SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

BullAI

BULLAI

ERC 20

Testnet Audit

0xa113Ce24919C08a26C952E81681dAc861d6a2466

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	BullAI
Ticker/Simbol	BULLAI
Blockchain	Binance Smart Chain BEP20
Contract Address	0xa113Ce24919C08a26C952E81681dAc861d6a2466
Creator Address	0x7BF175aa5054D1A1202225d873ca34232a5587A6
Current Owner Address	0x7BF175aa5054D1A1202225d873ca34232a5587A6
Contract Explorer	https://goerli.etherscan.io/address/0xa113ce24919c08a26c952e81681dac861d6a2466#code
Compiler Version	v0.8.17+commit.8df45f5f
License	
Optimisation	No with 200 Runs
Total Supply	10,000,000 $BULLAI
Decimals	18

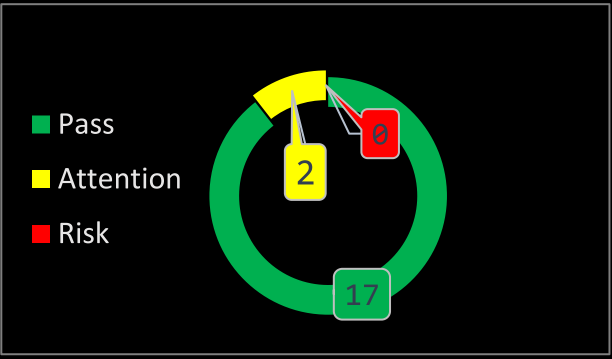
Creation/Audit

Contract Deployed	15 Dec 2023
Audit Created	19 Dec 2023
Audit Update	V 1.0

Verified Socials

Website	https://bullchartai.top/
Telegram	https://t.me/Bullchartaiofficial
Twitter (X)	https://x.com/BullChartAI

BULLAI ERC20



Contract Function Analysis

✓ Pass ⚠ Attention Item ❗ Risky Item

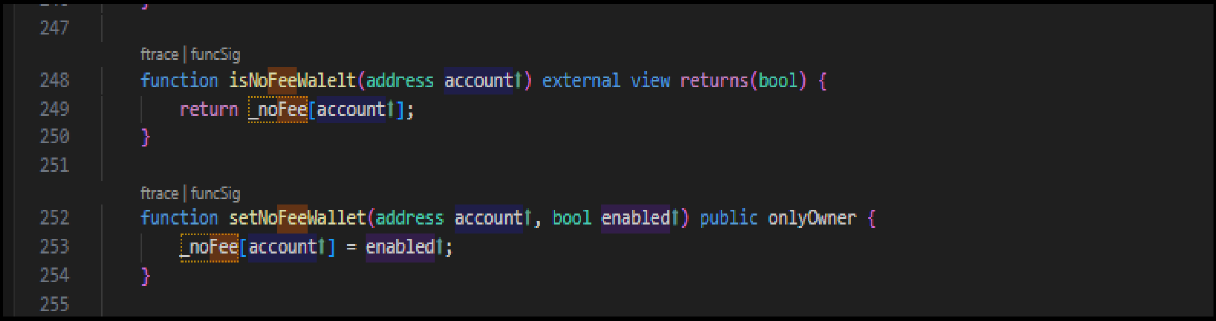
Contract Verified	✓	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it. Testnet
Contract Ownership		0x7BF175aa5054D1A1202225d873ca34232a5587A6
Buy Tax	0 %	Shows the taxes for purchase transactions. Above 10% may be considered a high tax rate. More 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
Liqudity Status		Testnet
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	⚠ buy max: 5 Sell max: 20	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 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. 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	✓	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)

Details of Risk - Attention Items

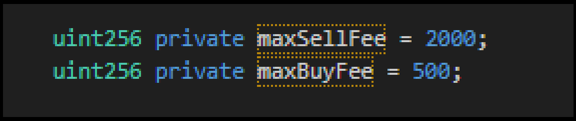
⚠ Whitelist Function

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)



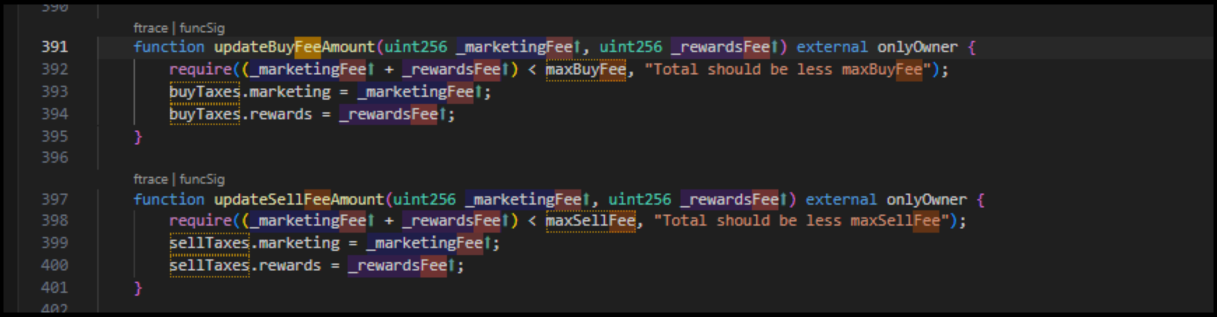
```
ftrace | funcSig
function isNoFeeWalelt(address account) external view returns(bool) {
    return _noFee[account];
}

ftrace | funcSig
function setNoFeeWallet(address account, bool enabled) public onlyOwner {
    _noFee[account] = enabled;
}
```

⚠ Set Fee Function Reducing Risk based on following limits: Max Fee on buy: 5% Max fee on sell: 20%



```
uint256 private maxSellFee = 2000;
uint256 private maxBuyFee = 500;
```

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).



```
ftrace | funcSig
function updateBuyFeeAmount(uint256 _marketingFee, uint256 _rewardsFee) external onlyOwner {
    require((_marketingFee + _rewardsFee) < maxBuyFee, "Total should be less maxBuyFee");
    buyTaxes.marketing = _marketingFee;
    buyTaxes.rewards = _rewardsFee;
}

ftrace | funcSig
function updateSellFeeAmount(uint256 _marketingFee, uint256 _rewardsFee) external onlyOwner {
    require((_marketingFee + _rewardsFee) < maxSellFee, "Total should be less maxSellFee");
    sellTaxes.marketing = _marketingFee;
    sellTaxes.rewards = _rewardsFee;
}
```

Contract Security

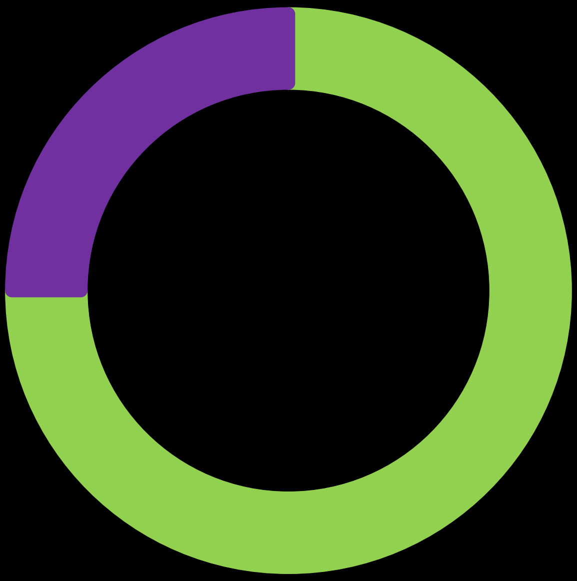
Total Findings: 4

🟥 **High 0**

🟨 **Medium**

🟩 **Low 3**

🟪 **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: (0)**

🟩 **Low severity issues: (3)**

- Missing Events
- Long number literals
- Usage of TX Origin

🟪 **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	High	Low	Low
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

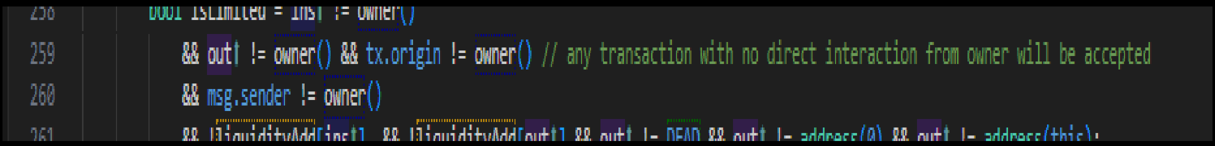
SWC-121	Missing Protection against Signature Replay Attacks	Passed	Passed	Passed
SWC-122	Lack of Proper Signature Verification	Passed	Passed	Passed
SWC-123	Requirement Violation	Passed	Passed	Passed
SWC-124	Write to Arbitrary Storage Location	Passed	Passed	Passed
SWC-125	Incorrect Inheritance Order	Passed	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.

⚠ Usage of TX.Origin (1 Item) Risk lowered to low based on the function

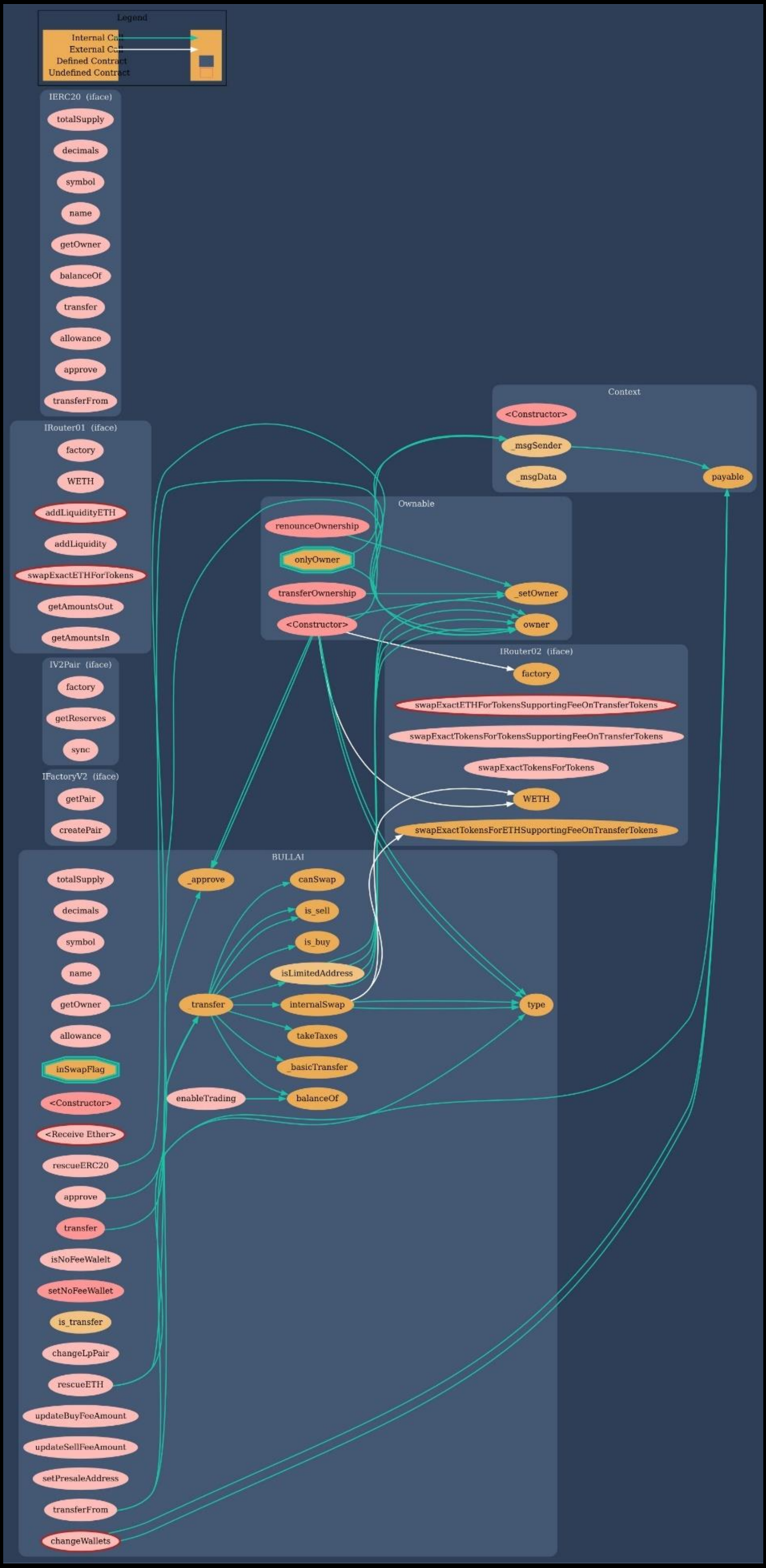
Item: 1	Location:	Line 259	Severity:	🟩 Low

Function	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.
Remedation	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.

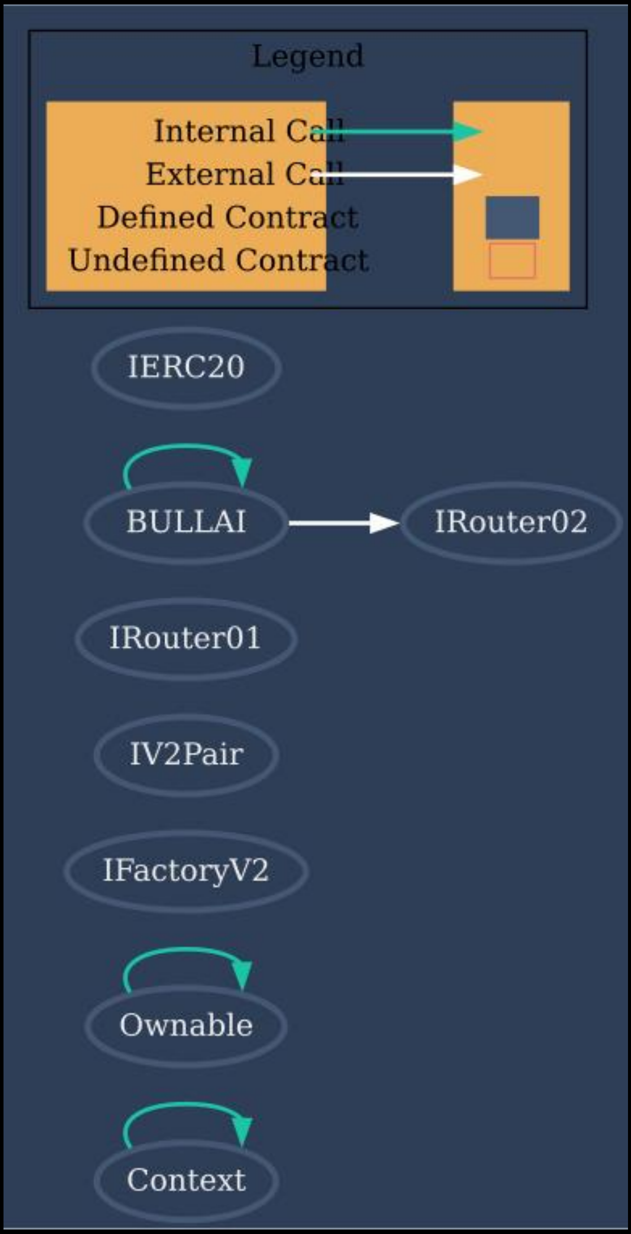


```
258        bool isLimited = ins! != owner()
259            && out! != owner() && tx.origin != owner() // any transaction with no direct interaction from owner will be accepted
260            && msg.sender != owner()
261            && !liquidityAdd[ins!] && !liquidityAdd[out!] && out! != DEAD && out! != address(0) && out! != address(this);
```

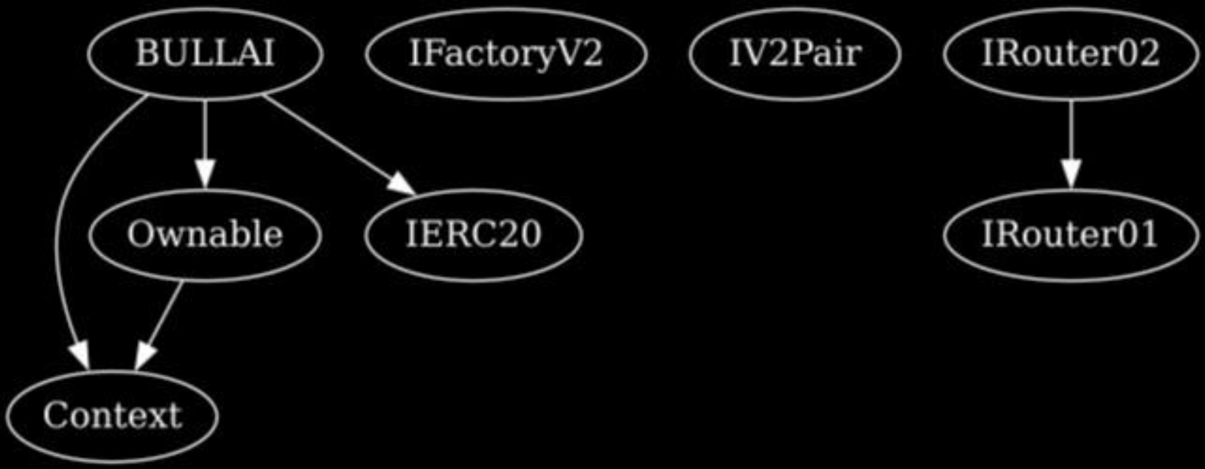
Contract Flow Graph



Contract Interaction Graph



Inheritance Graph



Contract Functions

Contract	Type	Bases		
└	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
└		Public ❗	🛑	NO❗
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		
Ownable	Implementation	Context		
└		Public ❗	🛑	NO❗
└	owner	Public ❗		NO❗
└	renounceOwnership	Public ❗	🛑	onlyOwner
└	transferOwnership	Public ❗	🛑	onlyOwner
└	_setOwner	Private 🔐	🛑	
IFactoryV2	Interface			
└	getPair	External ❗		NO❗
└	createPair	External ❗	🛑	NO❗
IV2Pair	Interface			
└	factory	External ❗		NO❗
└	getReserves	External ❗		NO❗
└	sync	External ❗	🛑	NO❗

Contract	Type	Bases		
IRouter01	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❗
IRouter02	Interface	IRouter01		
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO❗
L	swapExactTokensForTokensSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
L	swapExactTokensForTokens	External ❗	🛑	NO❗
IERC20	Interface			
L	totalSupply	External ❗		NO❗
L	decimals	External ❗		NO❗

Contract	Type	Bases		
└	symbol	External ❗		NO❗
└	name	External ❗		NO❗
└	getOwner	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
BULLAI	Implementation	Context, Ownable, IERC20		
└	totalSupply	External ❗		NO❗
└	decimals	External ❗		NO❗
└	symbol	External ❗		NO❗
└	name	External ❗		NO❗
└	getOwner	External ❗		NO❗
└	allowance	External ❗		NO❗
└	balanceOf	Public ❗		NO❗
└		Public ❗	🛑	NO❗
└		External ❗	💵	NO❗
└	transfer	Public ❗	🛑	NO❗
└	approve	External ❗	🛑	NO❗
└	_approve	Internal 🔒	🛑	

Contract	Type	Bases		
L	transferFrom	External ❗	🛑	NO❗
L	isNoFeeWalelt	External ❗		NO❗
L	setNoFeeWallet	Public ❗	🛑	onlyOwner
L	isLimitedAddress	Internal 🔒		
L	is_buy	Internal 🔒		
L	is_sell	Internal 🔒		
L	is_transfer	Internal 🔒		
L	canSwap	Internal 🔒		
L	changeLpPair	External ❗	🛑	onlyOwner
L	_transfer	Internal 🔒	🛑	
L	_basicTransfer	Internal 🔒	🛑	
L	changeWallets	External ❗	💵	onlyOwner
L	takeTaxes	Internal 🔒	🛑	
L	internalSwap	Internal 🔒	🛑	inSwapFlag
L	updateBuyFeeAmount	External ❗	🛑	onlyOwner
L	updateSellFeeAmount	External ❗	🛑	onlyOwner
L	setPresaleAddress	External ❗	🛑	onlyOwner
L	enableTrading	External ❗	🛑	onlyOwner
L	rescueETH	External ❗	🛑	onlyOwner
L	rescueERC20	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 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