SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

Pinto Inu [$PINTO]

BEP 20

0x9225Ebb553463Cd94204fE6e1AB035Fd4476FBE4

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	Pinto Inu
Ticker/Simbol	$PINTO
Blockchain	Binance Smart Chain BEP20
Contract Address	0x9225Ebb553463Cd94204fE6e1AB035Fd4476FBE4
Creator Address	0x8A01795e7d22389C7da4D5Bec6D98E026B5391f8
Current Owner Address	Renounced
Contract Explorer	https://bscscan.com/token/0x9225ebb553463cd94204fe6e1ab035fd4476fbe4
Compiler Version	v0.8.17+commit.8df45f5f
License	MIT
Optimisation	Yes with 1500 Runs
Total Supply	1,000,000 **PINTO**
Decimals	18

Creation/Audit

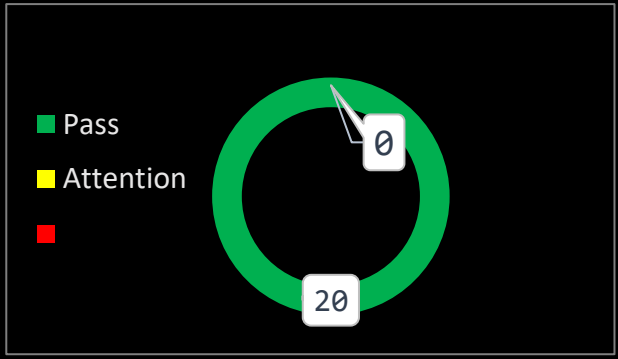
Contract Deployed	17 Oct 2023
Audit Created	26 Oct 2023
Audit Update	V 1.0

Verified Socials

Website	https://pintoinu.fun/
Telegram	https://t.me/pintoinubsc
Twitter (X)	https://x.com/pintoinubsc?s=21

SKELETON ECOSYSTEM
SMART CONTRACT AUDIT REPORT

PINTO INU BEP20



Contract Function Analysis

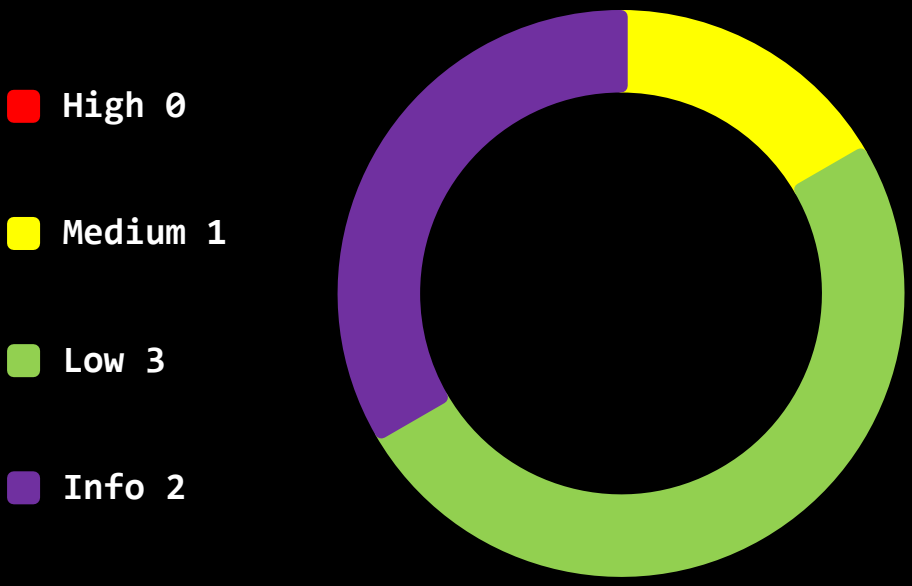
Pass Attention Item Risky Item

Contract Verified	✓	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership	✓	Renounced
Buy Tax	10 %	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	10 %	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 17.10.2023: 99% for 32 Days on Pinksale Locker
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 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 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 Renounced, this function can to 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 Multi Owner 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: 5



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 Front Running Attack

Low severity issues: (3)

- Missing Events
- Long Number Literals
- 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	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. Example Sandwitch bots (2 Items)

Item: 1	Location:	Line 91-95	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-run attack

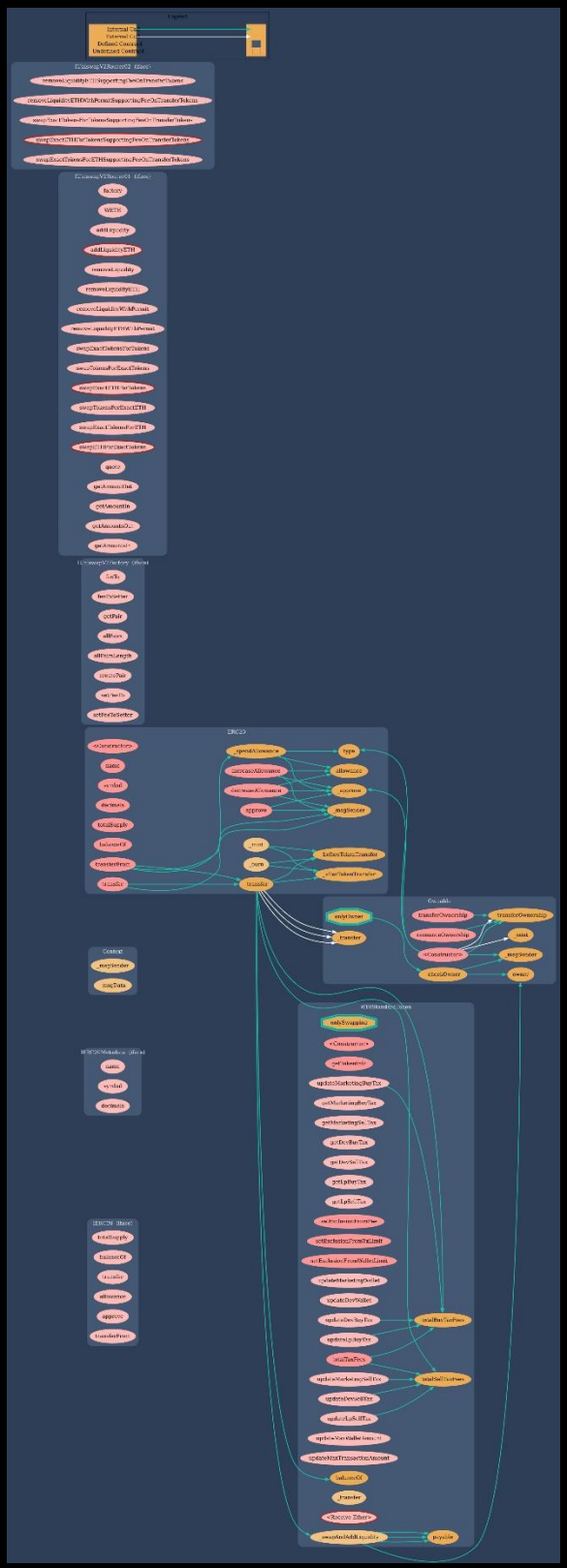
```
ftrace | funcSig
91  function approve(address spender, uint256 amount) public virtual override returns (bool) {
92      address owner = _msgSender();
93      _approve(owner, spender, amount);
94      return true;
95  }
96
```

Item: 2	Location:	Line 189-197	Severity:	Medium

Function	The `_spendAllowance()` 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 _spendAllowance 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-run attack

```
ftrace | funcSig
189     function _spendAllowance(address owner, address spender, uint256 amount) internal virtual {
190         uint256 currentAllowance = allowance(owner, spender);
191         if (currentAllowance != type(uint256).max) {
192             require(currentAllowance >= amount, "ERC20: insufficient allowance");
193             unchecked {
194                 _approve(owner, spender, currentAllowance - amount);
195             }
196         }
197     }
198
```

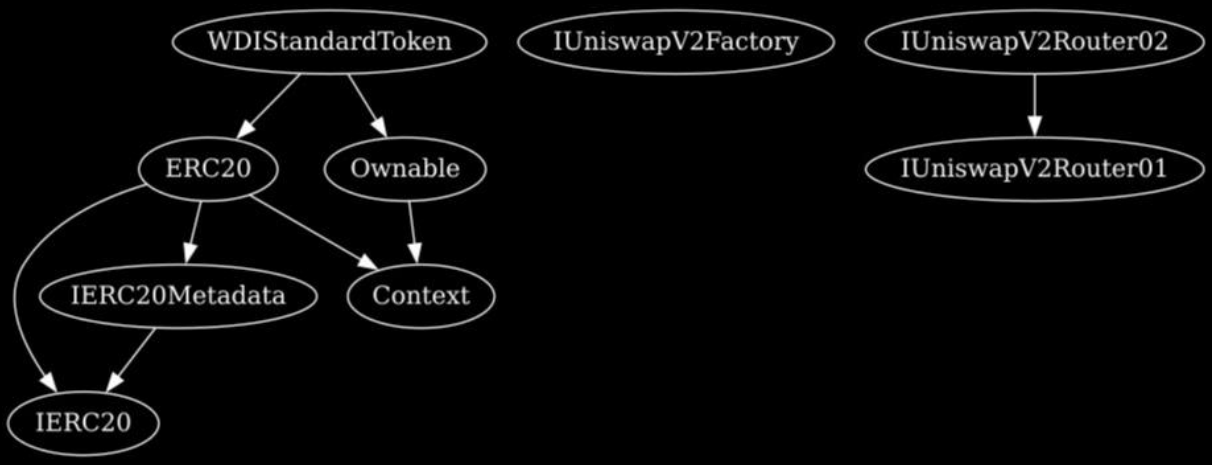
Contract Flow Graph



Contract Interaction Graph


Legend
Internal Call
External Call
Defined Contract
Undefined Contract
WDIStandardToken
Ownable
IUniswapV2Router02
IUniswapV2Router01
IUniswapV2Factory
ERC20
Context
IERC20Metadata
IERC20

Inheritance Graph



Contract Functions

Contract	Type	Bases		
L	**Function Name**	**Visibility**	**Mutability**	**Modifiers**
IERC20	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❗
IERC20Metadata	Interface	IERC20		
L	name	External ❗		NO❗
L	symbol	External ❗		NO❗
L	decimals	External ❗		NO❗
Context	Implementation			
L	_msgSender	Internal 🔒		
L	_msgData	Internal 🔒		
ERC20	Implementation	Context, IERC20, IERC20Metadata		
L		Public ❗	🛑	NO❗

Contract	Type	Bases		
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❗
L	transferFrom	Public ❗	🛑	NO❗
L	increaseAllowance	Public ❗	🛑	NO❗
L	decreaseAllowance	Public ❗	🛑	NO❗
L	_transfer	Internal 🔒	🛑	
L	_mint	Internal 🔒	🛑	
L	_burn	Internal 🔒	🛑	
L	_approve	Internal 🔒	🛑	
L	_spendAllowance	Internal 🔒	🛑	
L	_beforeTokenTransfer	Internal 🔒	🛑	
L	_afterTokenTransfer	Internal 🔒	🛑	
IUniswapV2Factory	Interface			

Contract	Type	Bases		
L	feeTo	External ❗		NO❗
L	feeToSetter	External ❗		NO❗
L	getPair	External ❗		NO❗
L	allPairs	External ❗		NO❗
L	allPairsLength	External ❗		NO❗
L	createPair	External ❗	🛑	NO❗
L	setFeeTo	External ❗	🛑	NO❗
L	setFeeToSetter	External ❗	🛑	NO❗
IUniswapV2Router01	Interface			
L	factory	External ❗		NO❗
L	WETH	External ❗		NO❗
L	addLiquidity	External ❗	🛑	NO❗
L	addLiquidityETH	External ❗	💵	NO❗
L	removeLiquidity	External ❗	🛑	NO❗
L	removeLiquidityETH	External ❗	🛑	NO❗
L	removeLiquidityWithPermit	External ❗	🛑	NO❗
L	removeLiquidityETHWithPermit	External ❗	🛑	NO❗
L	swapExactTokensForTokens	External ❗	🛑	NO❗
L	swapTokensForExactTokens	External ❗	🛑	NO❗

Contract	Type	Bases		
L	swapExactETHForTokens	External ❗	💸	NO❗
L	swapTokensForExactETH	External ❗	🛑	NO❗
L	swapExactTokensForETH	External ❗	🛑	NO❗
L	swapETHForExactTokens	External ❗	💸	NO❗
L	quote	External ❗		NO❗
L	getAmountOut	External ❗		NO❗
L	getAmountIn	External ❗		NO❗
L	getAmountsOut	External ❗		NO❗
L	getAmountsIn	External ❗		NO❗
IUniswapV2Router02	Interface	IUniswapV2Router01		
L	removeLiquidityETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
L	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
L	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
L	swapExactETHForTokensSuppo	External ❗	💸	NO❗

Contract	Type	Bases		
	rtingFeeOnTransferTokens			
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
Ownable	Implementation	Context		
L		Public ❗	🛑	NO❗
L	owner	Public ❗		NO❗
L	_checkOwner	Internal 🔒		
L	renounceOwnership	Public ❗	🛑	onlyOwner
L	transferOwnership	Public ❗	🛑	onlyOwner
L	_transferOwnership	Internal 🔒	🛑	
WDIStandardToken	Implementation	ERC20, Ownable		
L		Public ❗	🛑	ERC20
L	getTokenInfo	Public ❗		NO❗
L	totalBuyTaxFees	Public ❗		NO❗
L	totalSellTaxFees	Public ❗		NO❗
L	totalTaxFees	Public ❗		NO❗
L	getMarketingBuyTax	External ❗		NO❗
L	getMarketingSellTax	External ❗		NO❗

Contract	Type	Bases		
L	getDevBuyTax	External ❗		NO❗
L	getDevSellTax	External ❗		NO❗
L	getLpBuyTax	External ❗		NO❗
L	getLpSellTax	External ❗		NO❗
L	setExclusionFromFee	Public ❗	🔘	onlyOwner
L	setExclusionFromTxLimit	Public ❗	🔘	onlyOwner
L	setExclusionFromWalletLimit	Public ❗	🔘	onlyOwner
L	updateMarketingWallet	External ❗	🔘	onlyOwner
L	updateDevWallet	External ❗	🔘	onlyOwner
L	updateMarketingBuyTax	External ❗	🔘	onlyOwner
L	updateMarketingSellTax	External ❗	🔘	onlyOwner
L	updateDevBuyTax	External ❗	🔘	onlyOwner
L	updateDevSellTax	External ❗	🔘	onlyOwner
L	updateLpBuyTax	External ❗	🔘	onlyOwner
L	updateLpSellTax	External ❗	🔘	onlyOwner
L	updateMaxWalletAmount	External ❗	🔘	onlyOwner

Contract	Type	Bases		
L	updateMaxTransactionAmount	External ❗	🛑	onlyOwner
L	_swapAndAddLiquidity	Internal 🔒	🛑	onlySwapping
L	_transfer	Internal 🔒	🛑	
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