SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

King French Toast

KFT

BEP20

0x36cA7B4514bAE5c13a76a6fD200E02FC64557

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	DxFeeToken
Ticker/Simbol	KFT
Blockchain	Binance Smart Chain BEP20
Contract Address	0xAB1Ab4998E120C0377Ae01E0c38911DBa62391Af
Creator Address	0x87c6E6989E3Fe3828d87160f524B5BE2E78Ad784
Current Owner Address	0x0000000000000000000000000000000000000000
Contract Explorer	https://bscscan.com/address/0xab1ab4998e120c0377ae01e0c38911dba62391af#code
Compiler Version	v0.8.7+commit.e28d00a7
License	MIT
Optimisation	Yes with 200 Runs
Total Supply	200,000,000,000 **KFT**
Decimals	18

Creation/Audit

Contract Deployed	15.05.2024
Audit Created	16.05.2024
Audit Update	V 1.0

Verified Socials

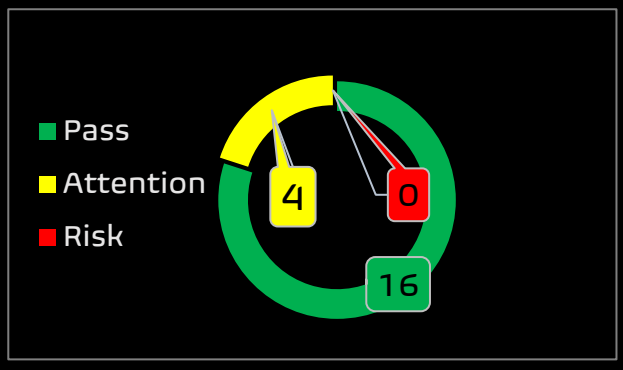
Website	https://kingfrenchtoast.com/
Telegram	https://t.me/kingfrenchtoastbnb
Twitter (X)	https://x.com/KingFrnchToast?s=09

SKELETON ECOSYSTEM
SMART CONTRACT AUDIT REPORT

KING FRENCH TOAST 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		0x0000000000000000000000000000000000000000 Sometimes referred to as the "zero address" or "dead address" and is not owned by anyone.
Buy Tax	6 %	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	6 %	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 16.05.2024 Lp Locked: 99.1% DX.app until 14.07.2026 (of initial LP Tokens) Lp Burned: 0.9% Note! Auto lp fee mechanism in code! lp lock-burn ratio will change
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	⚠ max 10%	Fee Setting function found. Contract renounced, function can not be triggered by owner. 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 with deployer authorisations!
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. Exclude from dividends only! Using this function Wallets may be set excluded from reveiving rewards. Contract renounced, function can not be triggered by owner.
Whitelist Function	⚠️	Whitelist Setting function found. Contract renounced, function can not be triggered by owner. 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 Contract renounced, function can not be triggered by owner.
Transaction Limiting Function	✅	No Transaction Limiter Function Found. The number of overall token transactions may be limited (honeypot risk)

Details of Risk - Attention Items

Removing Risk of contract function based on renounced ownership

Transaction Receipt Event Logs

228

Address 0xab1ab4998e120c0377ae01e0c38911dba62391af

Name OwnershipTransferred (index_topic_1 address previousOwner, index_topic_2 address newOwner) View Source

Topics
0 0x8be0079c531659141344cd1fd0a4f28419497f9722a3daafe3b4186f6b6457e0
1: previousOwner Dec → 0x87c6E6989E3Fe3828d87160f524B5BE2E78Ad784
2: newOwner Dec → 0x0000000000000000000000000000000000000000

Data 0x

Following detected contract functions serve as informational purposes about the contract. The owner has no more authorisation to trigger the following functions.

⚠ Set Fee 10% Max

Contract renounced, function can not be triggered by owner.

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 setTaxFeePercent(uint256 taxFee) external onlyOwner() {
    require(taxFee >= 0 && taxFee <=maxTaxFee,"taxFee out of range");
    _taxFee = taxFee;
    _previousTaxFee = _taxFee;
}

ftrace | funcSig
function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner() {
    require(liquidityFee >= 0 && liquidityFee <=maxLiqFee,"liquidityFee out of range");
    _liquidityFee = liquidityFee;
    _previousLiquidityFee = _liquidityFee;
}

ftrace | funcSig
function setDevFeePercent(uint256 devFee) external onlyOwner() {
    require(devFee >= 0 && devFee <=maxDevFee,"teamFee out of range");
    _devFee = devFee;
    _previousDevFee = _devFee;
}

ftrace | funcSig
function setSellTaxFeePercent(uint256 sellTaxFee) external onlyOwner() {
    require(sellTaxFee >= 0 && sellTaxFee <=maxSellTaxFee,"taxFee out of range");
    _sellTaxFee = sellTaxFee;
    _previousSellTaxFee = _sellTaxFee;
}

ftrace | funcSig
function setSellLiqFeePercent(uint256 sellLiqFee) external onlyOwner() {
    require(sellLiqFee >= 0 && sellLiqFee <=maxSellLiqFee,"taxFee out of range");
    _sellLiqFee = sellLiqFee;
    _previousSellLiqFee = _sellLiqFee;
}
```

⚠ Whitelist

Contract renounced, function can not be triggered by owner.

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 excludeFromFee(address account) public onlyOwner {
    require(!_isExcludedFromFee[account], "Account is already excluded");
    _isExcludedFromFee[account] = true;
}
```

⚠ Blacklist (Exclude from dividends only)

Using this function Wallets may be set excluded from reveiving rewards.

Contract renounced, function can not be triggered by owner.

```
ftrace | funcSig
    function excludeFromReward(address account) public onlyOwner {
    require(!_isExcluded[account], "Account is already excluded");
    if (_rOwned[account] > 0) {
        _tOwned[account] = tokenFromReflection(_rOwned[account]);
    }
    _isExcluded[account] = true;
    _excluded.push(account);
}
```

⚠ Max Transaction and Holding Modify Function

Contract renounced, function can not be triggered by owner.

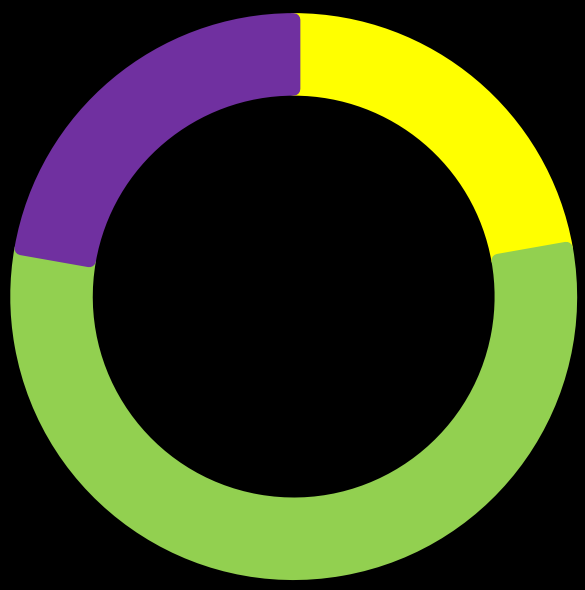
If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot

```
ftrace | funcSig
function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
    require(maxTxPercent >= minMxTxPercentage && maxTxPercent <=100,"maxTxPercent out of range");
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(10**2);
}
```

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)

- Incorrect Access Control
- Unchecked Array Lenght

Low severity issues: (5)

- Missing Events
- Long number literals
- Low level calls
- Approve of front running attack (Sandwich bots)
- Outdated Compiler Version

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	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	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.

⚠ Incorrect Access Control (2 Item)

Item: 1	Location:	Line 1121-1124	Severity:	Medium

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 DxFeeToken is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function approve is missing the modifier onlyOwner.
Remedation	1. Consider adding access control modifiers to the function to Ensure that initialization functions can only be called once and only by authorized entities. 2. Implement least-privilege roles using libraries like OpenZeppelin's Access Control. 3. Add proper access control modifiers to sensitive functions, such as onlyOwner or custom roles.

```
ftrace | funcSig
1121    function approve(address spender, uint256 amount) public override returns (bool) {
1122        _approve(_msgSender(), spender, amount);
1123        return true;
1124    }
1125
```

⚠ Incorrect Access Control (2 Item)

Item: 2	Location:	Line 1112-1115	Severity:	Medium

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 DxFeeToken is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function transfer is missing the modifier onlyOwner.
Remedation	4. Consider adding access control modifiers to the function to Ensure that initialization functions can only be called once and only by authorized entities. 5. Implement least-privilege roles using libraries like OpenZeppelin's Access Control. 6. Add proper access control modifiers to sensitive functions, such as onlyOwner or custom roles.

```
ftrace | funcSig
function transfer(address recipient, uint256 amount) public override returns (bool) {
    _transfer(_msgSender(), recipient, amount);
    return true;
}
```

⚠ Approve of front running attack (2 Items)

Item: 1	Location:	Line 1121-1124	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 override returns (bool) {
    _approve(_msgSender(), spender, amount);
    return true;
}
```

Item: 2	Location:	Line 1453-1515	Severity:	Low

Function	The swapTokensForEth() 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 swapTokensForEth 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

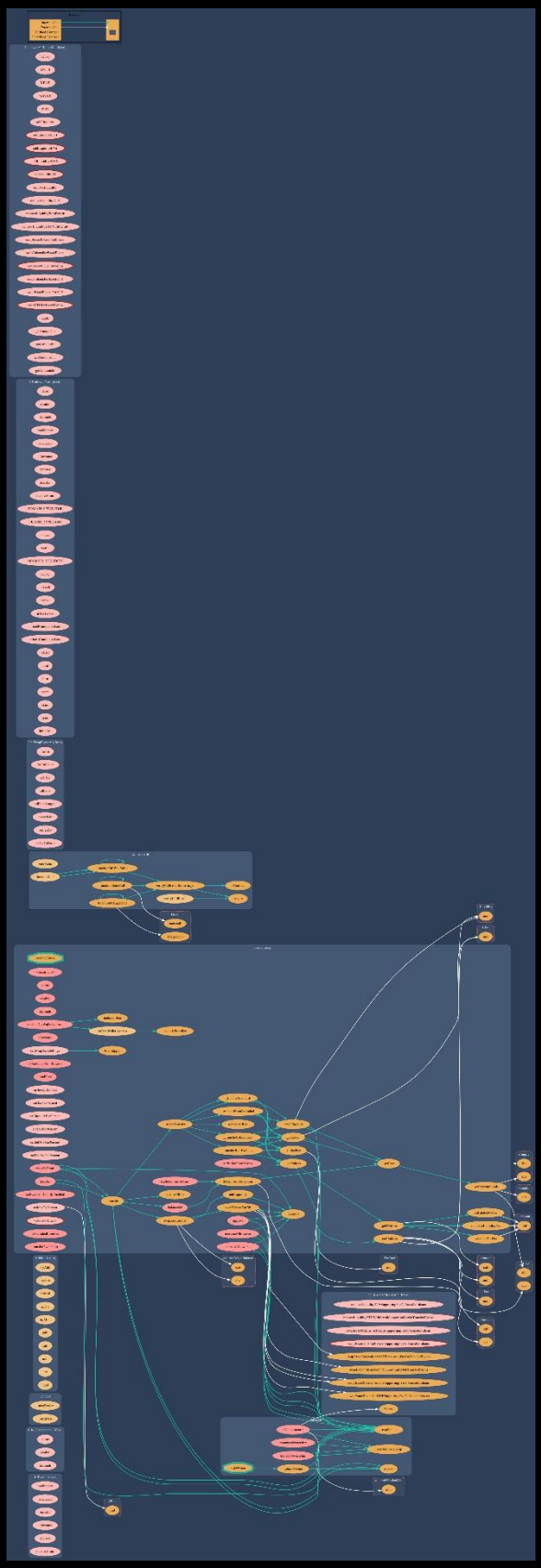
```
ftrace | funcSig
1453    function swapTokensForEth(uint256 tokenAmount1) private {
1454        // generate the uniswap pair path of token -> WHT
1455        address[] memory path = new address[](2);
1456        path[0] = address(this);
1457        path[1] = basePair;
1458
1459        _approve(address(this), address(uniswapV2Router), tokenAmount1);
1460
1461        // make the swap
1462        try uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
1463            tokenAmount1,
1464            0, // accept any amount of ETH
1465            path,
1466            address(this),
1467            block.timestamp
1468        ) {}
1469
1470        catch(bytes memory) {
1471
1472            try uniswapV2Router.swapExactTokensForBNBSupportingFeeOnTransferTokens(
1473                tokenAmount1,
1474                0, // accept any amount of ETH
1475                path,
1476                address(this),
1477                block.timestamp
1478            ) {}
1479            catch(bytes memory) {
1480                try uniswapV2Router.swapExactTokensForAVAXSupportingFeeOnTransferTokens(
1481                    tokenAmount1,
1482                    0, // accept any amount of ETH
1483                    path,
1484                    address(this),
1485                    block.timestamp
1486                ){}
1487
1488                catch(bytes memory) {
1489                    try uniswapV2Router.swapExactTokensForHTSupportingFeeOnTransferTokens(
1490                        tokenAmount1,
1491                        0, // accept any amount of ETH
1492                        path,
1493                        address(this),
1494                        block.timestamp
1495                    ){}
1496                    catch(bytes memory) {
1497
1498                        uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
1499                            tokenAmount1,
1500                            0, // accept any amount of ETH
1501                            path,
1502                            address(this),
1503                            block.timestamp
```

⚠ Outdated Compiler Version (1 Item)

Item: 1	Location:	Line 15	Severity:	Low

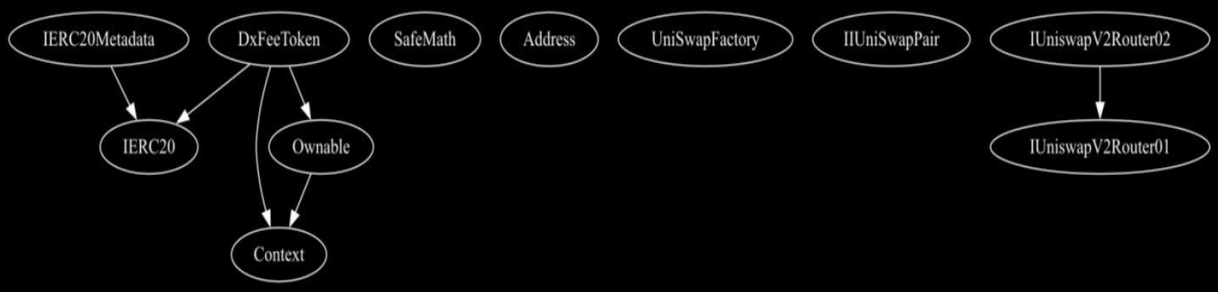
Function	Using an outdated compiler version can be problematic especially if there are publicly disclosed bugs and issues that affect the current compiler version. The following outdated versions were detected: /toast.sol - 0.8.7
Remedation	It is recommended to use a recent version of the Solidity compiler 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.

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		
SafeMath	Library			
L	tryAdd	Internal		
L	trySub	Internal		
L	tryMul	Internal		
L	tryDiv	Internal		
L	tryMod	Internal		
L	add	Internal		

Contract	Type	Bases		
L	sub	Internal 🔒		
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	mod	Internal 🔒		
L	sub	Internal 🔒		
L	div	Internal 🔒		
L	mod	Internal 🔒		
Ownable	Implementation	Context		
L		Public ❗	🛑	NO❗
L	owner	Public ❗		NO❗
L	_checkOwner	Internal 🔒		
L	renounceOwnership	Public ❗	🛑	onlyOwner
L	transferOwnership	Public ❗	🛑	onlyOwner
L	_transferOwnership	Internal 🔒	🛑	
Address	Library			
L	isContract	Internal 🔒		
L	sendValue	Internal 🔒	🛑	
L	functionCall	Internal 🔒	🛑	
L	functionCall	Internal 🔒	🛑	
L	functionCallWithValue	Internal 🔒	🛑	
L	functionCallWithValue	Internal 🔒	🛑	
L	functionStaticCall	Internal 🔒		
L	functionStaticCall	Internal 🔒		

Contract	Type	Bases		
L	functionDelegateCall	Internal		
L	functionDelegateCall	Internal		
L	verifyCallResultFrom Target	Internal		
L	verifyCallResult	Internal		
L	_revert	Private		
UniSwapFactory	Interface			
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!
IIUniSwapPair	Interface			
L	name	External		NO!
L	symbol	External		NO!
L	decimals	External		NO!
L	totalSupply	External		NO!
L	balanceOf	External		NO!
L	allowance	External		NO!
L	approve	External		NO!
L	transfer	External		NO!
L	transferFrom	External		NO!

Contract	Type	Bases		
L	DOMAIN_SEPARATOR	External		NO
L	PERMIT_TYPEHASH	External		NO
L	nonces	External		NO
L	permit	External		NO
L	MINIMUM_LIQUIDITY	External		NO
L	factory	External		NO
L	token0	External		NO
L	token1	External		NO
L	getReserves	External		NO
L	price0CumulativeLast	External		NO
L	price1CumulativeLast	External		NO
L	kLast	External		NO
L	mint	External		NO
L	burn	External		NO
L	swap	External		NO
L	skim	External		NO
L	sync	External		NO
L	initialize	External		NO
IUniswapV2Router01	Interface			
L	factory	External		NO
L	WETH	External		NO
L	WBNB	External		NO
L	WAVAX	External		NO

Contract	Type	Bases		
L	WHT	External !		NO!
L	addLiquidity	External !		NO!
L	addLiquidityETH	External !		NO!
L	addLiquidityBNB	External !		NO!
L	addLiquidityAVAX	External !		NO!
L	addLiquidityHT	External !		NO!
L	removeLiquidity	External !		NO!
L	removeLiquidityETH	External !		NO!
L	removeLiquidityWith Permit	External !		NO!
L	removeLiquidityETH WithPermit	External !		NO!
L	swapExactTokensFor Tokens	External !		NO!
L	swapTokensForExact Tokens	External !		NO!
L	swapExactETHForTok ens	External !		NO!
L	swapTokensForExact ETH	External !		NO!
L	swapExactTokensFor ETH	External !		NO!
L	swapETHForExactTok ens	External !		NO!
L	quote	External !		NO!
L	getAmountOut	External !		NO!
L	getAmountIn	External !		NO!
L	getAmountsOut	External !		NO!
L	getAmountsIn	External !		NO!

Contract	Type	Bases		
IUniswapV2Router02	Interface	IUniswapV2Router01		
L	removeLiquidityETHSupportingFeeOnTransferTokens	External !		NO!
L	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External !		NO!
L	swapExactTokensForTokensSupportingFeeOnTransferTokens	External !		NO!
L	swapExactETHForTokensSupportingFeeOnTransferTokens	External !		NO!
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External !		NO!
L	swapExactTokensForBNBSupportingFeeOnTransferTokens	External !		NO!
L	swapExactTokensForAVAXSupportingFeeOnTransferTokens	External !		NO!
L	swapExactTokensForHTSupportingFeeOnTransferTokens	External !		NO!
DxFeeToken	Implementation	Context, IERC20, Ownable		
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!

Contract	Type	Bases		
L	allowance	Public		NO
L	approve	Public		NO
L	transferFrom	Public		NO
L	increaseAllowance	Public		NO
L	decreaseAllowance	Public		NO
L	isExcludedFromReward	Public		NO
L	totalFees	Public		NO
L	reflectionFromToken	Public		NO
L	tokenFromReflection	Public		NO
L	excludeFromReward	Public		onlyOwner
L	includeInReward	External		onlyOwner
L	excludeFromFee	Public		onlyOwner
L	includeInFee	Public		onlyOwner
L	setTaxFeePercent	External		onlyOwner
L	setLiquidityFeePercent	External		onlyOwner
L	setDevFeePercent	External		onlyOwner
L	setSellTaxFeePercent	External		onlyOwner
L	setSellLiqFeePercent	External		onlyOwner
L	setMaxTxPercent	External		onlyOwner
L	setDevWalletAddress	Internal		
L	replaceDevWalletAddress	Public		onlyOwner
L	setSwapAndLiquifyEnabled	Public		onlyOwner
L	setSwapBackSettings	External		onlyOwner

Contract	Type	Bases		
L		External !		NO!
L	_reflectFee	Private		
L	_getValues	Private		
L	_getTValues	Private		
L	_getRValues	Private		
L	_getRate	Private		
L	_getCurrentSupply	Private		
L	_takeLiquidity	Private		
L	_takeDev	Private		
L	calculateTaxFee	Private		
L	calculateLiquidityFee	Private		
L	calculateDevFee	Private		
L	removeAllFee	Public !		NO!
L	restoreAllFee	Public !		NO!
L	isExcludedFromFee	Public !		NO!
L	_approve	Private		
L	_transfer	Private		
L	swapAndLiquify	Private		lockTheSwap
L	swapTokensForEth	Private		
L	addLiquidity	Private		
L	_tokenTransfer	Private		
L	_transferStandard	Private		
L	_transferToExcluded	Private		
L	_transferFromExclude d	Private		

Contract	Type	Bases		
L	_transferBothExclude d	Private		
L	transferOwnership	Public		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