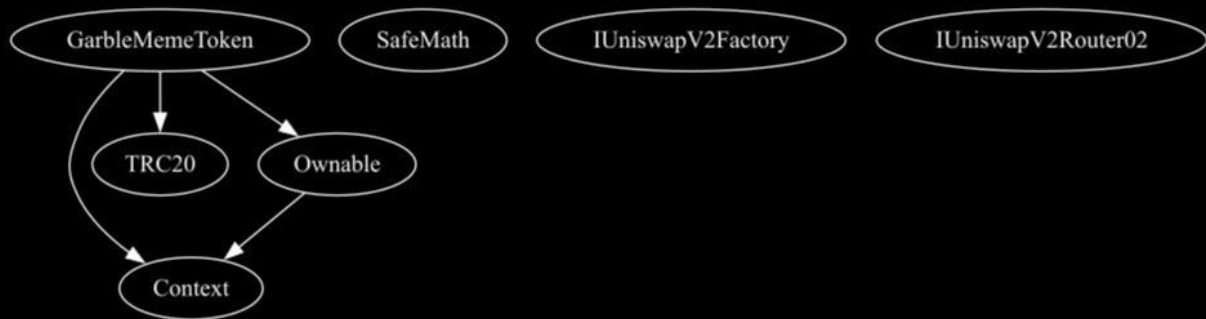































## Contract Interaction Graph





















































## Inheritance Graph



## Contract Functions

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
<b>Context</b>	Implementation			
L	_msgSender	Internal 		
<b>TRC20</b>	Interface			
L	totalSupply	External 		NO 
L	balanceOf	External 		NO 
L	transfer	External 		NO 
L	allowance	External 		NO 
L	approve	External 		NO 
L	transferFrom	External 		NO 
<b>SafeMath</b>	Library			
L	add	Internal 		
L	sub	Internal 		
L	sub	Internal 		
L	mul	Internal 		
L	div	Internal 		
L	div	Internal 		
<b>Ownable</b>	Implementation	Context		
L		Public 		NO 
L	owner	Public 		NO 

Contract	Type	Bases		
L	renounceOwnership	Public 		onlyOwner
<b>IUniswapV2Factory</b>	Interface			
L	createPair	External 		NO 
<b>IUniswapV2Router02</b>	Interface			
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External 		NO 
L	factory	External 		NO 
L	WETH	External 		NO 
L	addLiquidityETH	External 		NO 
<b>GarbleMemeToken</b>	Implementation	Context, TRC20, Ownable		
L		Public 		NO 
L	name	Public 		NO 
L	symbol	Public 		NO 
L	decimals	Public 		NO 
L	totalSupply	Public 		NO 
L	balanceOf	Public 		NO 
L	transfer	Public 		NO 
L	allowance	Public 		NO 
L	approve	Public 		NO 

Contract	Type	Bases		
L	transferFrom	Public !		NO!
L	_approve	Private 		
L	_transfer	Private 		
L	min	Private 		
L	swapTokensForEth	Private 		lockTheSwap
L	removeAllLimits	External !		onlyOwner
L	sendETHToFee	Private 		
L	openTrade	External !		onlyOwner
L		External !		NO!
L	manualSwap	External !		NO!



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>

