

SKELETON ECOSYSTEM

SMART CONTRACT AUDIT



Fuuka Swap
[\$FUUKA]
BEP 20

0x1D34c2F892A274DA16b13509f3075eebEFf87c64



Table of Contents

Table of Contents	1
Disclaimer	2
Overview	3
Creation/Audit Date	3
Verified Socials	3
Contract Functions Analysis	4
Contract Safety and Weakness	7
Detected Vulnerability Description	11
Contract Flow Graph	15
Contract Interaction Graph	16
Inheritance Graph	17
Contract Descriptions	18
Audit Scope	24

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% safety 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	FUUKA SWAP
Ticker/Symbol	FUUKA
Blockchain	Binance Smart Chain BEP20
Contract Address	0x1D34c2F892A274DA16b13509f3075eebEFf87c64
Creator Address	0xfB10949fB74934Ed1B8688AC8DB0Cd1eAb8915c0
Current Owner Address	0xfB10949fB74934Ed1B8688AC8DB0Cd1eAb8915c0
Contract Explorer	https://bscscan.com/token/0x1d34c2f892a274da16b13509f3075eebeff87c64
Compiler Version	v0.8.16+commit.07a7930e
License	MIT
Optimisation	Yes with 9999 Runs
Total Supply	100,000,000 FUUKA
Decimals	9




Creation/Audit




Contract Deployed	08 July 2023
Audit Created	14 Nov 2023
Audit Update	V 1.0

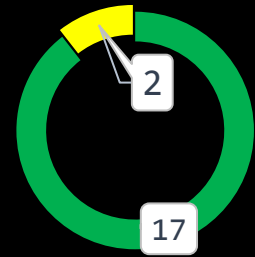
Verified Socials







Website	https://fuukaswap.online/
Telegram	https://t.me/fuukaswap
Twitter (X)	https://x.com/fuuka

Contract Function Analysis

 Pass
  Attention Item
  Risky Item

 Pass
 Attention
 Risky



Contract Verified		The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership		0xfB10949fB74934Ed1B8688AC8DB0Cd1eAb8915c0
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
Liquidity Status		Presale state
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		<p>No Balance Modifier function found.</p> <p>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.</p>
Blacklist Function		<p>No Blacklist Setting function found.</p> <p>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.</p>
Whitelist Function		<p>Whitelist Setting function found.</p> <p>If there is a function for this Developer can set zero fee or no max wallet size for addresses (for example team wallets can trade without fee. Can cause farming)</p>
Hidden Owner Analysis		<p>No hidden or multi owner</p> <p>For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned.</p>
Retrieve Ownership Function		<p>No functions found which can retrieve ownership of the contract.</p> <p>If this function exists, it is possible for the project owner to regain ownership even after relinquishing it. Also known as fake renounce.</p>
Self Destruct Function		<p>No Self Destruct function found.</p> <p>If this function exists and is triggered, the contract will be destroyed, all functions will be unavailable, and all related assets will be erased.</p>
Specific Tax Changing Function		<p>No Specific Tax Changing Functions found.</p> <p>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!</p>
Trading Cooldown Function		<p>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.</p>
Max Transaction and Holding Modify Function		<p>No Max Transaction and Holding Modify function found.</p> <p>If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot</p>
Transaction Limiting Function		<p>No Transaction Limiter Function Found.</p> <p>The number of overall token transactions may be limited (honeypot risk)</p>

Details of Risk - Attention Items

⚠ Set Fee (remediation: renounce ownership)

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

```

664      ftrace | funcSig
        function setTaxFeePercent(uint256 taxFeeBps!) external onlyOwner {
665          emit UpdatedTaxFeePercent(taxFeeBps!, taxFee);
666
667          taxFee = taxFeeBps!;
668
669          validateTaxes();
670      }
671
672      ftrace | funcSig
        function setLiquidityFeePercent(
673          uint256 liquidityFeeBps!
674      ) external onlyOwner {
675          emit UpdatedLiquidityFeePercent(liquidityFeeBps!, liquidityFee);
676
677          liquidityFee = liquidityFeeBps!;
678
679          validateTaxes();
680      }
681
682      ftrace | funcSig
        function setTeamFeePercent(uint256 teamFeeBps!) external onlyOwner {
683          emit UpdatedTeamFeePercent(teamFeeBps!, teamFee);
684
685          teamFee = teamFeeBps!;
686
687          validateTaxes();
688      }
  
```

⚠ Whitelist (Set excluded) (remediation: renounce ownership)

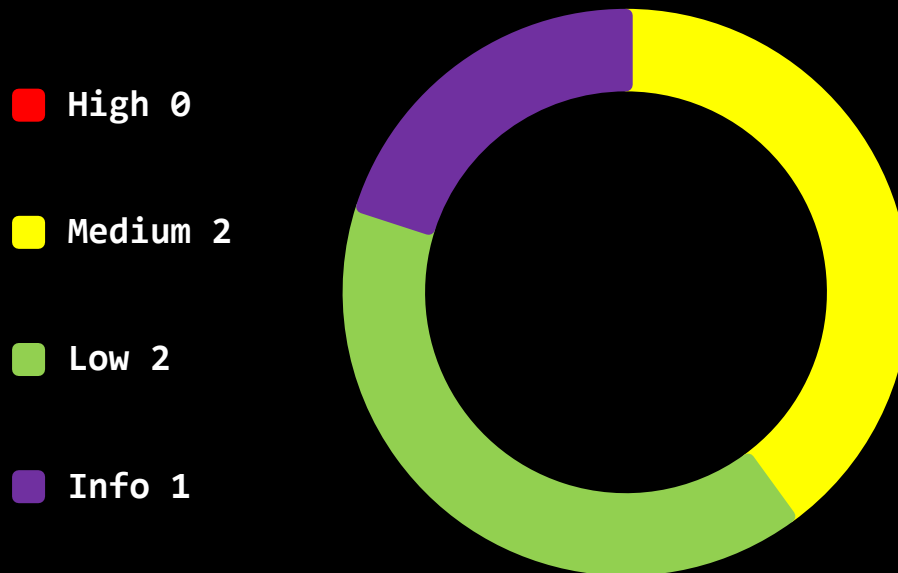
If there is a function for this Developer can set zero fee or no max wallet size for adresses (for example team wallets can trade without fee)


```


655
656      ftrace | funcSig
        function excludeFromFee(address account!) external onlyOwner {
657          isExcludedFromFee[account!] = true;
658      }
  
```


Contract Security


Total Findings: 5



 **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 SPECIFIC TO SMART CONTRACTS.

High severity Issues: (0)

Medium severity issues: (2)

- Unchecked Array Lenght
- Approve of Front Running Attack

Low severity issues: (2)

- Missing Events
- Outdated Compiler Version

Informational severity issues: (1)

- Public Functions Should be Declared External

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	Passed	Passed	Passed
SWC-104	Unchecked Call Return Value	Passed	Passed	Passed
SWC-105	Unprotected Ether Withdrawal	Passed	Passed	Passed
SWC-106	Unprotected SELFDESTRUCT Instruction	Passed	Passed	Passed
SWC-107	Reentrancy	Passed	Passed	Passed
SWC-108	State Variable Default Visibility	Passed	Passed	Passed
SWC-109	Uninitialized Storage Pointer	Passed	Passed	Passed
SWC-110	Assert Violation	Passed	Passed	Passed
SWC-111	Use of Deprecated Solidity Functions	Passed	Passed	Passed
SWC-112	Delegatecall to Untrusted Callee	Passed	Passed	Passed
SWC-113	DoS with Failed Call	Passed	Passed	Passed
SWC-114	Transaction Order Dependence	Passed	Passed	Passed
SWC-115	Authorization through tx.origin	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.

⚠️ Unchecked Array Length (1 Item)

Item: 1	Location: Line 621	Severity: ■ Medium
---------	--------------------	---

Function	<p>Ethereum is a very resource-constrained environment. Prices per computational step are orders of magnitude higher than with centralized providers. Moreover, Ethereum miners impose a limit on the total number of Gas consumed in a block. If array.length is large enough, the function exceeds the block gas limit, and transactions calling it will never be confirmed.</p> <pre>for (uint256 i = 0; i < array.length ; i++) { cosltyFunc(); }</pre> <p>This becomes a security issue if an external actor influences array.length. E.g., if an array enumerates all registered addresses, an adversary can register many addresses, causing the problem described above.</p>
Remediation	<p>Either explicitly or just due to normal operation, the number of iterations in a loop can grow beyond the block gas limit, which can cause the complete contract to be stalled at a certain point. Therefore, loops with a bigger or unknown number of steps should always be avoided.</p>

620	
621	<pre>for (uint256 i = 0; i < excluded.length; i++) {</pre>
622	<pre> if (excluded[i] == account1) {</pre>

Approve of front running attack (3 Items)

Item: 1	Location:	Line 510-516	Severity:	 Medium
---------	-----------	--------------	-----------	--

Function	<p>The <code>approve()</code> 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.</p> <p>The function approve can be front-run by abusing the <code>_approve</code> function.</p>
Remediation	<ol style="list-style-type: none"> 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

```

510         function approve(
511             address spender!,
512             uint256 amount!
513         ) external override returns (bool) {
514             _approve(_msgSender(), spender!, amount!);
515             return true;
516         }
517     
```

Item: 2	Location: Line 518-533	Severity: Medium
---------	------------------------	---

Function	<p>The <code>transferFrom()</code> 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 <code>transferFrom</code> can be front-run by abusing the <code>_approve</code> function.</p>
Remediation	<ol style="list-style-type: none"> 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

```

518     function transferFrom(
519         address sender!,
520         address recipient!,
521         uint256 amount!
522     ) external override returns (bool) {
523         _transfer(sender!, recipient!, amount!);
524         _approve(
525             sender!,
526             _msgSender(),
527             allowances[sender!][_msgSender()].sub(
528                 amount!,
529                 "ERC20: transfer amount exceeds allowance"
530             )
531         );
532         return true;
  
```

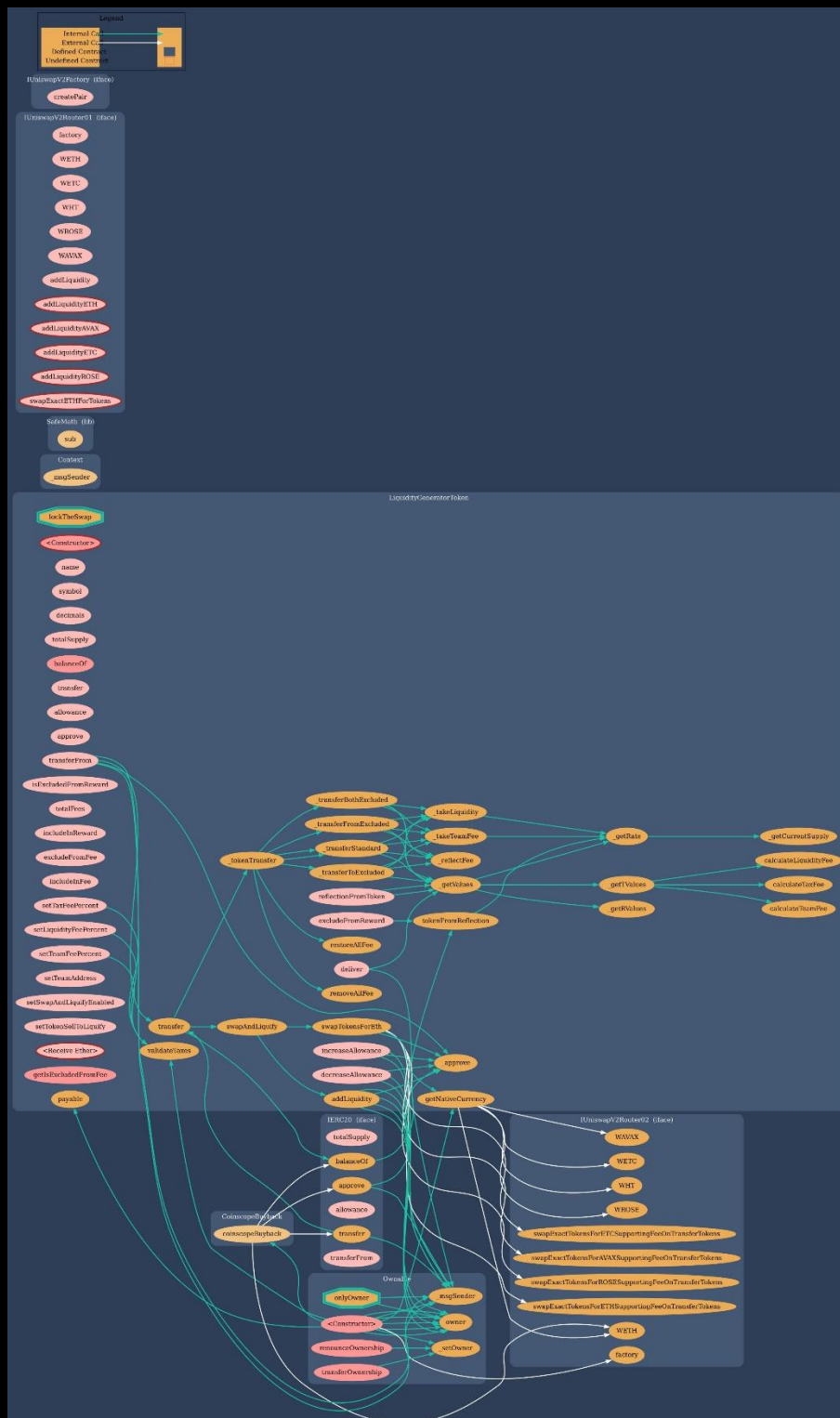
Item: 3	Location: Line 919-982	Severity: Medium
---------	------------------------	---

Function	<p>The <code>swapTokensForEth()</code> 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 <code>swapTokensForEth</code> can be front-run by abusing the <code>_approve</code> function.</p>
Remediation	<ol style="list-style-type: none"> 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

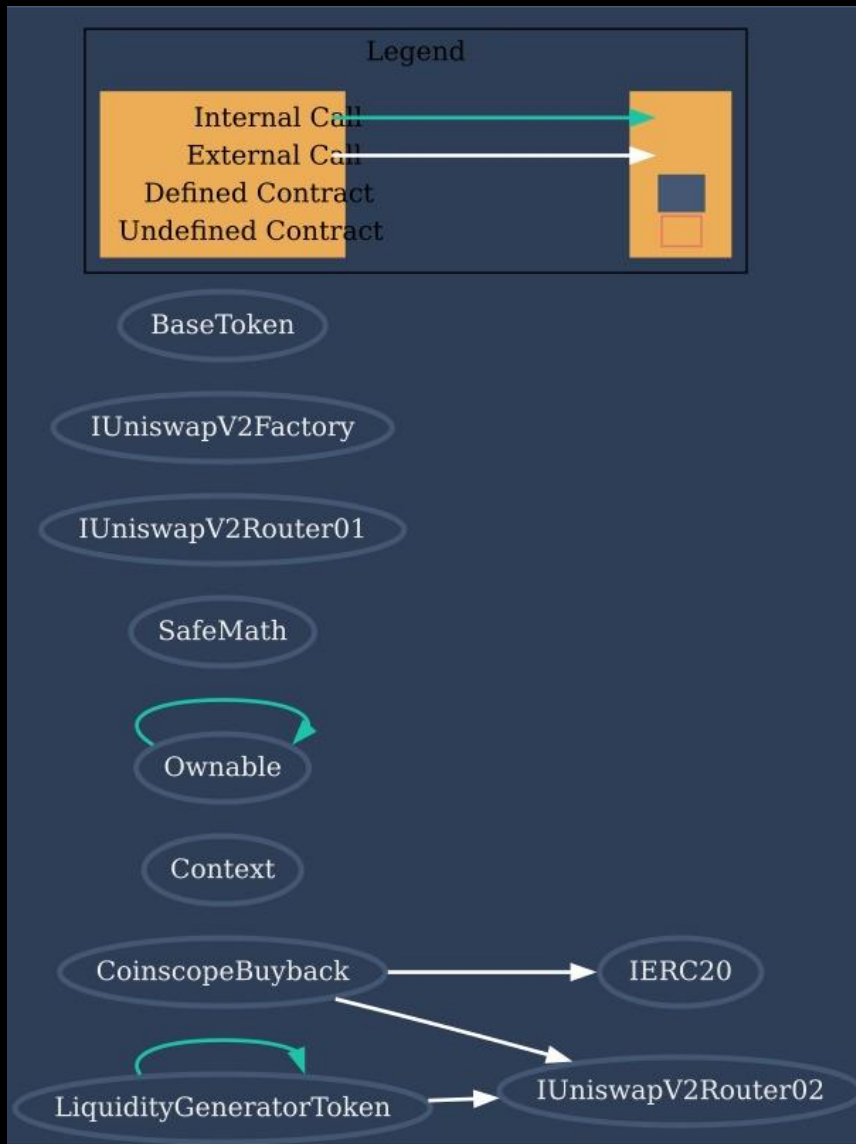
```

919 function swapTokensForEth(uint256 tokenAmount) private {
920     address[] memory path = new address[](2);
921     path[0] = address(this);
922     path[1] = getNativeCurrency();
923
924     _approve(address(this), address(uniswapV2Router), tokenAmount);
925
926     if (block.chainid == 61) {
927         //etc
928         try
929             uniswapV2Router
930                 .swapExactTokensForETCSupportingFeeOnTransferTokens(
931                     tokenAmount,
932                     0, // accept any amount of ETH
933                     path,
934                     address(this),
935                     block.timestamp
936                 )
937         {} catch {
938             emit SwapError(tokenAmount);
939         }
940     } else if (block.chainid == 42262) {
941         //oasis
942         try
943             uniswapV2Router
944                 .swapExactTokensForROSESupportingFeeOnTransferTokens(
945                     tokenAmount,
946                     0, // accept any amount of ETH
947                     path,
948                     address(this),
949                     block.timestamp
950                 )
951         {} catch {
952             emit SwapError(tokenAmount);
953         }
954     } else if (block.chainid == 43114 || block.chainid == 43113) {
955         //avalanche
956         try
957             uniswapV2Router
958                 .swapExactTokensForAVAXSupportingFeeOnTransferTokens(
959                     tokenAmount,
960                     0, // accept any amount of ETH
961                     path,
962                     address(this),
963                     block.timestamp
964                 )
965         {} catch {
966             emit SwapError(tokenAmount);
967         }
968     } else {
969         try
970             uniswapV2Router
971                 .swapExactTokensForETHSupportingFeeOnTransferTokens(
972                     tokenAmount,
973                     0, // accept any amount of ETH
974                     path,
975                     address(this),
976                     block.timestamp
977                 )
978         {} catch {
979             emit SwapError(tokenAmount);
980         }
981     }
982 }
  
```

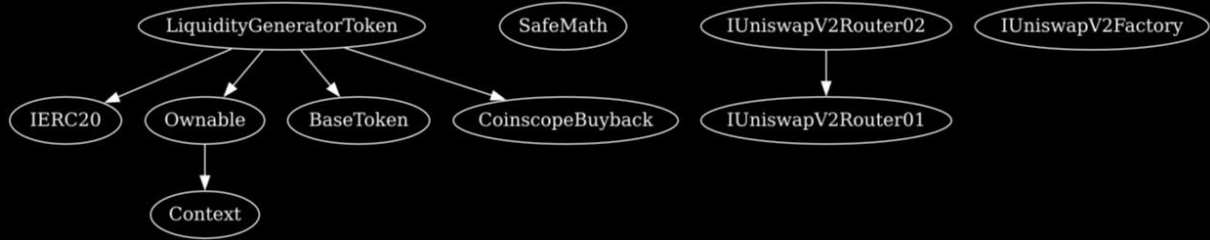
Contract Flow Graph































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 
Context	Implementation			
L	_msgSender	Internal 		
Ownable	Implementation	Context		
L		Public 		NO 
L	owner	Public 		NO 
L	renounceOwnership	Public 		onlyOwner
L	transferOwnership	Public 		onlyOwner
L	_setOwner	Private 		
SafeMath	Library			
L	sub	Internal 		

Contract	Type	Bases		
IUniswapV2Router01	Interface			
L	factory	External 		NO 
L	WETH	External 		NO 
L	WETC	External 		NO 
L	WHT	External 		NO 
L	WROSE	External 		NO 
L	WAVAX	External 		NO 
L	addLiquidity	External 		NO 
L	addLiquidityETH	External 		NO 
L	addLiquidityAVAX	External 		NO 
L	addLiquidityETC	External 		NO 
L	addLiquidityROSE	External 		NO 
L	swapExactETHForTokens	External 		NO 
IUniswapV2Router02	Interface	IUniswapV2Router01		
L	swapExactTokensForETCSupportingFeeOnTransferTokens	External 		NO 
L	swapExactTokensForAVAXSupportingFeeOnTransferTokens	External 		NO 

Contract	Type	Bases		
L	swapExactTokensForROSESupportingFeeOnTransferTokens	External 		NO 
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External 		NO 
IUniswapV2Factory	Interface			
L	createPair	External 		NO 
BaseToken	Implementation			
CoinscopeBuyback	Implementation			
L	coinscopeBuyback	Internal 		
LiquidityGeneratorToken	Implementation	IERC20, Ownable, BaseToken, CoinscopeBuyback		
L		Public 		NO 
L	getNativeCurrency	Internal 		
L	name	External 		NO 
L	symbol	External 		NO 
L	decimals	External 		NO 
L	totalSupply	External 		NO 

Contract	Type	Bases		
L	balanceOf	Public 🔒		NO 🔒
L	transfer	External 🔒	🔒	NO 🔒
L	allowance	External 🔒		NO 🔒
L	approve	External 🔒	🔒	NO 🔒
L	transferFrom	External 🔒	🔒	NO 🔒
L	increaseAllowance	External 🔒	🔒	NO 🔒
L	decreaseAllowance	External 🔒	🔒	NO 🔒
L	isExcludedFromReward	External 🔒		NO 🔒
L	totalFees	External 🔒		NO 🔒
L	deliver	External 🔒	🔒	NO 🔒
L	reflectionFromToken	External 🔒		NO 🔒
L	tokenFromReflection	Public 🔒		NO 🔒
L	excludeFromReward	External 🔒	🔒	onlyOwner
L	includeInReward	External 🔒	🔒	onlyOwner
L	_transferBothExcluded	Private 🔒	🔒	
L	excludeFromFee	External 🔒	🔒	onlyOwner
L	includeInFee	External 🔒	🔒	onlyOwner
L	setTaxFeePercent	External 🔒	🔒	onlyOwner

Contract	Type	Bases		
L	setLiquidityFeePercent	External !		onlyOwner
L	setTeamFeePercent	External !		onlyOwner
L	setTeamAddresses	External !		onlyOwner
L	validateTaxes	Internal 		
L	setSwapAndLiquifyEnabled	External !		onlyOwner
L	setTokenSellToLiquify	External !		onlyOwner
L		External !		NO!
L	_getValues	Private 		
L	_getTValues	Private 		
L	_getRValues	Private 		
L	_getRate	Private 		
L	_getCurrentSupply	Private 		
L	_takeLiquidity	Private 		
L	_takeTeamFee	Private 		
L	_reflectFee	Private 		
L	calculateTaxFee	Private 		
L	calculateLiquidityFee	Private 		
L	calculateTeamFee	Private 		

Contract	Type	Bases		
L	removeAllFee	Private 		
L	restoreAllFee	Private 		
L	getIsExcludedFromFee	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	_transferFromExcluded	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 vulnerabilities in the code. Findings getting reported and improvements getting suggested.

Automatic and Manual Review

We are using automated tools to scan functions and weaknesses 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>

