

SKELETON ECOSYSTEM

SMART CONTRACT AUDIT



Davey Nakamoto
DANA
BEP 20

0x40Fc9cBc0a90F65fB8B99d88c3468A676BF116f0



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	8
Detected Vulnerability Description	12
Contract Flow Graph	15
Contract Interaction Graph	16
Inheritance Graph	17
Contract Descriptions	18
Audit Scope	23

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	TOKEN
Ticker/Symbol	DANA
Blockchain	Binance Smart Chain BEP20
Contract Address	0x40Fc9cBc0a90F65fB8B99d88c3468A676BF116f0
Creator Address	0x56d2142B03f2f4220eBBd921400660B56DD3B825
Current Owner Address	0x56d2142B03f2f4220eBBd921400660B56DD3B825
Contract Explorer	https://bscscan.com/token/0x40fc9cbc0a90f65fb8b99d88c3468a676bf116f0#code
Compiler Version	v0.8.19+commit.7dd6d404
License	Unlicense
Optimisation	Yes with 200 Runs
Total Supply	50 DANA
Decimals	18




Creation/Audit

Contract Deployed	03 Oct 2023
Audit Created	11 Dec 2023
Audit Update	V 1.0








Verified Socials


Website	https://www.daveynakamoto.lol
Telegram	https://t.me/daveynakamoto
Twitter (X)	https://twitter.com/daveynakamoto

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		0x56d2142B03f2f4220eBBd921400660B56DD3B825
Buy Tax	5 %	Shows the taxes for purchase transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable. Fee can be set!
Sell Tax	5 %	Shows the taxes for sell transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable. Fee can be set!
Honeypot Analyse		Holder is able to buy and sell. If honeypot: The contract blocks sell transfer from holder wallet. Multiple events may cause honeypot. Trading disabled, extremely high tax
Liquidity Status		LP Lock Status on 10.12.2023: 90.36 % until 31.01.2024 on Gempad
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 15%	Fee Setting function found. Max 15% Set 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 with authorisation</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>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 (Reducing Risk: max 15% fee trigger)

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

```

266 // Set Buy and Sell Fees
    ftrace | funcSig
    function Launch_01_Set_Fees(
267
268         uint8 Marketing_on_BUY!,
269         uint8 Liquidity_on_BUY!,
270         uint8 Burn_on_BUY!,
271
272         uint8 Marketing_on_SELL!,
273         uint8 Liquidity_on_SELL!,
274         uint8 Burn_on_SELL!
275     ) external onlyOwner {
276
277         // Buyer Protection: Max Fee 15% (includes the 2% contract fee until paid)
278         require (Marketing_on_BUY! + Liquidity_on_BUY! + Burn_on_BUY! + _fee_Buy_Contract <= 15, "FEE1"); // Max fee 15%
279
280         // Buyer Protection: Max Fee 15% (includes the 2% contract fee until paid)
281         require (Marketing_on_SELL! + Liquidity_on_SELL! + Burn_on_SELL! + _fee_Sell_Contract <= 15, "FEE2"); // Max fee 15%
282
283         // Update Fees
284         _fee_Buy_Marketing = Marketing_on_BUY!;
285         _fee_Buy_Liquidity = Liquidity_on_BUY!;
286         _fee_Buy_Burn = Burn_on_BUY!;
287
288         _fee_Sell_Marketing = Marketing_on_SELL!;
289         _fee_Sell_Liquidity = Liquidity_on_SELL!;
290         _fee_Sell_Burn = Burn_on_SELL!;
291
292         // Fees For Processing
293         _SwapFeeTotal_Sell = _fee_Sell_Marketing + _fee_Sell_Liquidity + _fee_Buy_Contract;
294         _SwapFeeTotal_Buy = _fee_Buy_Marketing + _fee_Buy_Liquidity + _fee_Sell_Contract;
295
296         emit updated_Buy_fees(_fee_Buy_Marketing, _fee_Buy_Liquidity, _fee_Buy_Burn, _fee_Buy_Contract);
297         emit updated_Sell_fees(_fee_Sell_Marketing, _fee_Sell_Liquidity, _fee_Sell_Burn, _fee_Sell_Contract);
298
299     }
300
301 }
  
```

⚠ Whitelist Function

If there is a function for this, Developer can set zero fee or no max wallet size for addreses (for example team wallets can trade without fee. Can cause farming)

```

661
662 // Exclude From Fees
    ftrace | funcSig
    function Wallet_Exclude_From_Fees(
663
664         address Wallet_Address!,
665         bool true_or_false!
666     ) external onlyOwner {
667
668         _isExcludedFromFee[Wallet_Address!] = true_or_false!;
669
670     }
671
672 }
  
```

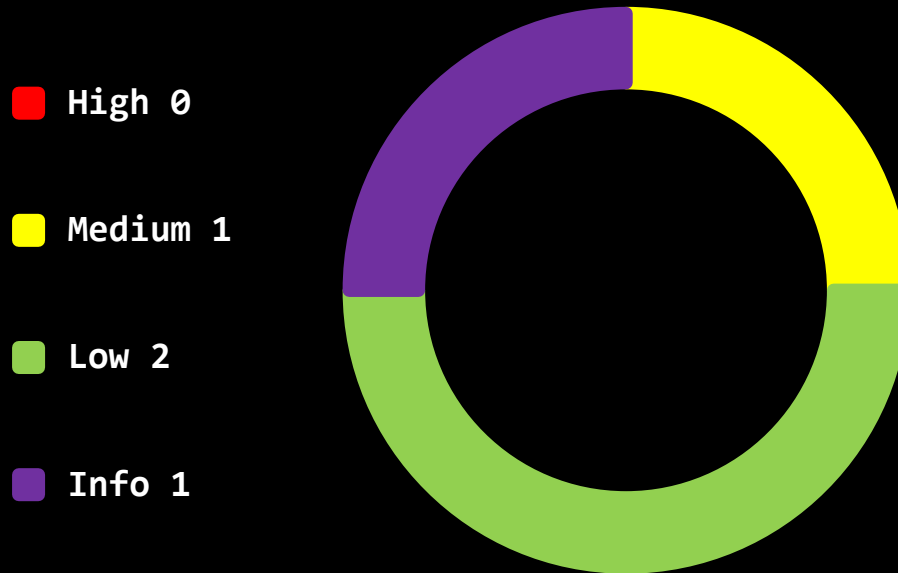
⚠ Max Transaction and Holding Modify Function


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


```
315  function Launch_02_Set_Wallet_Limits(  
316  
317      uint256 Max_Transaction_Percent!,  
318      uint256 Max_Wallet_Percent!  
319  ) external onlyOwner {  
320  
321      if (Max_Transaction_Percent! < 1){  
322          // Defaults to 0.5% if 0 is entered  
323          max_Tran = tTotal / 200;  
324      } else {  
325          max_Tran = tTotal * Max_Transaction_Percent! / 100;  
326      }  
327  
328      if (Max_Wallet_Percent! < 1){  
329          // Defaults to 0.5% if 0 is entered  
330          max_Hold = tTotal / 200;  
331      } else {  
332          max_Hold = tTotal * Max_Wallet_Percent! / 100;  
333      }  
334  
335      emit updated_Wallet_Limits(max_Tran, max_Hold);  
336  }
```



Contract Security


Total Findings: 4



 **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)

- Approve of Front running Attack

Low severity issues: (2)

- Missing Events
- Long Number Literals

Informational severity issues: (1)

- Public Functions Should be Declared External

Contract Weakness Classification

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.

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	Passed	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	Passed	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 (3 Item)

Item: 1	Location: Line 780-783	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	<p>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.</p> <p>2. Use transaction taxes to prevent against front-run attack</p>

```

780  ftrace | funcSig
781  function approve(address spender!, uint256 amount!) public override returns (bool) {
782      _approve(_msgSender(), spender!, amount!);
783      return true;
784  }
  
```

Item: 2	Location:	Line 799-807	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.</p> <p>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-runattack

```

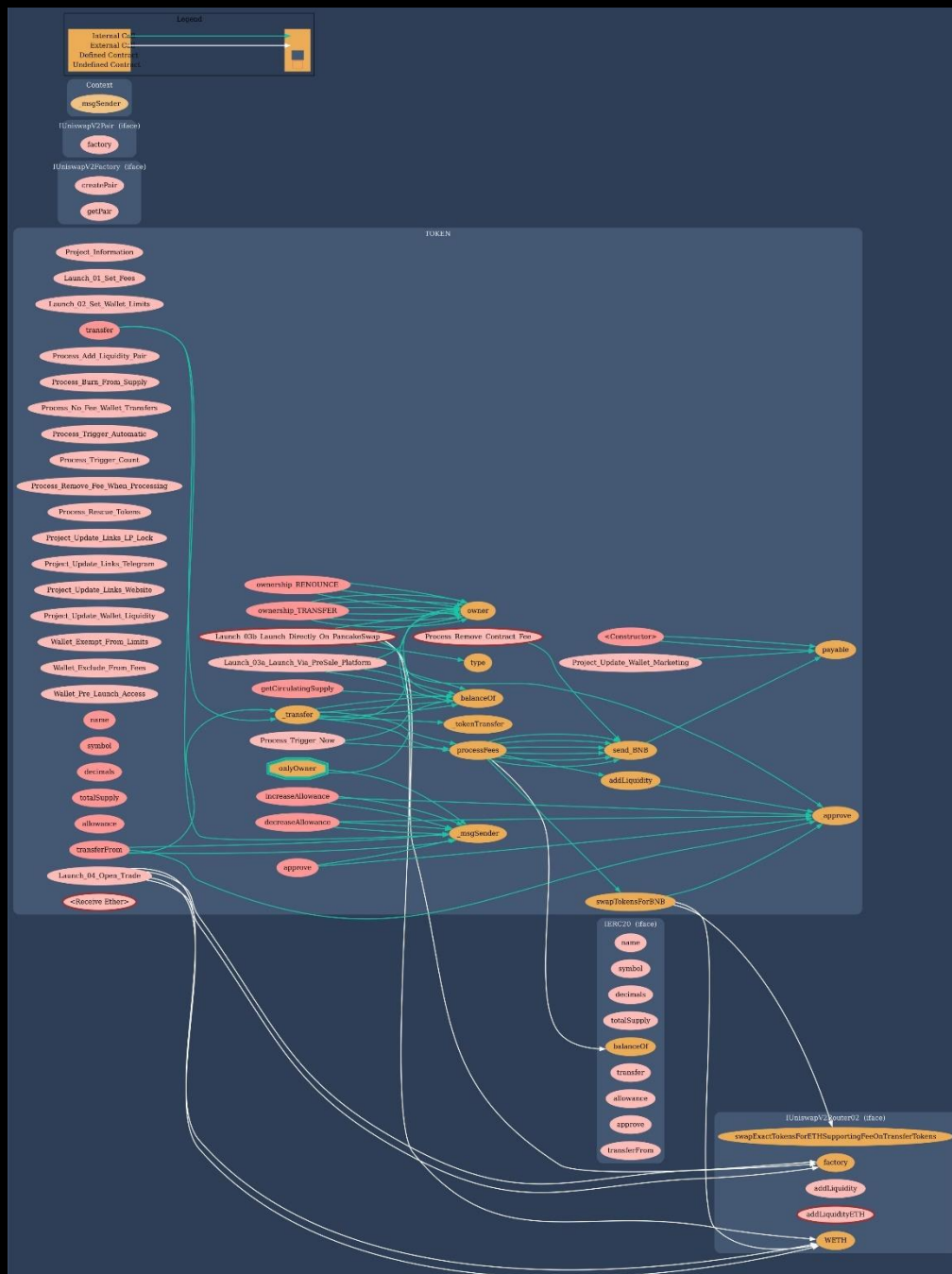
799  fttrace | funcSig
800  function transferFrom(address sender!, address recipient!, uint256 amount!) public virtual override returns (bool) {
801      _transfer(sender!, recipient!, amount!);
802
803      uint256 currentAllowance = _allowances[sender!][_msgSender()];
804      require(currentAllowance >= amount!, "ERC20: transfer amount exceeds allowance");
805      _approve(sender!, _msgSender(), currentAllowance - amount!);
806
807      return true;
808  }
  
```

Item: 3	Location: 1019-1033	Severity: ■ Medium
---------	---------------------	--

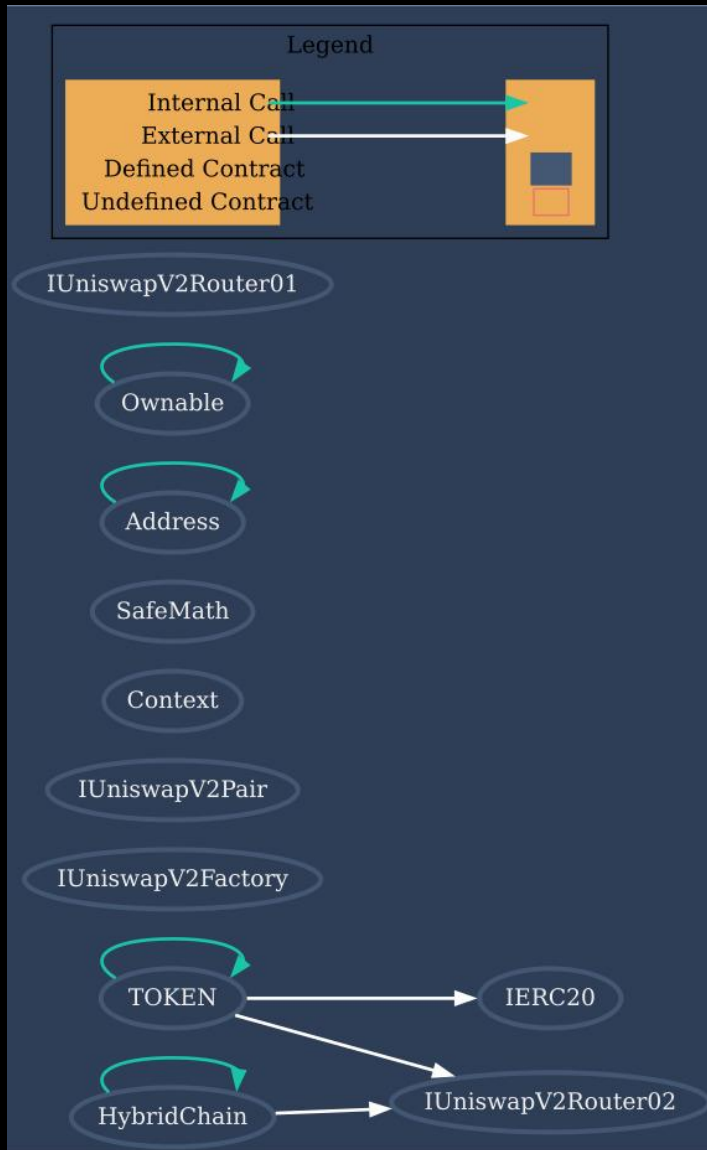
Function	<p>The <code>_swapTokensForBNB</code> method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account.</p> <p>This can be 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

	<pre> ftrace funcSig 1019 function swapTokensForBNB(uint256 tokenAmount) private { 1020 1021 address[] memory path = new address[](2); 1022 path[0] = address(this); 1023 path[1] = uniswapV2Router.WETH(); 1024 _approve(address(this), address(uniswapV2Router), tokenAmount); 1025 uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens(1026 tokenAmount, 1027 0, 1028 path, 1029 address(this), 1030 block.timestamp 1031); 1032 } 1033 </pre>
--	--

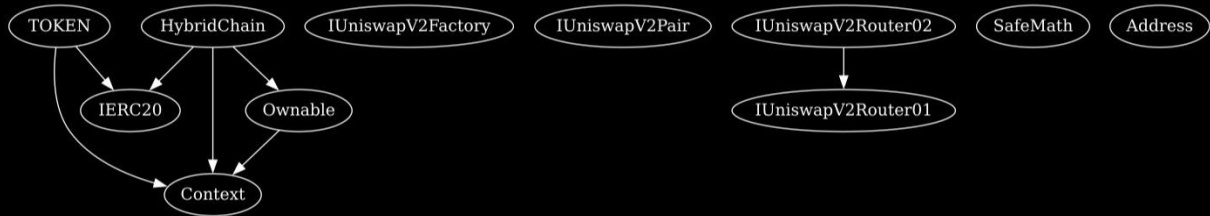
Contract Flow Graph

































Contract Interaction Graph












































Inheritance Graph



























































Contract Functions

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
IERC20	Interface			
L	name	External 		NO 
L	symbol	External 		NO 
L	decimals	External 		NO 
L	totalSupply	External 		NO 
L	balanceOf	External 		NO 
L	transfer	External 		NO 
L	allowance	External 		NO 
L	approve	External 		NO 
L	transferFrom	External 		NO 
IUniswapV2Factory	Interface			
L	createPair	External 		NO 
L	getPair	External 		NO 
IUniswapV2Pair	Interface			
L	factory	External 		NO 
IUniswapV2Router02	Interface			
L	factory	External 		NO 

Contract	Type	Bases		
L	WETH	External 		NO 
L	addLiquidity	External 		NO 
L	addLiquidityETH	External 		NO 
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External 		NO 
Context	Implementation			
L	_msgSender	Internal 		
TOKEN	Implementation	Context, IERC20		
L		Public 		NO 
L	Project_Information	External 		NO 
L	Launch_01_Set_Fees	External 		onlyOwner
L	Launch_02_Set_Wallet_Limits	External 		onlyOwner
L	Launch_03a_Launch_Via_PreSale_Platform	External 		onlyOwner
L	Launch_03b_Launch_Directly_On_PancakeSwap	External 		onlyOwner
L	Launch_04_Open_Trade	External 		onlyOwner
L	Process_Add_Liquidity_Pair	External 		onlyOwner

Contract	Type	Bases		
L	Process_Burn_From_Supply	External 		onlyOwner
L	Process_No_Fee_Wallet_Transfers	External 		onlyOwner
L	Process_Trigger_Automatic	External 		onlyOwner
L	Process_Trigger_Count	External 		onlyOwner
L	Process_Trigger_Now	External 		onlyOwner
L	Process_Remove_Contract_Fee	External 		NO 
L	Process_Remove_Fee_When_Processing	External 		onlyOwner
L	Process_Rescue_Tokens	External 		onlyOwner
L	Project_Update_Links_LP_Lock	External 		onlyOwner
L	Project_Update_Links_Telegram	External 		onlyOwner
L	Project_Update_Links_Website	External 		onlyOwner
L	Project_Update_Wallet_Liquidity	External 		onlyOwner
L	Project_Update_Wallet_Marketing	External 		onlyOwner
L	Wallet_Exempt_From_Limits	External 		onlyOwner

Contract	Type	Bases		
L	Wallet_Exclude_From_Fees	External 		onlyOwner
L	Wallet_Pre_Launch_Access	External 		onlyOwner
L	ownership_RENOUNCE	Public 		onlyOwner
L	ownership_TRANSFER	Public 		onlyOwner
L	owner	Public 		NO 
L	name	Public 		NO 
L	symbol	Public 		NO 
L	decimals	Public 		NO 
L	totalSupply	Public 		NO 
L	balanceOf	Public 		NO 
L	allowance	Public 		NO 
L	increaseAllowance	Public 		NO 
L	decreaseAllowance	Public 		NO 
L	approve	Public 		NO 
L	_approve	Private 		
L	transfer	Public 		NO 
L	transferFrom	Public 		NO 
L	send_BNB	Internal 		
L	getCirculatingSupply	Public 		NO 

Contract	Type	Bases		
L	_transfer	Private 		
L	processFees	Private 		
L	swapTokensFor BNB	Private 		
L	addLiquidity	Private 		
L	_tokenTransfer	Private 		
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 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>

