SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

Grokie Inu
GROKIE
BEP 20

0x7Dcec4FFa9A0adbE1C207F283E4f46C704D506ba

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 Desciptions	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% 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	REFLECTIONS_TOKEN
Ticker/Simbol	GROKIE
Blockchain	Binance Smart Chain BEP20
Contract Address	0x7Dcec4FFa9A0adbE1C207F283E4f46C704D506ba
Creator Address	0xE421486468C6b5Fe293fE0d6138a612c29decA9E
Current Owner Address	0x0000000000000000000000000000000000000000
Contract Explorer	https://bscscan.com/token/0x7dcec4ffa9a0adbe1c207f283e4f46c704d506ba#code
Compiler Version	v0.8.19+commit.7dd6d404
License	Unlicense
Optimisation	Yes with 200 Runs
Total Supply	993,500,000 **GROKIE**
Decimals	18

Creation/Audit

Contract Deployed	01 Dec 2023
Audit Created	10 Dec 2023
Audit Update	V 1.0

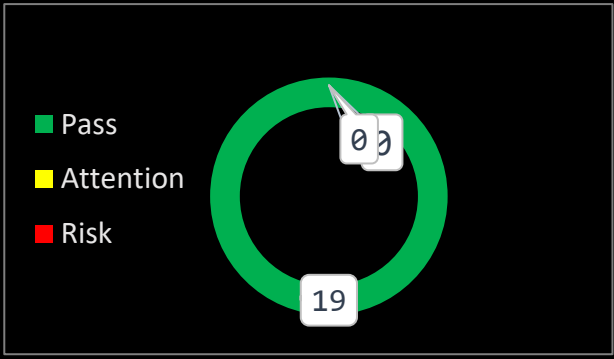
Verified Socials

Website	https://grokieinu.com/
Telegram	https://t.me/grokieinu
Twitter (X)	https://twitter.com/grokieinu

GROKIE INU 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 0x0000000000000000000000000000000000000000
Buy Tax	4 %	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	4 %	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
Liqudity Status	✅	LP Lock Status on 09.12.2023: 99.41% Pinklock for 388 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	⚠️✅	Fee Setting function found, but contract is renounced, this function can not be triggered. 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	✅	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	✅	No Blacklist Setting function found. 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 Setting function found, but 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. Can cause farming)
Hidden Owner Analysis	✅	No Hidden or multi owner with authorisation For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned.
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	⚠️✅	Max Transaction and Holding Modify function found, but contract is renounced, this function can not be triggered. 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)

Details of Risk - Attention Items

⚠ Set Fee

Risk Removed → Renounced Contract!

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

```
ftrace | funcSig
function Set_Fees(

    uint8 Marketing_on_BUY!,
    uint8 Liquidity_on_BUY!,
    uint8 Reflection_on_BUY!,
    uint8 Burn_on_BUY!,

    uint8 Marketing_on_SELL!,
    uint8 Liquidity_on_SELL!,
    uint8 Reflection_on_SELL!,
    uint8 Burn_on_SELL!

    ) external onlyOwner {

    // Buyer Protection: Max Fee 15%
    require (Marketing_on_BUY! + Liquidity_on_BUY! + Reflection_on_BUY! + Burn_on_BUY! <= 15, "FEE1");  // Max fee 15%

    // Buyer Protection: Max Fee 15%
    require (Marketing_on_SELL! + Liquidity_on_SELL! + Reflection_on_SELL! + Burn_on_SELL! <= 15, "FEE2");  // Max fee 15%

    // Update Fees
    _fee__Buy_Marketing   = Marketing_on_BUY!;
    _fee__Buy_Liquidity   = Liquidity_on_BUY!;
    _fee__Buy_Reflection  = Reflection_on_BUY!;
    _fee__Buy_Burn        = Burn_on_BUY!;

    _fee__Sell_Marketing   = Marketing_on_SELL!;
    _fee__Sell_Liquidity   = Liquidity_on_SELL!;
    _fee__Sell_Reflection  = Reflection_on_SELL!;
    _fee__Sell_Burn        = Burn_on_SELL!;

    // Fees For Processing
    _SwapFeeTotal_Sell   = _fee__Sell_Marketing + _fee__Sell_Liquidity;
    _SwapFeeTotal_Buy    = _fee__Buy_Marketing + _fee__Buy_Liquidity;

    emit updated_Buy_fees(_fee__Buy_Marketing, _fee__Buy_Liquidity, _fee__Buy_Reflection, _fee__Buy_Burn);
    emit updated_Sell_fees(_fee__Sell_Marketing, _fee__Sell_Liquidity, _fee__Sell_Reflection, _fee__Sell_Burn);
```

⚠ Whitelist Function

Risk Removed → Renounced Contract!

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)

```
// Exclude From Fees
ftrace | funcSig
function Wallet_Exclude_From_Fees(

    address Wallet_Address,
    bool true_or_false

    ) external onlyOwner {
    _isExcludedFromFee[Wallet_Address] = true_or_false;

}
```

⚠ Max Transaction and Holding Modify Function

Risk Removed → Renounced Contract!

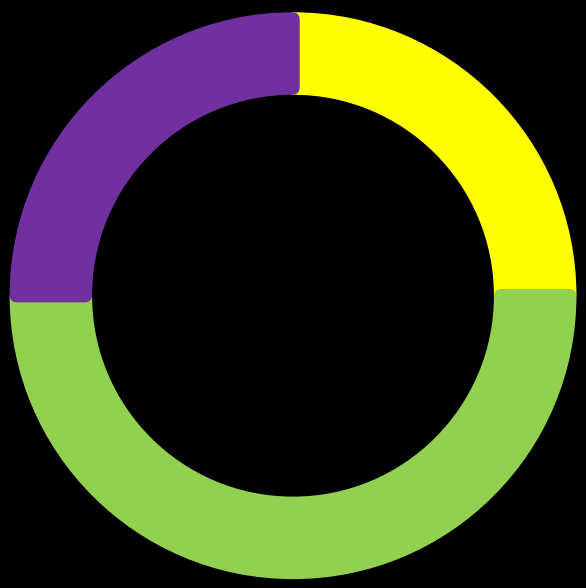
If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot

```
ftrace | funcSig
function Set_Wallet_Limits(

    uint256 Max_Transaction_Percent,
    uint256 Max_Wallet_Percent

    ) external onlyOwner {

    if (Max_Transaction_Percent < 1){

        // Defaults to 0.5% if 0 is entered
        max_Tran = _tTotal / 200;

    } else {

        max_Tran = _tTotal * Max_Transaction_Percent / 100;

    }


    if (Max_Wallet_Percent < 1){

        // Defaults to 0.5% if 0 is entered
        max_Hold = _tTotal / 200;

    } else {

        max_Hold = _tTotal * Max_Wallet_Percent / 100;
```

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)

- **Approve of Front running Attack**

Low severity issues: (2)

- Missing Events
- Long Number Literals

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 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	**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.

⚠ Approve of front running attack (3 Item)

Item: 1	Location:	Line 767-770	Severity:	Medium

Function	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. The function approve can be front-run by abusing the _approve function.
Remedation	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. 2.Use transaction taxes to prevent against front-runattack

```
ftrace | funcSig
767 function approve(address spender, uint256 amount) public override returns (bool) {
768     _approve(_msgSender(), spender, amount);
769     return true;
770 }
771
```

Item: 2	Location:	Line 809-817	Severity:	Medium

Function	The transferFrom() 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. The function approve can be front-run by abusing the _approve function.
Remedation	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. 2.Use transaction taxes to prevent against front-runattack

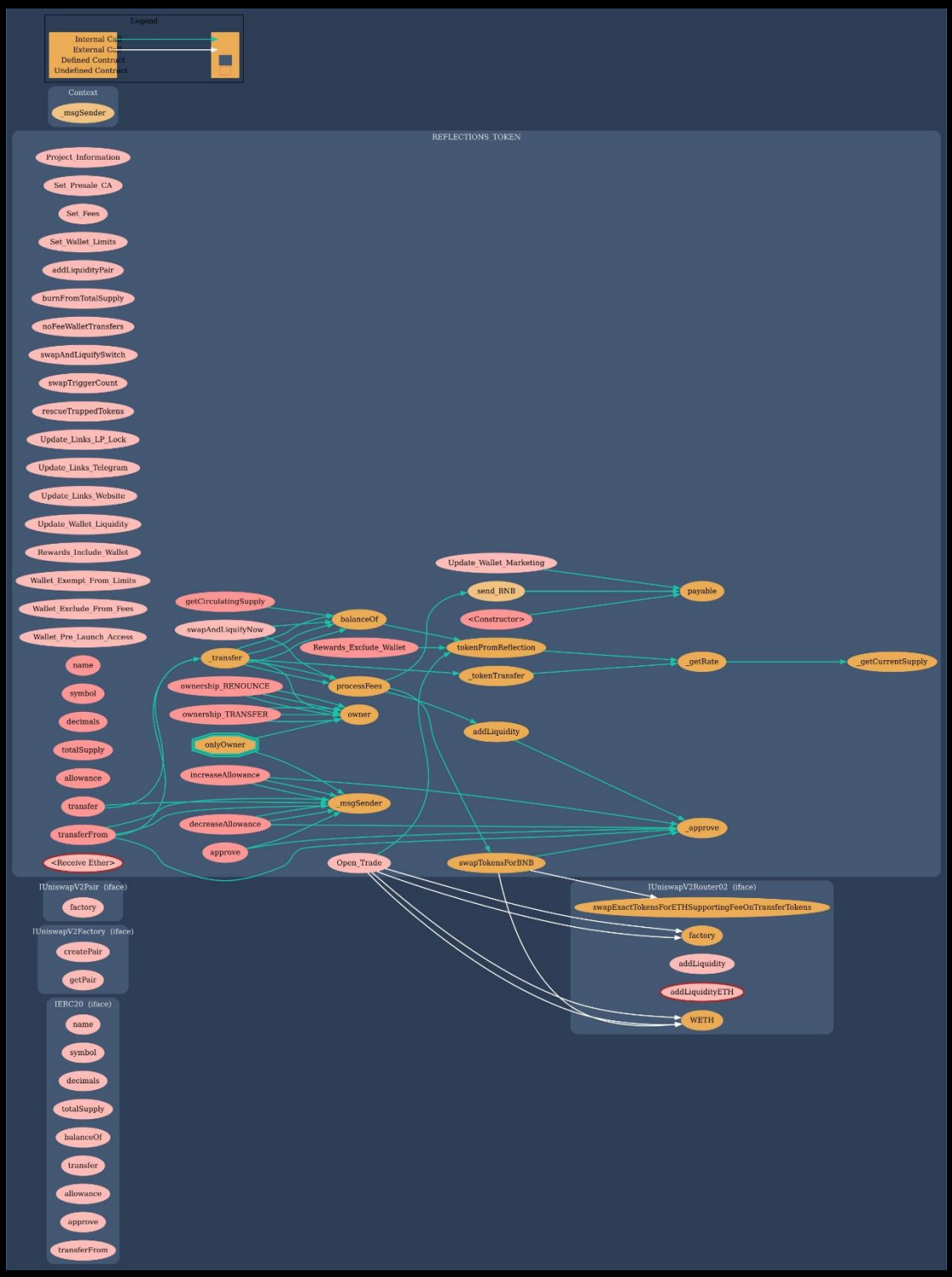
```
808
        ftrace | funcSig
809     function transferFrom(address sender, address recipient, uint256 amount) public virtual override returns (bool) {
810         _transfer(sender, recipient, amount);
811
812         uint256 currentAllowance = _allowances[sender][_msgSender()];
813         require(currentAllowance >= amount, "ERC20: transfer amount exceeds allowance");
814         _approve(sender, _msgSender(), currentAllowance - amount);
815
816         return true;
817     }
818
```

Item: 3	Location:	Line 981-994	Severity:	Medium

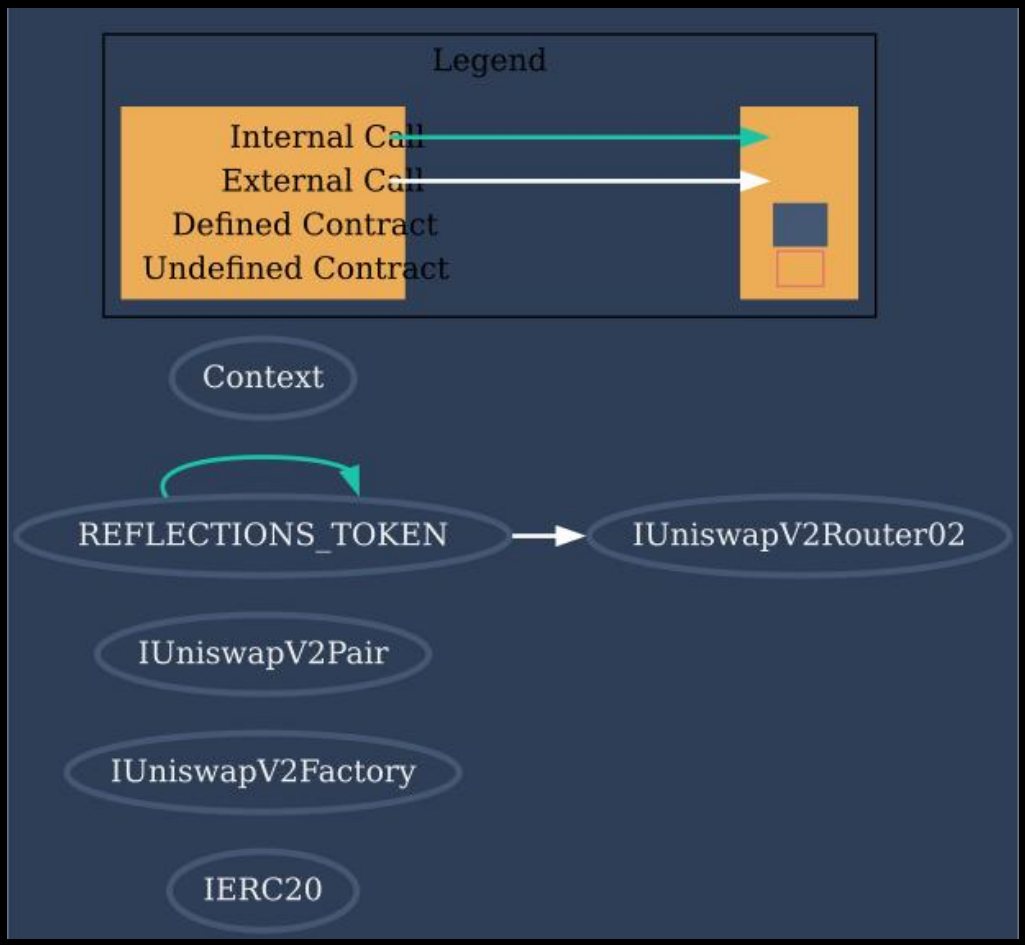
Function	The _swapTokensForBNB method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver 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. The function approve can be front-run by abusing the _approve function.
Remedation	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. 2.Use transaction taxes to prevent against front-runattack

```
981     function swapTokensForBNB(uint256 tokenAmount) private {
982
983         address[] memory path = new address[](2);
984         path[0] = address(this);
985         path[1] = uniswapV2Router.WETH();
986         _approve(address(this), address(uniswapV2Router), tokenAmount);
987         uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
988             tokenAmount,
989             0,
990             path,
991             address(this),
992             block.timestamp
993         );
994     }
995
```

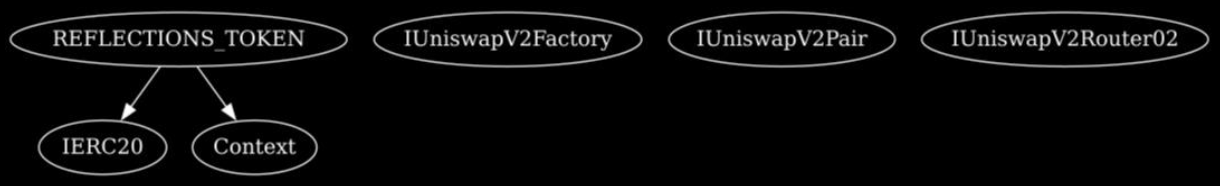
Contract Flow Graph



Contract Interaction Graph



Inheritance Graph



Contract Functions

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
IERC20	Interface			
L	name	External !		NO!
L	symbol	External !		NO!
L	decimals	External !		NO!
L	totalSupply	External !		NO!
L	balanceOf	External !		NO!
L	transfer	External !		NO!
L	allowance	External !		NO!
L	approve	External !		NO!
L	transferFrom	External !		NO!
IUniswapV2Factory	Interface			
L	createPair	External !		NO!
L	getPair	External !		NO!
IUniswapV2Pair	Interface			
L	factory	External !		NO!
IUniswapV2Router02	Interface			
L	factory	External !		NO!

Contract	Type	Bases		
L	WETH	External		NO!
L	addLiquidity	External		NO!
L	addLiquidityETH	External		NO!
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External		NO!
Context	Implementation			
L	_msgSender	Internal		
REFLECTIONS_TOKEN	Implementation	Context, IERC20		
L		Public		NO!
L	Project_Information	External		NO!
L	Set_Presale_CA	External		onlyOwner
L	Set_Fees	External		onlyOwner
L	Set_Wallet_Limits	External		onlyOwner
L	Open_Trade	External		onlyOwner
L	addLiquidityPair	External		onlyOwner
L	burnFromTotalSupply	External		onlyOwner
L	noFeeWalletTransfers	External		onlyOwner
L	swapAndLiquifySwitch	External		onlyOwner

Contract	Type	Bases		
L	swapTriggerCount	External !		onlyOwner
L	swapAndLiquifyNow	External !		onlyOwner
L	rescueTrappedTokens	External !		onlyOwner
L	Update_Links_LP_Lock	External !		onlyOwner
L	Update_Links_Telegram	External !		onlyOwner
L	Update_Links_Website	External !		onlyOwner
L	Update_Wallet_Liquidity	External !		onlyOwner
L	Update_Wallet_Marketing	External !		onlyOwner
L	Rewards_Exclude_Wallet	Public !		onlyOwner
L	Rewards_Include_Wallet	External !		onlyOwner
L	Wallet_Exempt_From_Limits	External !		onlyOwner
L	Wallet_Exclude_From_Fees	External !		onlyOwner
L	Wallet_Pre_Launch_Access	External !		onlyOwner
L	ownership_RENOUNCE	Public !		onlyOwner
L	ownership_TRANSFER	Public !		onlyOwner

Contract	Type	Bases		
L	owner	Public		NO!
L	name	Public		NO!
L	symbol	Public		NO!
L	decimals	Public		NO!
L	totalSupply	Public		NO!
L	balanceOf	Public		NO!
L	allowance	Public		NO!
L	increaseAllowance	Public		NO!
L	decreaseAllowance	Public		NO!
L	approve	Public		NO!
L	_approve	Private		
L	tokenFromReflection	Internal		
L	_getRate	Private		
L	_getCurrentSupply	Private		
L	transfer	Public		NO!
L	transferFrom	Public		NO!
L	send_BNB	Internal		
L	getCirculatingSupply	Public		NO!
L	_transfer	Private		
L	processFees	Private		

Contract	Type	Bases		
L	swapTokensFor BNB	Private		
L	addLiquidity	Private		
L	_tokenTransfer	Private		
L		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 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