SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

0x Token

$0x

BEP20

0x2123356ca6e6e3b6c076e0c8f39edb046e4fcd

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	**Token0x**
Ticker/Simbol	0x Token
Blockchain	Binance Smart Chain BEP20
Contract Address	0x2123356ca6e6e3b6c076e0c8f39edb046e4fcd25
Creator Address	0x32F6FC650BdCB2FE05E99E2FDb8DF99B6fE816e7
Current Owner Address	0x0000000000000000000000000000000000000000
Contract Explorer	https://bscscan.com/token/0x2123356ca6E6E3b6C076e0c8F39eDb046E4fcD25#code
Compiler Version	v0.8.24+commit.e11b9ed9
License	Unlicense
Optimisation	Yes with 240 Runs
Total Supply	420,000**0x Token**
Decimals	9

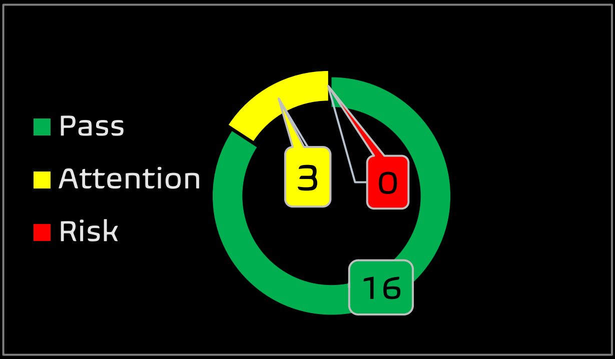
Creation/Audit

Contract Deployed	02.07.2024
Audit Created	05.07.2024
Audit Update	V 1.0

Verified Socials

Website	
Telegram	https://t.me/Entry0x
Twitter (X)	https://twitter.com/Entry0x

0x Token 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
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	10 %	Shows the taxes for sell transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable. Fee can be set!
Honeypot Analyse	✓	Holder is able to buy and sell. If honeypot: The contract blocks sell transfer from holder wallet. Multiple events may cause honeypot. Trading disabled, extremely high tax
Liqudity Status	✓	Liqudity status on 04.07.2024 98,9% Locked for 82 Days on Pinksale Locker https://bscscan.com/tx/0xca7c457bcf72990d8aac6e4927698d0b74c67ce2afe2ec955dbecf13d852d80e#eventlog
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. 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
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 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. 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. 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
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

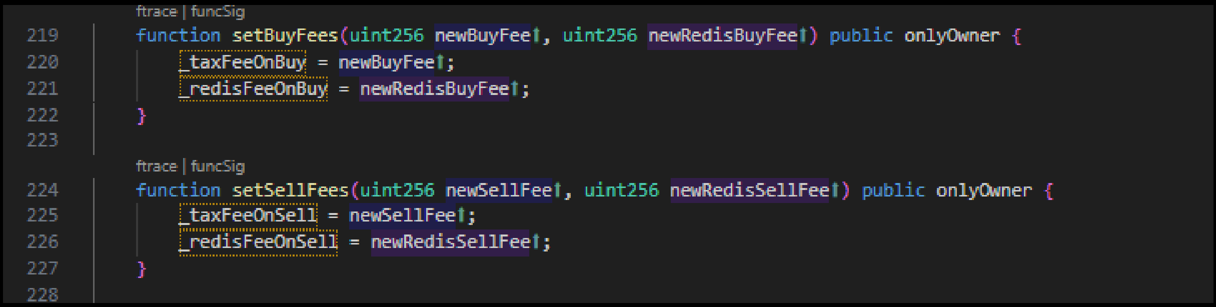
https://bscscan.com/tx/0xfc58fcc82d7400cc1767c36a90fdd14ad50684f59d315eb0dd0ce280591387bd

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

Set Fee

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).



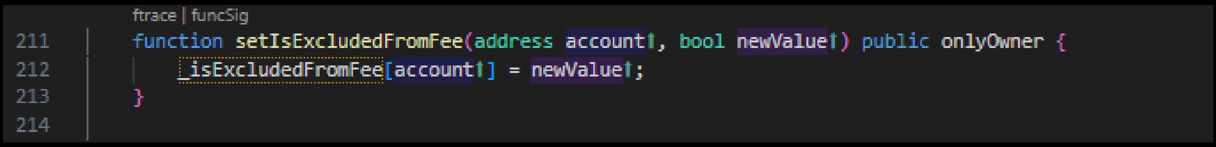
```
ftrace | funcSig
function setBuyFees(uint256 newBuyFee, uint256 newRedisBuyFee) public onlyOwner {
    _taxFeeOnBuy = newBuyFee;
    _redisFeeOnBuy = newRedisBuyFee;
}

ftrace | funcSig
function setSellFees(uint256 newSellFee, uint256 newRedisSellFee) public onlyOwner {
    _taxFeeOnSell = newSellFee;
    _redisFeeOnSell = newRedisSellFee;
}
```

Whitelist (Set wallets excluded from fees)

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)

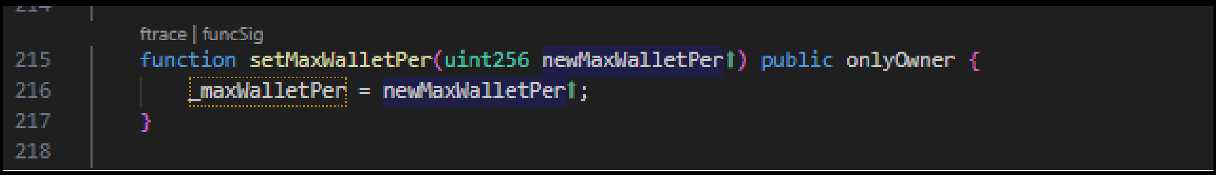


```
ftrace | funcSig
function setIsExcludedFromFee(address account, bool newValue) public onlyOwner {
    _isExcludedFromFee[account] = newValue;
}
```

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 setMaxWalletPer(uint256 newMaxWalletPer) public onlyOwner {
    _maxWalletPer = newMaxWalletPer;
}
```

Contract Security

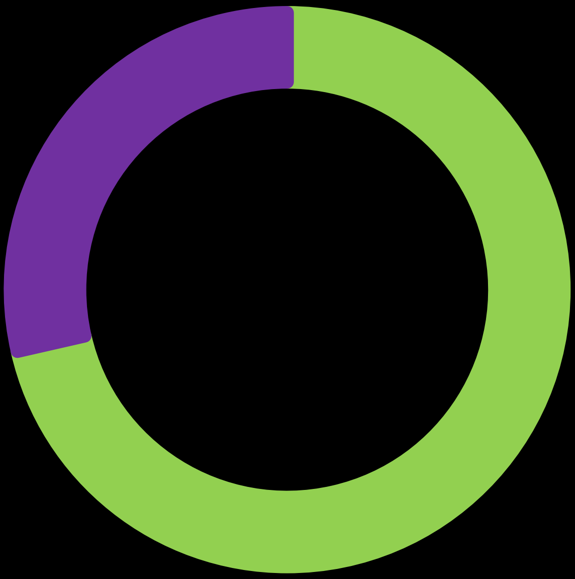
Total Findings: 7

🟥 High 0

🟨 Medium 0

🟩 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 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

🟥 High severity Issues: (0)

🟨 Medium severity issues: (0)

🟩 Low severity issues: (5)

- Missing Events
- Long number literals
- Outdated compiler Version
- Approve of Front Running Attack
- Floating Pragma

🟪 Informational severity issues: (2)

- Public Functions Should be Declared External
- State Variables Should be Declared Constant

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	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

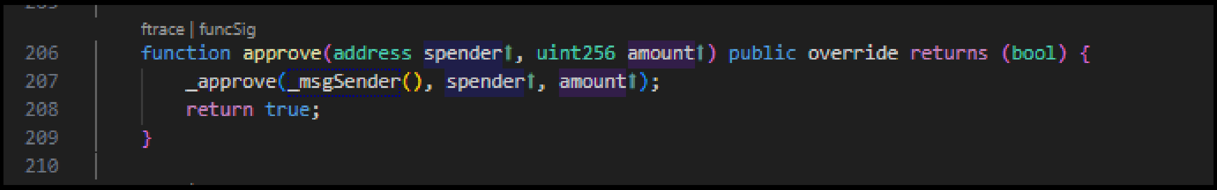
SWC-127	Arbitrary Jump with Function Type Variable	Passed	Passed	Passed
SWC-128	DoS With Block Gas Limit	Passed	Passed	Passed
SWC-129	Typographical Error	low	Passed	Passed
SWC-130	Right-To-Left-Override control character (U+202E)	Passed	Passed	Passed
SWC-131	Presence of unused variables	Passed	Passed	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.

⚠ Approve of Front running Attack (2 Item)

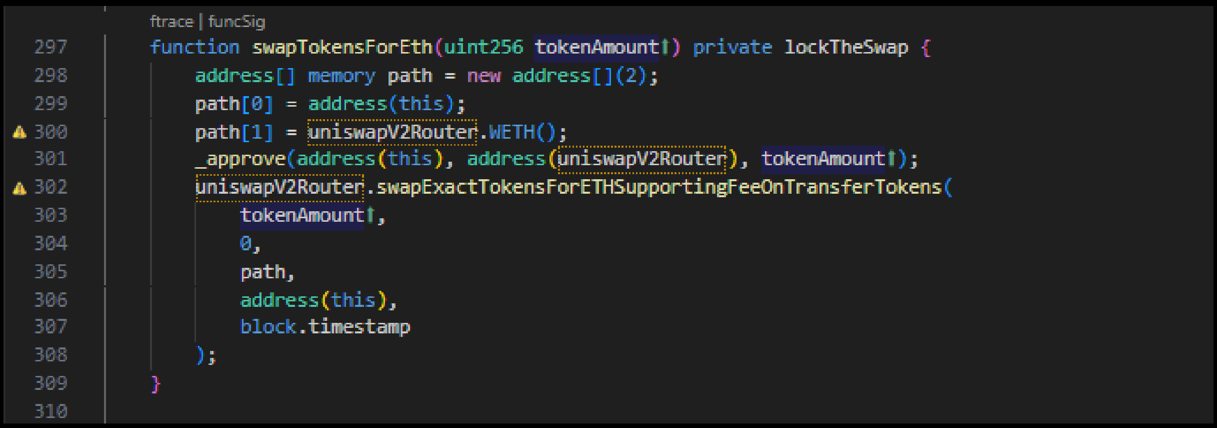
Item: 1	Location:	Line 206-209	Severity:	🟩 Low

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 297-309	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



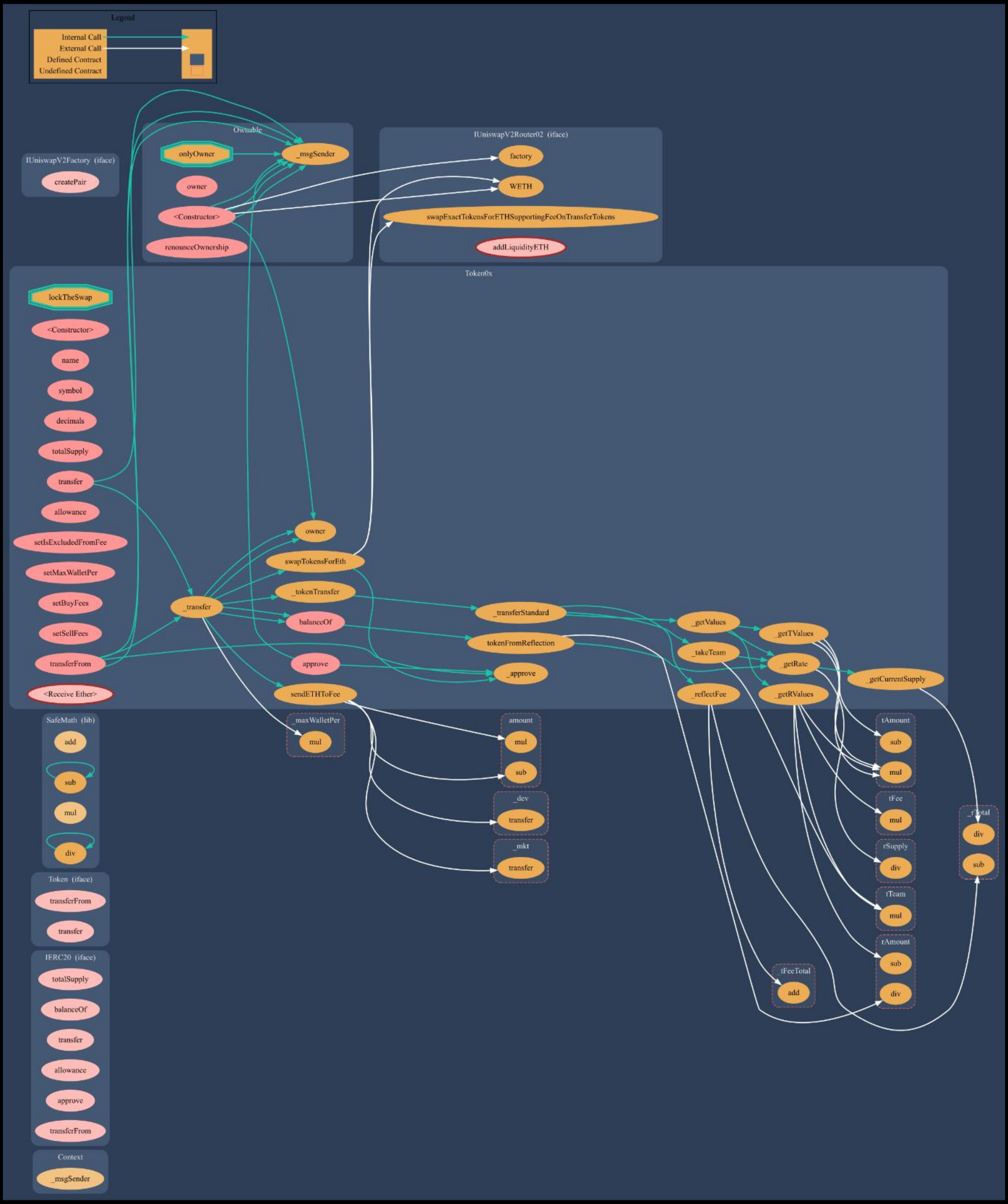
```
        ftrace | funcSig
297     function swapTokensForEth(uint256 tokenAmount) private lockTheSwap {
298         address[] memory path = new address[](2);
299         path[0] = address(this);
300         path[1] = uniswapV2Router.WETH();
301         _approve(address(this), address(uniswapV2Router), tokenAmount);
302         uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(
303             tokenAmount,
304             0,
305             path,
306             address(this),
307             block.timestamp
308         );
309     }
310
```

⚠ Outdated Compiler Version. (1 Item)

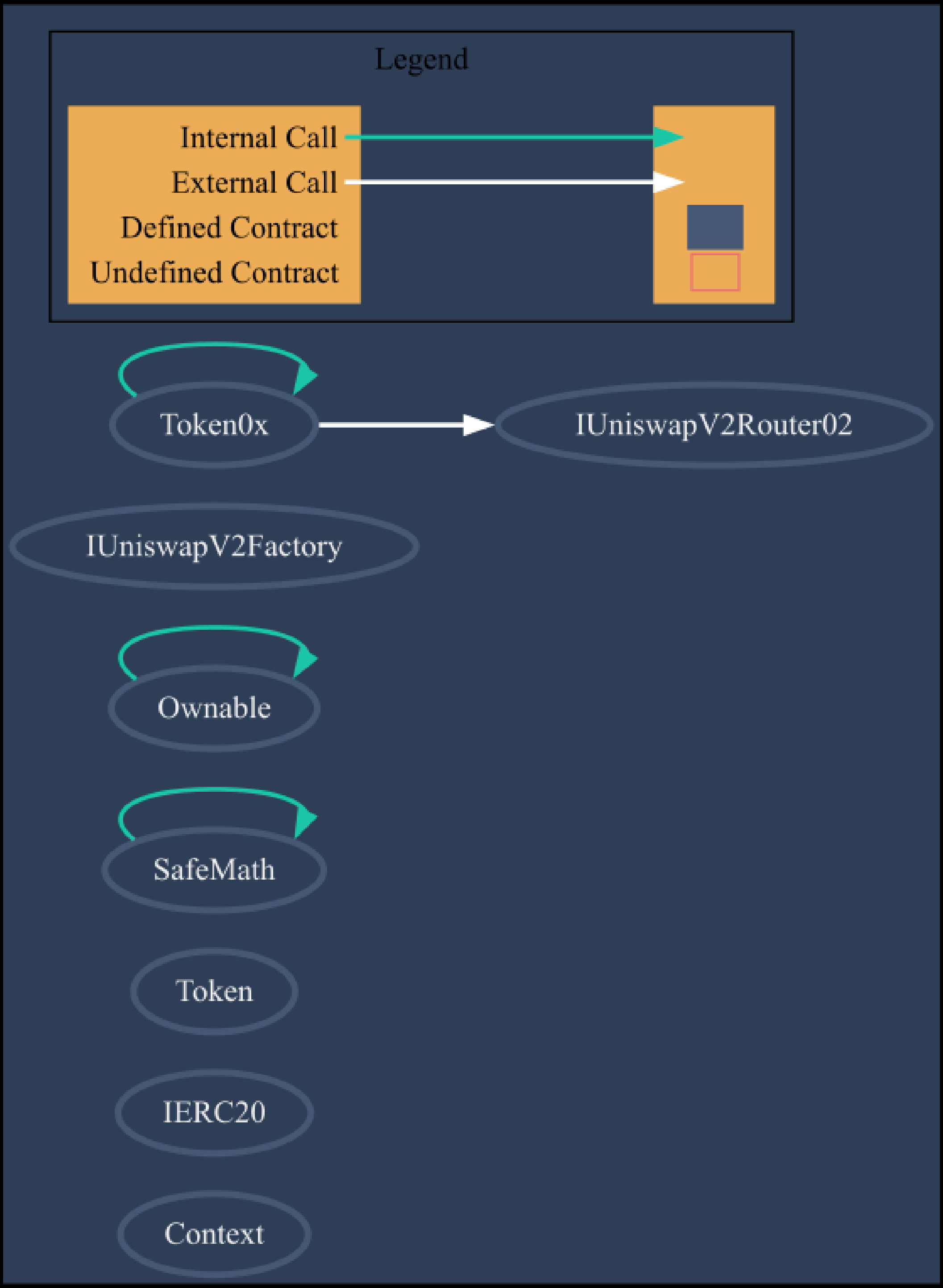
Item: 1	Location:	Line 7	Severity:	🟩 Low

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: /0xtoken.sol - ^0.8.4
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 v0.8.25, which patches most solidity vulnerabilities.

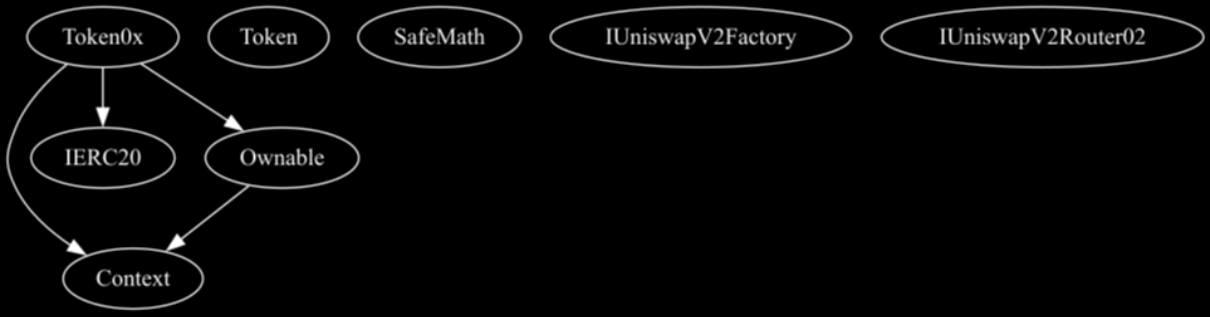
Contract Flow Graph



Contract Interaction Graph



Inheritance Graph



Contract Functions

Contract	Type	Bases		
└	Function Name	Visibility	Mutability	Modifiers
Context	Implementation			
└	_msgSender	Internal 🔒		
IERC20	Interface			
└	totalSupply	External ❗		NO❗
└	balanceOf	External ❗		NO❗
└	transfer	External ❗	🛑	NO❗
└	allowance	External ❗		NO❗
└	approve	External ❗	🛑	NO❗
└	transferFrom	External ❗	🛑	NO❗
Token	Interface			
└	transferFrom	External ❗	🛑	NO❗
└	transfer	External ❗	🛑	NO❗
SafeMath	Library			
└	add	Internal 🔒		
└	sub	Internal 🔒		
└	sub	Internal 🔒		
└	mul	Internal 🔒		
└	div	Internal 🔒		
└	div	Internal 🔒		
Ownable	Implementation	Context		
└		Public ❗	🛑	NO❗

└	owner	Public ❗️		NO❗️
└	renounceOwnership	Public ❗️	🛑	onlyOwner
IUniswapV2Factory	Interface			
└	createPair	External ❗️	🛑	NO❗️
IUniswapV2Router02	Interface			
└	swapExactTokensForETHSupportingFeeOnTransferTokens	External ❗️	🛑	NO❗️
└	factory	External ❗️		NO❗️
└	WETH	External ❗️		NO❗️
└	addLiquidityETH	External ❗️	💵	NO❗️
Token0x	Implementation	Context, IERC20, Ownable		
└		Public ❗️	🛑	NO❗️
└	name	Public ❗️		NO❗️
└	symbol	Public ❗️		NO❗️
└	decimals	Public ❗️		NO❗️
└	totalSupply	Public ❗️		NO❗️
└	balanceOf	Public ❗️		NO❗️
└	transfer	Public ❗️	🛑	NO❗️
└	allowance	Public ❗️		NO❗️
└	approve	Public ❗️	🛑	NO❗️
└	setIsExcludedFromFee	Public ❗️	🛑	onlyOwner
└	setMaxWalletPer	Public ❗️	🛑	onlyOwner
└	setBuyFees	Public ❗️	🛑	onlyOwner

└	setSellFees	Public ❗️	🛑	onlyOwner
└	transferFrom	Public ❗️	🛑	NO❗️
└	tokenFromReflection	Private 🔐		
└	_approve	Private 🔐	🛑	
└	_transfer	Private 🔐	🛑	
└	swapTokensForEth	Private 🔐	🛑	lockTheSwap
└	sendETHToFee	Private 🔐	🛑	
└	_tokenTransfer	Private 🔐	🛑	
└	_transferStandard	Private 🔐	🛑	
└	_takeTeam	Private 🔐	🛑	
└	_reflectFee	Private 🔐	🛑	
└		External ❗️	💵	NO❗️
└	_getValues	Private 🔐		
└	_getTValues	Private 🔐		
└	_getRValues	Private 🔐		
└	_getRate	Private 🔐		
└	_getCurrentSupply	Private 🔐		

🛑	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