SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

Limitless Network (LNT)

BEP20

0xC13CbF50370E5EaE6f5Dd9D8a1015007f34C4

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	Limitless
Ticker/Simbol	LNT
Blockchain	Binance Smart Chain BEP20
Contract Address	0xC13CbF50370E5EaE6f5Dd9D8a1015007f34C4eaD
Creator Address	0x89856aa2F9D74B70bA67fB821A86E17E9796FB35
Current Owner Address	0x7df6408B48130015A62A42f5D8818e78AFE2b319
Contract Explorer	https://bscscan.com/address/0xC13CbF50370E5EaE6f5Dd9D8a1015007f34C4eaD#code
Compiler Version	v0.8.15+commit.e14f2714
License	MIT
Optimisation	Yes with 2000 Runs
Total Supply	1,000,000,000 LNT
Decimals	18

Creation/Audit

Contract Deployed	19.04.2023
Audit Created	11.04.2024
Audit Update	V 1.0

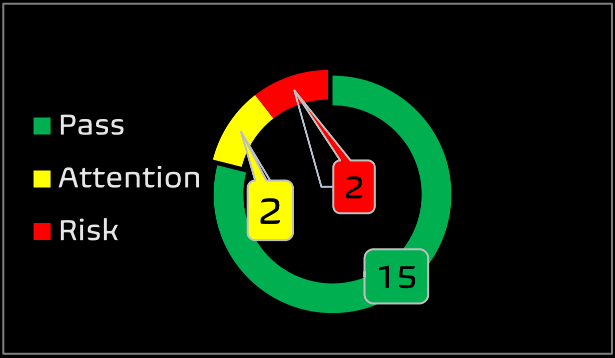
Verified Socials

Website	https://limitlessnetwork.org/
Telegram	https://t.me/limitless_Network_Token
Twitter (X)	https://twitter.com/Limitless_LNT

LIMITLESS NETWORK BEP20

Contract Function Analysis

✓ Pass ⚠ Attention Item ❗ Risky Item



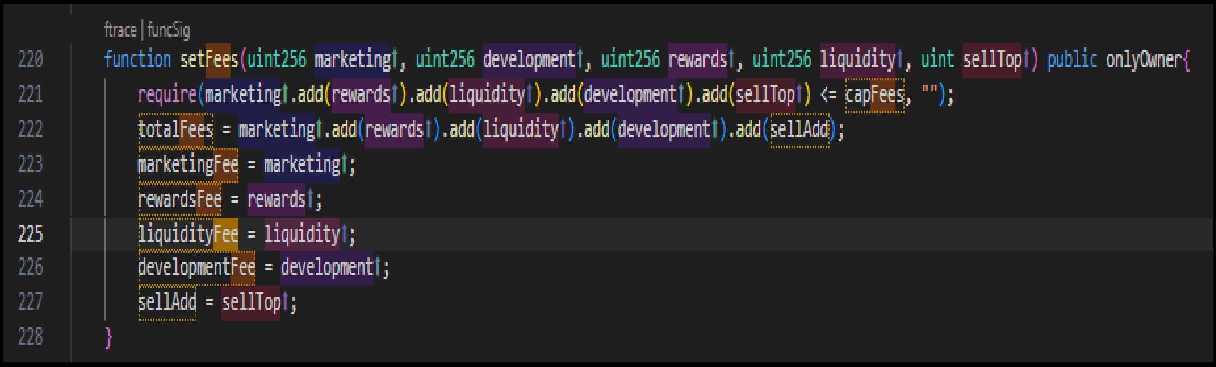
Contract Verified	✓	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership		0x7df6408B48130015A62A42f5D8818e78AFE2b319
Buy Tax	5 %	Shows the taxes for purchase transactions. Above 10% may be considered a high tax rate. More than 50% 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
Liqudity Status	✓	Liqudity status on 11.04.2024 98% for 11 Days on Mudra
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	⚠️	Blacklist and Multi-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	⚠️	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

Set Fee

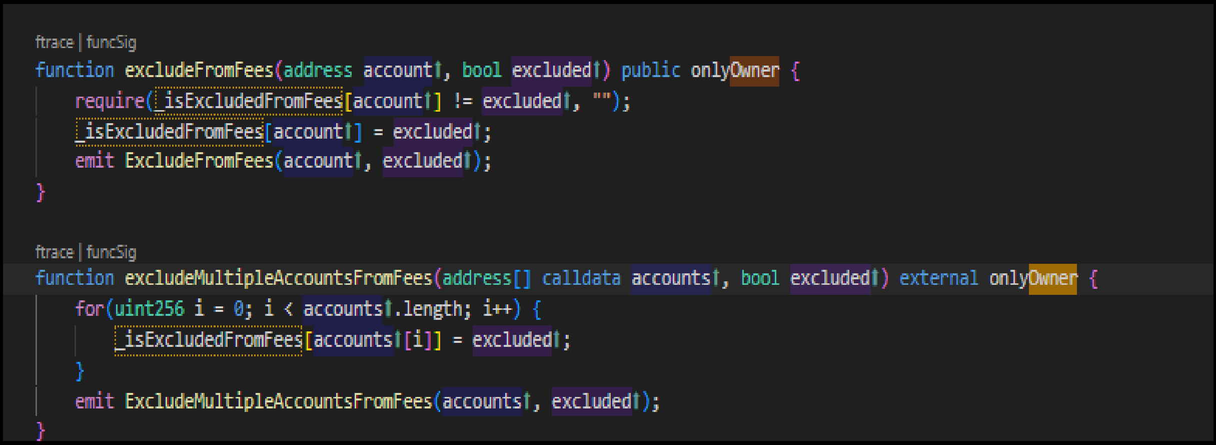
The contract owner may contain the authority to modify the transaction tax. If the transaction tax is increased to more than 49%, the tokens may not be able to be traded (honeypot risk).



```
ftrace | funcSig
function setFees(uint256 marketing, uint256 development, uint256 rewards, uint256 liquidity, uint sellTop) public onlyOwner{
    require(marketing.add(rewards).add(liquidity).add(development).add(sellTop) <= capFees, "");
    totalFees = marketing.add(rewards).add(liquidity).add(development).add(sellAdd);
    marketingFee = marketing;
    rewardsFee = rewards;
    liquidityFee = liquidity;
    developmentFee = development;
    sellAdd = sellTop;
}
```

Whitelist (Set wallets excluded from fees)

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)

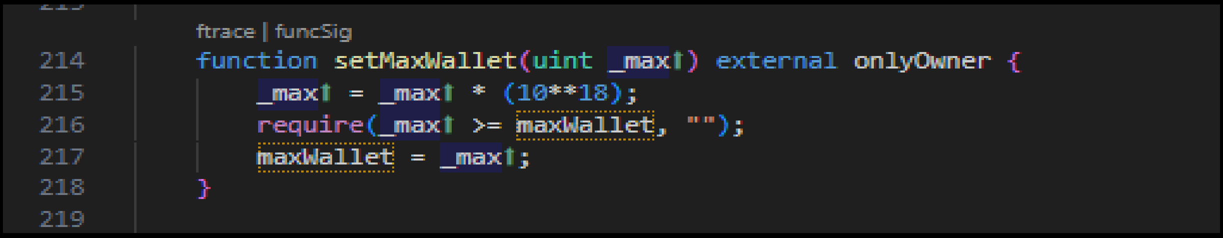


```
ftrace | funcSig
function excludeFromFees(address account, bool excluded) public onlyOwner {
    require(_isExcludedFromFees[account] != excluded, "");
    _isExcludedFromFees[account] = excluded;
    emit ExcludeFromFees(account, excluded);
}

ftrace | funcSig
function excludeMultipleAccountsFromFees(address[] calldata accounts, bool excluded) external onlyOwner {
    for(uint256 i = 0; i < accounts.length; i++) {
        _isExcludedFromFees[accounts[i]] = excluded;
    }
    emit ExcludeMultipleAccountsFromFees(accounts, excluded);
}
```

⚠ Max Transaction and Holding Modify function

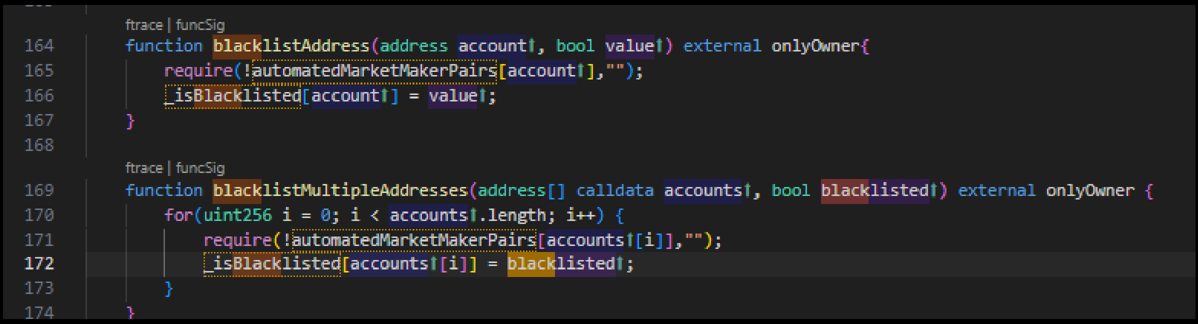
If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot



```
ftrace | funcSig
function setMaxWallet(uint _max) external onlyOwner {
    _max = _max * (10**18);
    require(_max >= maxWallet, "");
    maxWallet = _max;
}
```

❗ Blacklist and Multiple Blacklist 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.

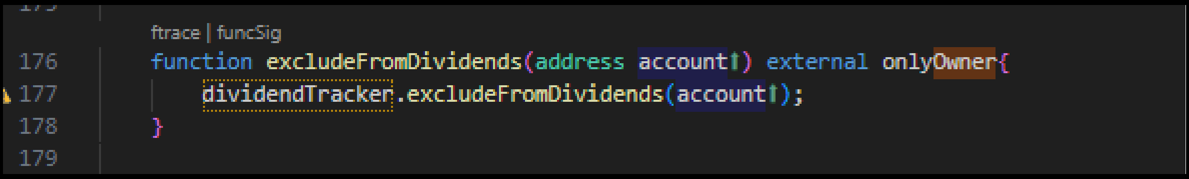


```
ftrace | funcSig
function blacklistAddress(address account, bool value) external onlyOwner{
    require(!automatedMarketMakerPairs[account],"");
    _isBlacklisted[account] = value;
}

ftrace | funcSig
function blacklistMultipleAddresses(address[] calldata accounts, bool blacklisted) external onlyOwner {
    for(uint256 i = 0; i < accounts.length; i++) {
        require(!automatedMarketMakerPairs[accounts[i]],"");
        _isBlacklisted[accounts[i]] = blacklisted;
    }
}
```

⚠ Blacklist Wallets from receiving dividend

Wallets can be blacklisted from receiving dividens rewards. In some cases this is to avoid team wallets, burn address to receive rewards, in some cases holder wallets can be blacklisted as well.



```
ftrace | funcSig
function excludeFromDividends(address account) external onlyOwner{
    dividendTracker.excludeFromDividends(account);
}
```

Contract Security

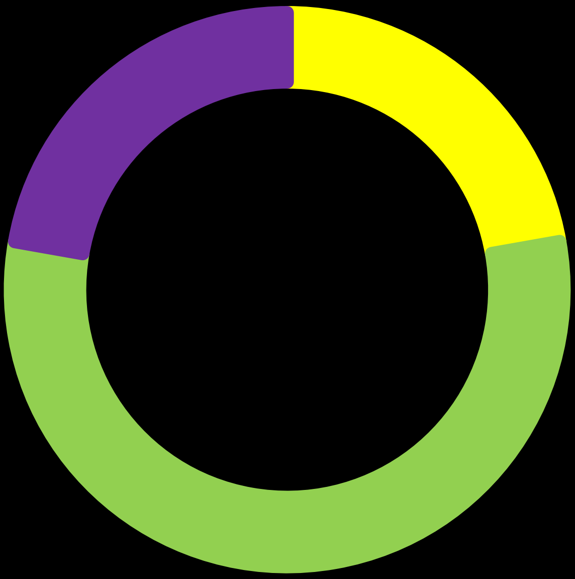
Total Findings: 9

🟥 High 0

🟨 Medium 2

🟩 Low 5

🟪 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: (2)

- Usage of tx. origin
- Incorrect Acces Control

🟩 Low severity issues: (5)

- Missing Events
- Long number literals
- Outdated compiler Version
- Unchecked Array Lenght
- Approve of Front Running Attack

🟪 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

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	low	low	low
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	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

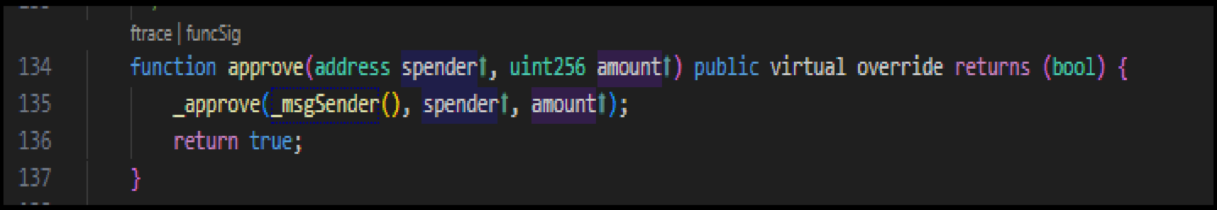
SWC-121	Missing Protection against Signature Replay Attacks	Passed	Passed	Passed
SWC-122	Lack of Proper Signature Verification	Passed	Passed	Passed
SWC-123	Requirement Violation	Passed	Passed	Passed
SWC-124	Write to Arbitrary Storage Location	Passed	Passed	Passed
SWC-125	Incorrect Inheritance Order	Passed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

⚠ Approve of Front running Attack (2 Item)

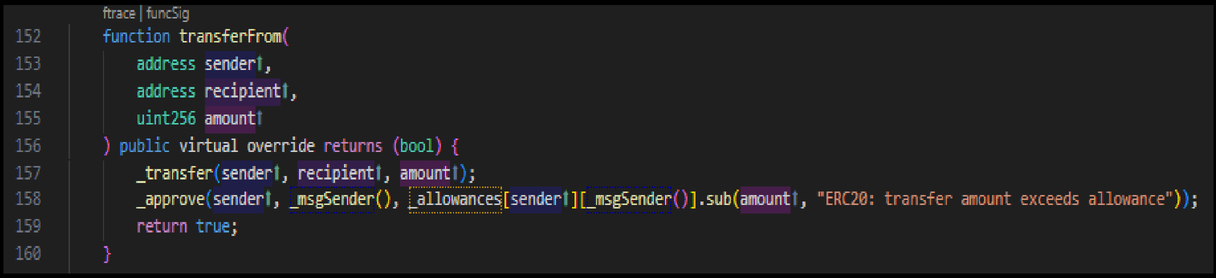
Item: 1	Location:	ERC20.sol Line 134-137	Severity:	🟩 Low

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
function approve(address spender, uint256 amount) public virtual override returns (bool) {
    _approve(_msgSender(), spender, amount);
    return true;
}
```

Item: 2	Location:	ERC20.sol Line 152-160	Severity:	🟩 Low

Function	Access control plays an important role in segregation of privileges in smart contracts and other applications. If this is misconfigured or not properly validated on sensitive functions, it may lead to loss of funds, tokens and in some cases compromise of the smart contract. The contract Limitless is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function claim is missing the modifier onlyOwner.
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
152    function transferFrom(
153        address sender,
154        address recipient,
155        uint256 amount
156    ) public virtual override returns (bool) {
157        _transfer(sender, recipient, amount);
158        _approve(sender, _msgSender(), _allowances[sender][_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));
159        return true;
160    }
```

⚠️ Outdated Compiler Version. (18 Items)

Item: 1	Location:	Multiple	Severity:	🟩 Low

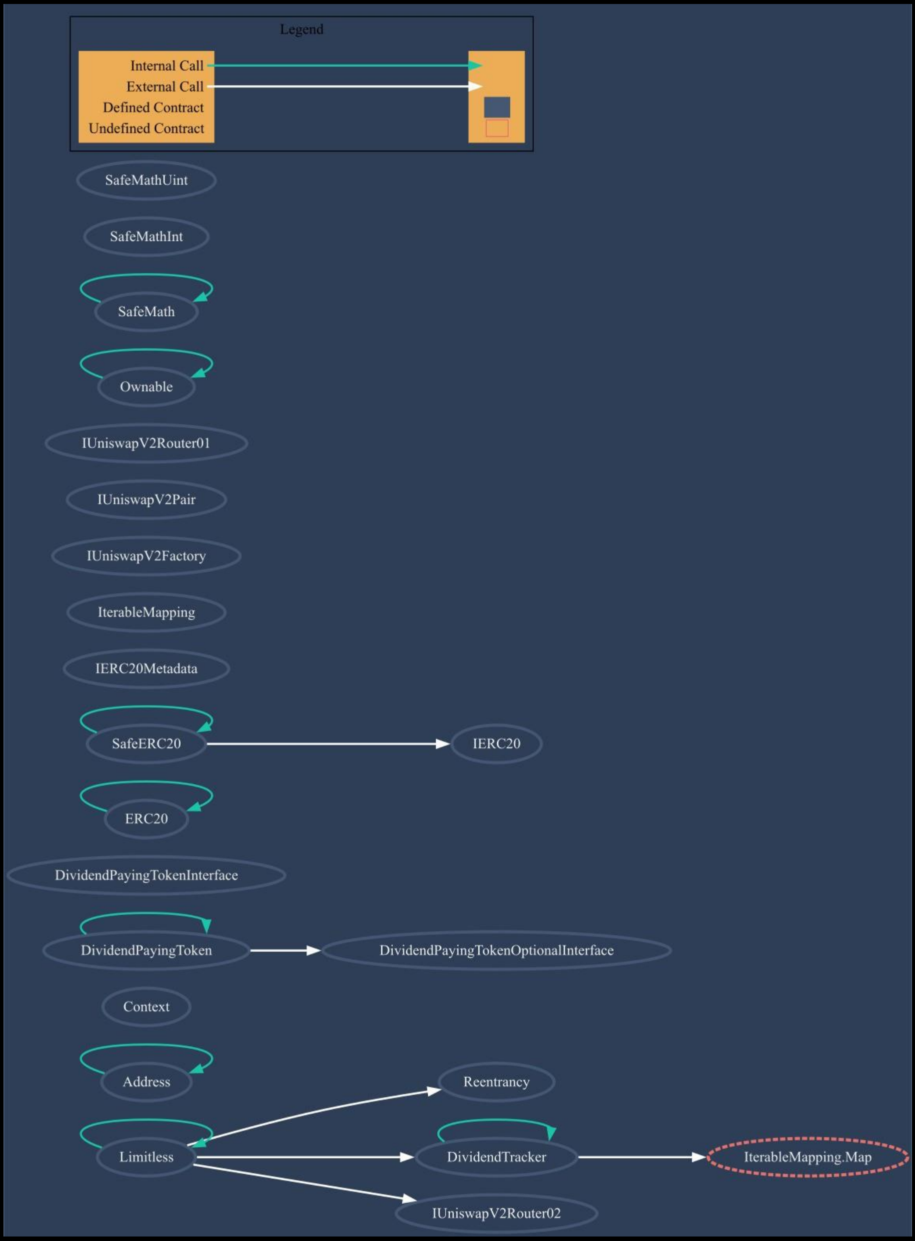
Function	Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler versions
Remedation	It is recommended to use a recent version of the Solidity compiler that should not be the most recent version, and it should not be an outdated version as well. Using very old versions of Solidity prevents the benefits of bug fixes and newer security checks. Consider using the solidity version v0.8.24, which patches most solidity vulnerabilities.

⚠ Usage of tx. origin (1 Item)

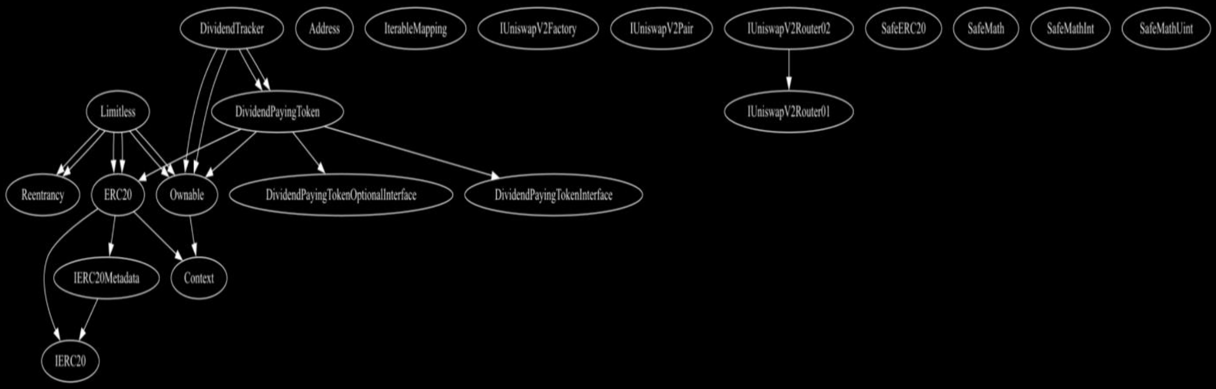
Item: 1	Location:	Limitless.sol Line 324	Severity:	🟨 Medium

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	The best way to prevent Tx Origin attacks is not to use the tx.origin for authentication purposes. Instead, it is advisable to use msg.sender You can implement a require of the form require(tx.origin == msg.sender).

Contract Interaction Graph



Inheritance Graph



Contract Functions

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
Reentrancy	Implementation			
L		Public ❗	🛑	NO❗
Limitless	Implementation	ERC20, Ownable, Reentrancy		
L		Public ❗	🛑	ERC20
L		External ❗	💸	NO❗
L	updateDividendTracker	Public ❗	🛑	onlyOwner
L	updateUniswapV2Router	Public ❗	🛑	onlyOwner
L	updateSwapTokensAtAmount	External ❗	🛑	onlyOwner
L	excludeFromFees	Public ❗	🛑	onlyOwner
L	excludeMultipleAccountsFromFees	External ❗	🛑	onlyOwner
L	blacklistAddress	External ❗	🛑	onlyOwner
L	blacklistMultipleAddresses	External ❗	🛑	onlyOwner
L	excludeFromDividends	External ❗	🛑	onlyOwner
L	includeInDividends	External ❗	🛑	onlyOwner
L	updateGasStipend	Public ❗	🛑	onlyOwner
L	setMarketingWallet	External ❗	🛑	onlyOwner

Contract	Type	Bases		
L	setDevelopment Wallet	External ❗	🛑	onlyOwner
L	setSlippage	External ❗	🛑	onlyOwner
L	setGas	External ❗	🛑	onlyOwner
L	setMaxWallet	External ❗	🛑	onlyOwner
L	setFees	Public ❗	🛑	onlyOwner
L	setAutomatedM arketMakerPair	Public ❗	🛑	onlyOwner
L	updateClaimWai t	External ❗	🛑	onlyOwner
L	activateTrading	Public ❗	🛑	onlyOwner
L	processDividend Tracker	External ❗	🛑	nonReentrant
L	claim	External ❗	🛑	nonReentrant
L	processAccount	External ❗	🛑	nonReentrant
L	_transfer	Internal 🔒	🛑	
L	swapBack	Internal 🔒	🛑	
L	swapTokensFor Eth	Private 🔐	🛑	
L	addLiquidity	Private 🔐	🛑	
L	withdrawBep20	Public ❗	🛑	onlyOwner nonReentrant
L	withdrawBNB	Public ❗	🛑	onlyOwner nonReentrant
L	getClaimWait	External ❗		NO❗
L	getTotalDividen dsDistributed	External ❗		NO❗
L	isExcludedFrom Fees	Public ❗		NO❗

Contract	Type	Bases		
L	getDividendTokensMinimum	External ❗		NO❗
L	withdrawableDividendOf	Public ❗		NO❗
L	dividendTokenBalanceOf	Public ❗		NO❗
L	getAccountDividendsInfo	External ❗		NO❗
L	getAccountDividendsInfoAtIndex	External ❗		NO❗
L	getLastProcessedIndex	External ❗		NO❗
L	getNumberOfDividendTokenHolders	External ❗		NO❗
DividendTracker	Implementation	Ownable, DividendPayingToken		
L		Public ❗	🛑	DividendPayingToken
L	_transfer	Internal 🔒		
L	excludeFromDividends	External ❗	🛑	onlyOwner
L	includeInDividends	External ❗	🛑	onlyOwner
L	updateClaimWait	External ❗	🛑	onlyOwner
L	setBalance	External ❗	🛑	onlyOwner
L	process	Public ❗	🛑	onlyOwner
L	processAccount	Public ❗	🛑	onlyOwner
L	getLastProcessedIndex	External ❗		NO❗

Contract	Type	Bases		
└	getNumberOfTokenHolders	External ❗		NO❗
└	getAccount	Public ❗		NO❗
└	getAccountAtIndex	Public ❗		NO❗
└	canAutoClaim	Private 🔐		
Address	Library			
└	isContract	Internal 🔒		
└	sendValue	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCall	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionCallWithValue	Internal 🔒	🛑	
└	functionStaticCall	Internal 🔒		
└	functionStaticCall	Internal 🔒		
└	functionDelegateCall	Internal 🔒	🛑	
└	functionDelegateCall	Internal 🔒	🛑	
└	verifyCallResult	Internal 🔒		
Context	Implementation			
└	_msgSender	Internal 🔒		
└	_msgData	Internal 🔒		

Contract	Type	Bases		
DividendPaying Token	Implementation	ERC20, Ownable, DividendPaying TokenInterface, DividendPaying TokenOptionalIn terface		
L		Public ❗	🛑	ERC20
L		External ❗	💵	NO❗
L	updateStipend	Public ❗	🛑	onlyOwner
L	setSlippage	External ❗	🛑	onlyOwner
L	distributeDivide nds	Public ❗	💵	NO❗
L	_withdrawDivid endOfUser	Internal 🔒	🛑	
L	_mint	Internal 🔒	🛑	
L	_burn	Internal 🔒	🛑	
L	_setBalance	Internal 🔒	🛑	
L	dividendOf	Public ❗		NO❗
L	withdrawableDi videndOf	Public ❗		NO❗
L	withdrawnDivid endOf	Public ❗		NO❗
L	accumulativeDiv idendOf	Public ❗		NO❗
DividendPaying TokenInterface	Interface			
L	dividendOf	External ❗		NO❗
DividendPaying TokenOptionalIn terface	Interface			

Contract	Type	Bases		
L	withdrawableDividendOf	External ❗		NO❗
L	withdrawnDividendOf	External ❗		NO❗
L	accumulativeDividendOf	External ❗		NO❗
ERC20	Implementation	Context, IERC20, IERC20Metadata		
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❗
L	transferFrom	Public ❗	🛑	NO❗
L	increaseAllowance	Public ❗	🛑	NO❗
L	decreaseAllowance	Public ❗	🛑	NO❗
L	_transfer	Internal 🔒	🛑	
L	_mint	Internal 🔒	🛑	
L	_burn	Internal 🔒	🛑	
L	_approve	Internal 🔒	🛑	

Contract	Type	Bases		
└	_beforeTokenTransfer	Internal 🔒	🛑	
IERC20	Interface			
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
IERC20Metadata	Interface	IERC20		
└	name	External ❗		NO❗
└	symbol	External ❗		NO❗
└	decimals	External ❗		NO❗
IterableMapping	Library			
└	get	Public ❗		NO❗
└	getIndexOfKey	Public ❗		NO❗
└	getKeyAtIndex	Public ❗		NO❗
└	size	Public ❗		NO❗
└	set	Public ❗	🛑	NO❗
└	remove	Public ❗	🛑	NO❗
IUniswapV2Factory	Interface			
└	feeTo	External ❗		NO❗

Contract	Type	Bases		
└	feeToSetter	External ❗		NO❗
└	getPair	External ❗		NO❗
└	allPairs	External ❗		NO❗
└	allPairsLength	External ❗		NO❗
└	createPair	External ❗	🛑	NO❗
└	setFeeTo	External ❗	🛑	NO❗
└	setFeeToSetter	External ❗	🛑	NO❗
IUniswapV2Pair	Interface			
└	name	External ❗		NO❗
└	symbol	External ❗		NO❗
└	decimals	External ❗		NO❗
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transfer	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
└	DOMAIN_SEPARATOR	External ❗		NO❗
└	PERMIT_TYPEHASH	External ❗		NO❗
└	nonces	External ❗		NO❗
└	permit	External ❗	🛑	NO❗
└	MINIMUM_LIQUIDITY	External ❗		NO❗

Contract	Type	Bases		
└	factory	External ❗		NO❗
└	token0	External ❗		NO❗
└	token1	External ❗		NO❗
└	getReserves	External ❗		NO❗
└	price0CumulativeLast	External ❗		NO❗
└	price1CumulativeLast	External ❗		NO❗
└	kLast	External ❗		NO❗
└	mint	External ❗	🛑	NO❗
└	burn	External ❗	🛑	NO❗
└	swap	External ❗	🛑	NO❗
└	skim	External ❗	🛑	NO❗
└	sync	External ❗	🛑	NO❗
└	initialize	External ❗	🛑	NO❗
IUniswapV2Router01	Interface			
└	factory	External ❗		NO❗
└	WETH	External ❗		NO❗
└	addLiquidity	External ❗	🛑	NO❗
└	addLiquidityETH	External ❗	💵	NO❗
└	removeLiquidity	External ❗	🛑	NO❗
└	removeLiquidityETH	External ❗	🛑	NO❗
└	removeLiquidityWithPermit	External ❗	🛑	NO❗

Contract	Type	Bases		
L	removeLiquidityETHWithPermit	External ❗	🛑	NO❗
L	swapExactTokensForTokens	External ❗	🛑	NO❗
L	swapTokensForExactTokens	External ❗	🛑	NO❗
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❗

Contract	Type	Bases		
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External ❗	💵	NO❗
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗	🛑	NO❗
Ownable	Implementation	Context		
L		Public ❗	🛑	NO❗
L	owner	Public ❗		NO❗
L	renounceOwnership	Public ❗	🛑	onlyOwner
L	transferOwnership	Public ❗	🛑	onlyOwner
SafeERC20	Library			
L	safeTransfer	Internal 🔒	🛑	
L	safeTransferFrom	Internal 🔒	🛑	
L	safeApprove	Internal 🔒	🛑	
L	safeIncreaseAllowance	Internal 🔒	🛑	
L	safeDecreaseAllowance	Internal 🔒	🛑	
L	_callOptionalReturn	Private 🔐	🛑	
SafeMath	Library			
L	add	Internal 🔒		
L	sub	Internal 🔒		
L	sub	Internal 🔒		

Contract	Type	Bases		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	div	Internal 🔒		
└	mod	Internal 🔒		
└	mod	Internal 🔒		
SafeMathInt	Library			
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	sub	Internal 🔒		
└	add	Internal 🔒		
└	abs	Internal 🔒		
└	toUint256Safe	Internal 🔒		
SafeMathUint	Library			
└	toInt256Safe	Internal 🔒		

🛑	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