SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

Safe Degen

Safe

BEP20

0x6482aC16643ebaFbd86DBeCEDAF6b49cA0a3

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	Token
Ticker/Simbol	Safe
Blockchain	Binance Smart Chain BEP20
Contract Address	0x6482aC16643ebaFbd86DBeCEDAF6b49cA0a36E5E
Creator Address	0xbFCE539B992465cDD90595E276B56a9355db023f
Current Owner Address	0xbFCE539B992465cDD90595E276B56a9355db023f
Contract Explorer	https://bscscan.com/address/0x6482ac16643ebafbd86dbecedaf6b49ca0a36e5e#code
Compiler Version	v0.8.6+commit.11564f7e
License	MIT
Optimisation	Yes with 200 Runs
Total Supply	21,000,000 **Safe**
Decimals	9

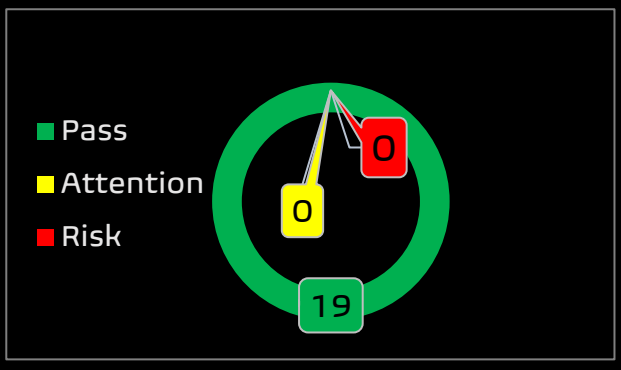
Creation/Audit

Contract Deployed	12.12.2023
Audit Created	15.02.2024
Audit Update	V 1.0

Verified Socials

Website	https://www.safedegens.com/
Telegram	https://t.me/safedegensportal
Twitter (X)	http://x.com/SafeDegens

SAFE DEGEN BEP20



Contract Function Analysis

Pass Attention Item Risky Item

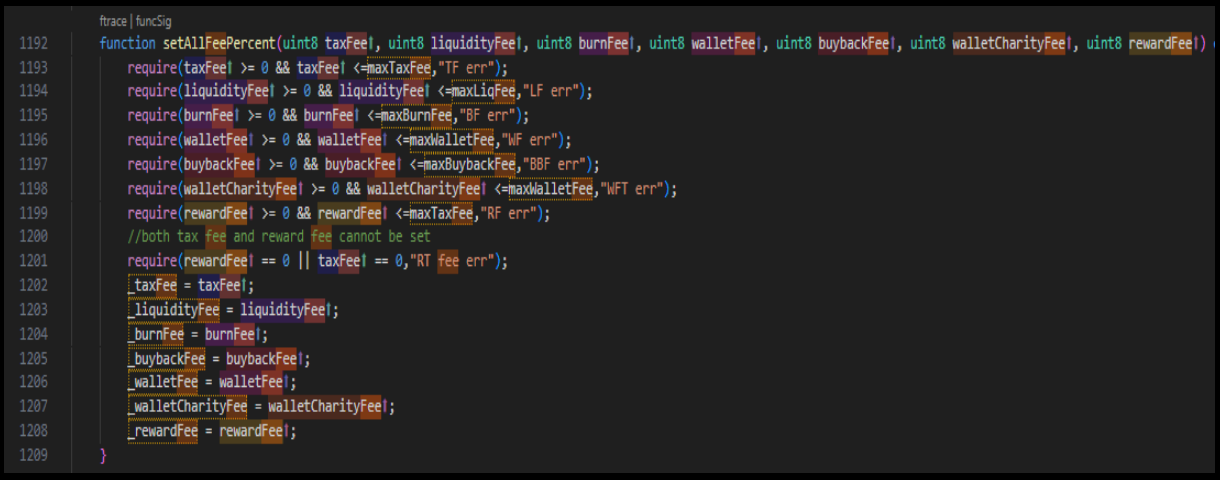
Contract Verified	✓	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership		0x6482aC16643ebaFbd86DBeCEDAF6b49cA0a36E5E
Buy Tax	9 %	Shows the taxes for purchase transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable. Fee can be set!
Sell Tax	9 %	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 15.02.2024 Lp Locked: 90.96% Mudra Locker for 685 days. Lp Burned: 9.01%
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 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

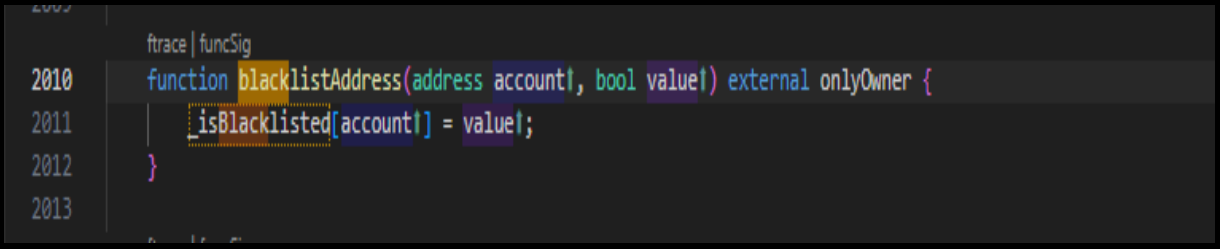
The contract owner may contain the authority to modify the transaction tax. If the transaction tax is increased to more than 49%, the tokens may not be able to be traded (honeypot risk).



```
ftrace | funcSig
function setAllFeePercent(uint8 taxFee!, uint8 liquidityFee!, uint8 burnFee!, uint8 walletFee!, uint8 buybackFee!, uint8 walletCharityFee!, uint8 rewardFee!) {
    require(taxFee! >= 0 && taxFee! <=maxTaxFee,"TF err");
    require(liquidityFee! >= 0 && liquidityFee! <=maxLiqFee,"LF err");
    require(burnFee! >= 0 && burnFee! <=maxBurnFee,"BF err");
    require(walletFee! >= 0 && walletFee! <=maxWalletFee,"WF err");
    require(buybackFee! >= 0 && buybackFee! <=maxBuybackFee,"BBF err");
    require(walletCharityFee! >= 0 && walletCharityFee! <=maxWalletFee,"WFT err");
    require(rewardFee! >= 0 && rewardFee! <=maxTaxFee,"RF err");
    //both tax fee and reward fee cannot be set
    require(rewardFee! == 0 || taxFee! == 0,"RT fee err");
    _taxFee = taxFee!;
    _liquidityFee = liquidityFee!;
    _burnFee = burnFee!;
    _buybackFee = buybackFee!;
    _walletFee = walletFee!;
    _walletCharityFee = walletCharityFee!;
    _rewardFee = rewardFee!;
}
```

⚠ Blacklist

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
2010  function blacklistAddress(address account!, bool value!) external onlyOwner {
2011      isBlacklisted[account!] = value!;
2012  }
2013
```

⚠ Whitelist

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 excludeFromFee(address account) public onlyOwner {
    _isExcludedFromFee[account] = true;
}
```

⚠ 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 setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
    require(maxTxPercent >= minMxTxPercentage && maxTxPercent <=10000,"err");
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(
        10**4
    );
}

ftrace | funcSig
function setMaxWalletPercent(uint256 maxWalletPercent) external onlyOwner() {
    require(maxWalletPercent >= minMxWalletPercentage && maxWalletPercent <=10000,"err");
    _maxWalletAmount = _tTotal.mul(maxWalletPercent).div(
        10**4
    );
}
```

Contract Security

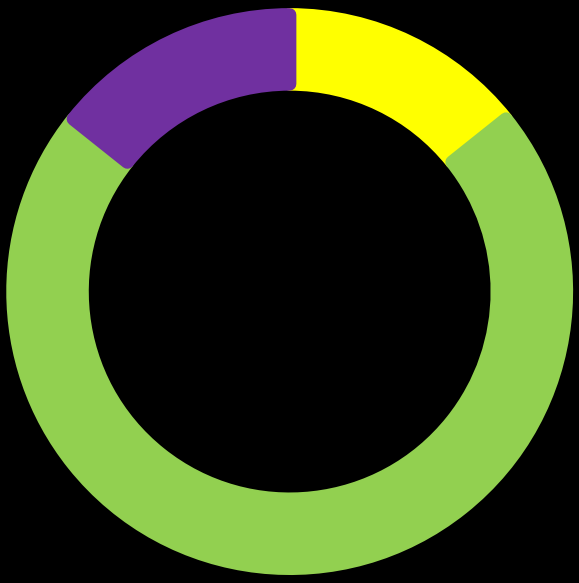
Total Findings: 7

High 0

Medium 1

Low 5

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: (5)

- Missing Events
- Long number literals
- Outdated compiler Version
- Costly Loop Operations
- Unchecked Array Lenght

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	low	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	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 Item)

Item: 1	Location:	Line 1424	Severity:	Medium
Item: 2	Location:	Line 2007	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	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.

```
1423            (uint256 iterations, uint256 claims, uint256 _lastProcessedIndex) = process(gas);
1424            emit ProcessedDividendTracker(iterations, claims, _lastProcessedIndex, true, gas, tx.origin);
1425        }
```

```
2006        (uint256 iterations, uint256 claims, uint256 _lastProcessedIndex) = process(gas);
2007        emit ProcessedDividendTracker(iterations, claims, _lastProcessedIndex, false, gas, tx.origin);
2008    }
```

⚠ Approve of front running attack. Also known as Sandwich bot attack. (2 Items)

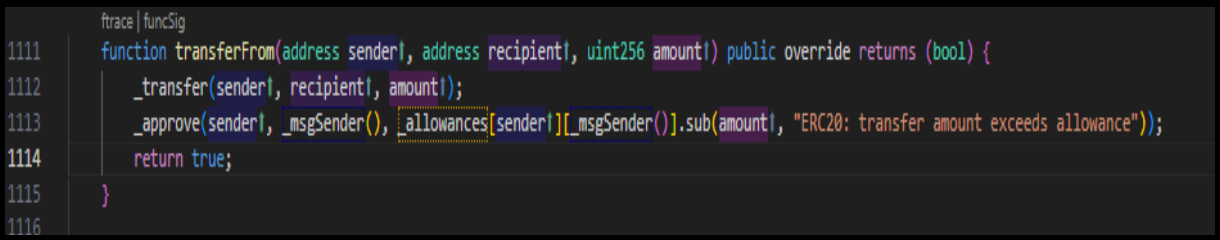
Item: 1	Location:	Line 1106-1109	Severity:	Low

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

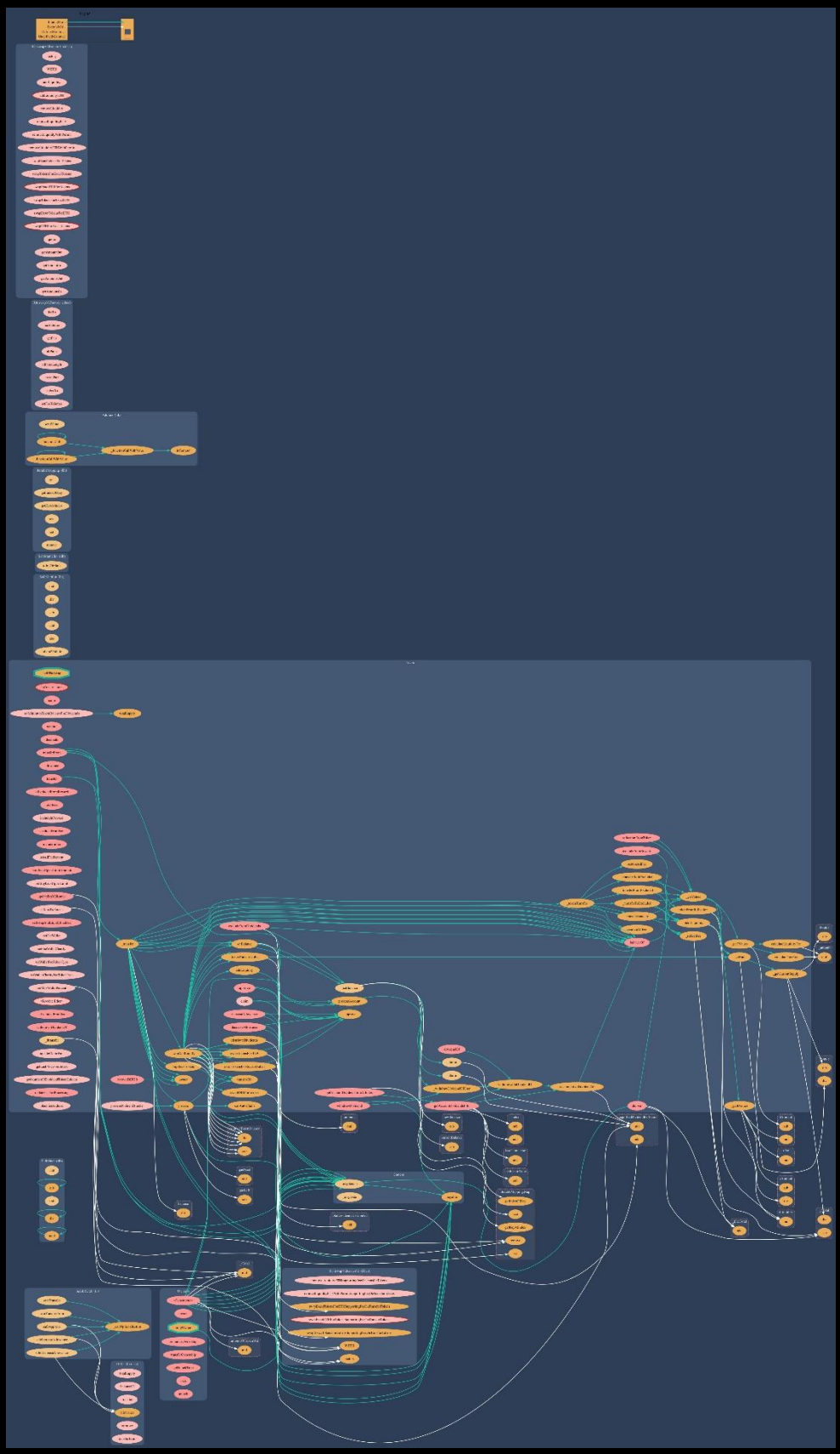
```
ftrace | funcSig
1106    function approve(address spender1, uint256 amount1) public override returns (bool) {
1107        _approve(_msgSender(), spender1, amount1);
1108        return true;
1109    }
1110
```

Item: 2	Location:	Line 1111-1115	Severity:	Low

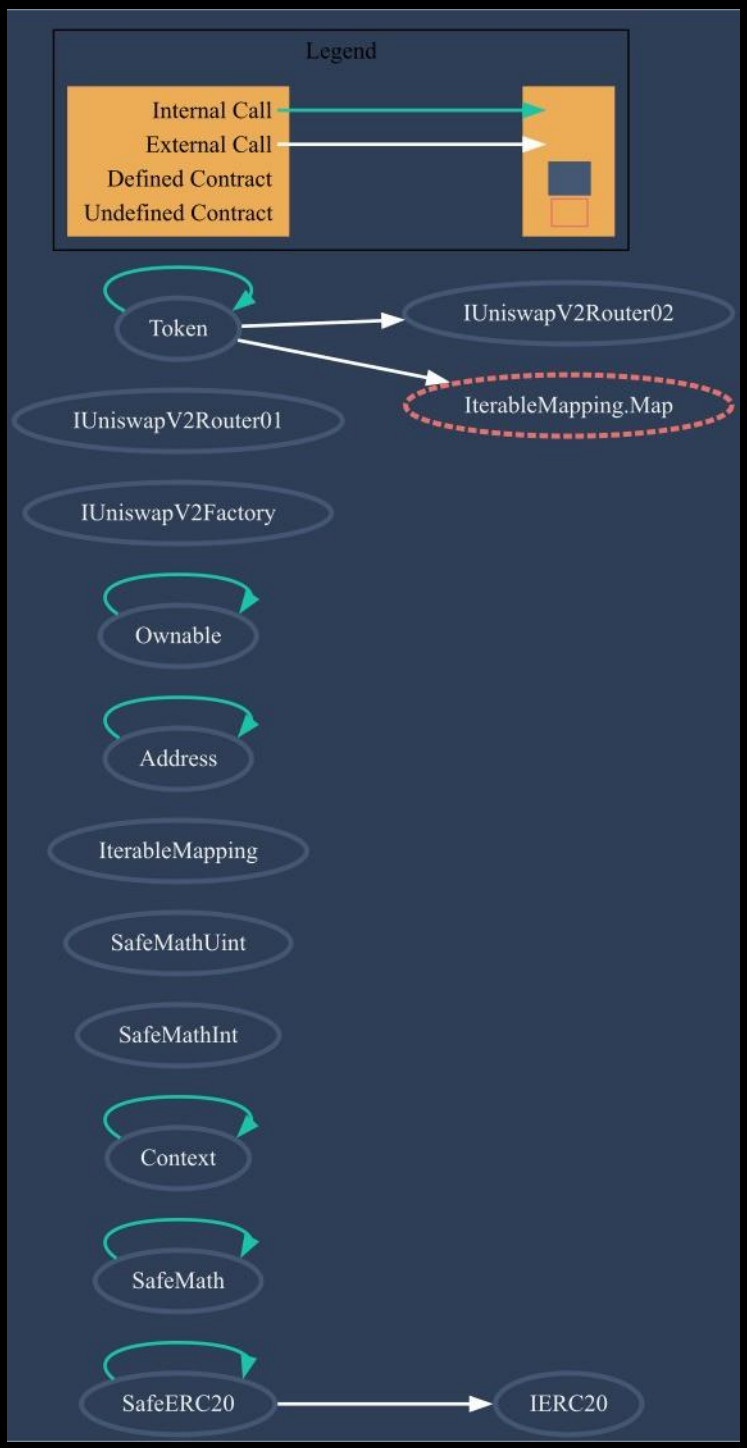
Function	The transferFrom() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account. This can be abused by a token receiver 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	3. 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. 4. Use transaction taxes to prevent against front-run attack



Contract Flow Graph



Contract Interaction Graph



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❗
L	name	Public ❗		NO❗

Contract	Type	Bases		
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			
L	feeTo	External ❗		NO❗

Contract	Type	Bases		
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