

SKELETON ECOSYSTEM

SMART CONTRACT AUDIT



Royal Shiba
[ROYALSHIBA]
BEP 20

0x8E84d99561b76cBc46b21d40e8dd062976f28CDA



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	6
Detected Vulnerability Description	10
Contract Flow Graph	18
Inheritance Graph	19
Contract Descriptions	20
Audit Scope	28

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	Royal Shiba
Ticker/Symbol	ROYALSHIBA
Blockchain	Binance Smart Chain BEP20
Contract Address	0x8E84d99561b76cBc46b21d40e8dd062976f28CDA
Creator Address	0x978A71492D0658c69Caf1aF01cCfC012Bb124d34
Current Owner Address	0x978A71492D0658c69Caf1aF01cCfC012Bb124d34
Contract Explorer	https://bscscan.com/token/0x8e84d99561b76cbc46b21d40e8dd062976f28cda#code
Compiler Version	v0.8.17+commit.8df45f5f
License	MIT
Optimisation	Yes with 200 Runs
Total Supply	10,000,000,000 ROYALSHIBA
Decimals	9




Creation/Audit

Contract Deployed	02 Sept 2023
Audit Created	14 Sept 2023
Audit Update	V 0.1

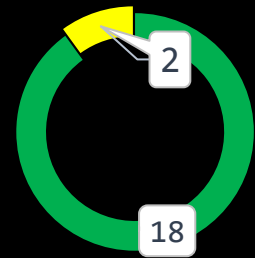
Verified Socials





Website	https://www.royalshiba.com/
Telegram	https://t.me/RoyalShibaGlobal
Twitter (X)	https://twitter.com/RoyalShibabsc

Contract Function Analysis

 Pass
  Attention Item
  Risky Item

■ Pass
 ■ Attention
 ■ Risk

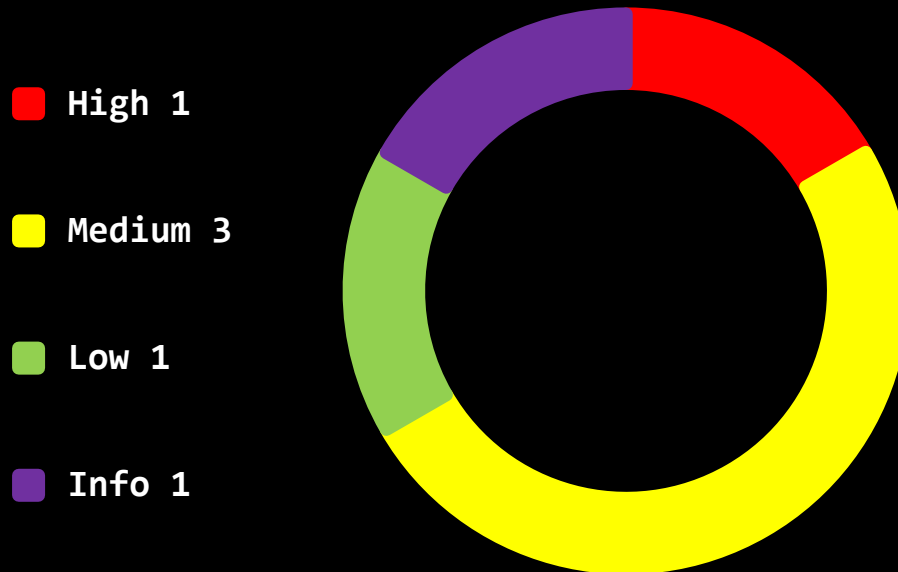



Contract Verified		The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership		Current Owner: 0x978A71492D0658c69Caf1aF01cCfC012Bb124d34
Buy Tax	10 %	Shows the taxes for purchase transactions. Above 10% may be considered a high tax rate. More than 50% tax rate means may not be tradable.
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.
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		No Liquidity Lock found! Token is in Presale phase!
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. The proxy contract means contract owner can modify the function of the token and possibly effect the price. The Owner is not the creator but the creator may have authorisation to change functions.
Mint Function		No mint function found. 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 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 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 authorised hidden owner found.</p> <p>For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned. Fake renounce.</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>


Contract Security


Total Findings: 7



 **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: (1)

- Incorrect Access control

Medium severity issues: (3)

- Approve front running attack
- Unchecked Transfer
- Authorization through tx.origin

Low severity issues: (1)

- Missing Events

Informational severity issues: (1)

- Hard Coded Address

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.

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	medium	low	low
SWC-114	Transaction Order Dependence	Passed	Passed	Passed
SWC-115	Authorization through tx.origin	low	medium	medium
SWC-116	Block values as a proxy for time	Passed	Passed	Passed
SWC-117	Signature Malleability	Passed	Passed	Passed
SWC-118	Incorrect Constructor Name	Passed	Passed	Passed
SWC-119	Shadowing State Variables	Passed	Passed	Passed
SWC-120	Weak Sources of Randomness from Chain Attributes	Passed	Passed	Passed

SWC-121	Missing Protection against Signature Replay Attacks	Passed	Passed	Passed
SWC-122	Lack of Proper Signature Verification	Passed	Passed	Passed
SWC-123	Requirement Violation	Passed	Passed	Passed
SWC-124	Write to Arbitrary Storage Location	Passed	Passed	Passed
SWC-125	Incorrect Inheritance Order	Passed	Passed	Passed
SWC-126	Insufficient Gas Griefing	Passed	Passed	Passed
SWC-127	Arbitrary Jump with Function Type Variable	Passed	Passed	Passed
SWC-128	DoS With Block Gas Limit	Passed	Passed	Passed
SWC-129	Typographical Error	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	Low	Passed	Passed
SWC-136	Unencrypted Private Data On-Chain	Passed	Passed	Passed

Detected High and Medium Severity Vulnerability Description

Incorrect access control (3 Items)

Item: 1	Location:	Line 819-864	Severity:	 High
---------	-----------	--------------	-----------	--

Function	<p>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.</p> <p>The contract DividendTracker is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function process is missing the modifier onlyOwner.</p>
Remediation	<p>It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same</p>

```

819  function process(uint256 gas) public returns (uint256, uint256, uint256) {
820      uint256 numberOfTokenHolders = tokenHoldersMap.keys.length;
821
822      if(numberOfTokenHolders == 0) {
823          return (0, 0, lastProcessedIndex);
824      }
825
826      uint256 _lastProcessedIndex = lastProcessedIndex;
827
828      uint256 gasUsed = 0;
829
830      uint256 gasLeft = gasleft();
831
832      uint256 iterations = 0;
833      uint256 claims = 0;
834
835      while(gasUsed < gas && iterations < numberOfTokenHolders) {
836          _lastProcessedIndex++;
837
838          if(_lastProcessedIndex >= tokenHoldersMap.keys.length) {
839              _lastProcessedIndex = 0;
840          }
841
842          address account = tokenHoldersMap.keys[_lastProcessedIndex];
843
844          if(canAutoClaim(lastClaimTimes[account])) {
845              if(processAccount payable(account), true) {
846                  claims++;
847              }
848          }
849
850          iterations++;
851
852          uint256 newGasLeft = gasleft();
853
854          if(gasLeft > newGasLeft) {
855              gasUsed = gasUsed.add(gasLeft.sub(newGasLeft));
856          }
857
858          gasLeft = newGasLeft;
859      }
860
861      lastProcessedIndex = _lastProcessedIndex;
862
863      return (iterations, claims, lastProcessedIndex);
864  }
865

```

Item: 2	Location: Line 1210-1213	Severity: ■ High
---------	--------------------------	---

Function	<p>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.</p> <p>The contract ROYALSHIBA is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function processDividendTracker is missing the modifier onlyOwner.</p>
Remediation	<p>It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same</p>

```

1210     function processDividendTracker(uint256 gas!) external {
1211         (uint256 iterations, uint256 claims, uint256 lastProcessedIndex) = dividendTracker.process(gas!);
1212         emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, false, gas!, tx.origin);
1213     }
1214 
```

Item: 3	Location:	Line 1215-1217	Severity: ■ High
---------	-----------	----------------	---

Function	<p>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.</p> <p>The contract ROYALSHIBA is importing an access control library @openzeppelin/contracts/access/Ownable.sol but the function claim is missing the modifier onlyOwner.</p>
Remedation	<p>It is recommended to go through the contract and observe the functions that are lacking an access control modifier. If they contain sensitive administrative actions, it is advised to add a suitable modifier to the same</p>

```

1215     function claim() external {
1216         dividendTracker.processAccount(payable(msg.sender), false);
1217     }
1218
  
```

⚠️ Unchecked Transfer (4 Items)

Item: 1	Location:	Line 1030	Severity:	Medium
Item: 2	Location:	Line 1049	Severity:	Medium
Item: 3	Location:	Line 1109	Severity:	Medium
Item: 4	Location:	Line 1112	Severity:	Medium

Function	Some tokens do not revert the transaction when the transfer or transferFrom fails and returns False. Hence we must check the return value after calling the transfer or transferFrom function.
Remediation	Use OpenZeppelin SafeERC20's safetransfer and safetransferFrom functions.

```

1029     if(amount! == 0) {
1030         super._transfer(from!, to!, 0);
1031         return;
  
```

```

1048         burnTokens = contractTokenBalance * (burnFeeOnBuy + burnFeeOnSell) / 100;
1049         super._transfer(address(this), DEAD, burnTokens);
1050     }
  
```

```

1108
1109         super._transfer(from!, address(this), fees);
1110     }
  
```

```

1111
1112         super._transfer(from!, to!, amount!);
1113     }
  
```

⚠️ Authorization through tx.origin (2 Items)

Item: 1	Location:	Line 1121	Severity:	Medium
Item: 2	Location:	Line 1212	Severity:	Medium

Function	In Solidity, tx.origin is a global variable that returns the address of the account that sent the transaction. Using the variable for authorization could make a contract vulnerable. For example, if an authorized account calls a malicious contract which triggers it to call the vulnerable contract that passes an authorization check since tx.origin returns the original sender of the transaction which in this case is the authorized account.
Remediation	tx.origin should not be used for authorization in smart contracts. It does have some legitimate use cases, for example, To prevent external contracts from calling the current contract, you can implement a require of the form require(tx.origin == msg.sender). This prevents intermediate contracts from calling the current contract, thus limiting the contract to regular codeless addresses.


```

1119
1120   try dividendTracker.process(gas) returns (uint256 iterations, uint256 claims, uint256 lastProcessedIndex) {
1121       emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, true, gas, tx.origin);
1122   }
  
```

```

1210   function processDividendTracker(uint256 gas) external {
1211       (uint256 iterations, uint256 claims, uint256 lastProcessedIndex) = dividendTracker.process(gas);
1212       emit ProcessedDividendTracker(iterations, claims, lastProcessedIndex, false, gas, tx.origin);
  
```


Approve front-running attack (2 Item)

Item: 1	Location:	Line 468-471	Severity:	 Medium
---------	-----------	--------------	-----------	--

Function	<p>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.</p> <p>The function approve can be front-run by abusing the _approve function.</p>
Remediation	<p>Only use the approve function of the ERC/BEP standard to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved).</p> <p>Token owner just needs to make sure that the first transaction actually changed allowance from N to 0, i.e., that the spender didn't manage to transfer some of N allowed tokens before the first transaction was mined. Such checking is possible using advanced blockchain explorers such as [Etherscan.io](https://etherscan.io/)</p> <p>Another way to mitigate the threat is to approve token transfers only to smart contracts with verified source code that does not contain logic for performing attacks like described above, and to accounts owned by the people you may trust.</p>

```

468   ttrace | func5ig
469   function approve(address spender!, uint256 amount!) public virtual override returns (bool) {
470       _approve(msgSender(), spender!, amount!);
471       return true;
472   }

```

Item: 2	Location:	Line 473-481	Severity:	Medium
---------	-----------	--------------	-----------	---

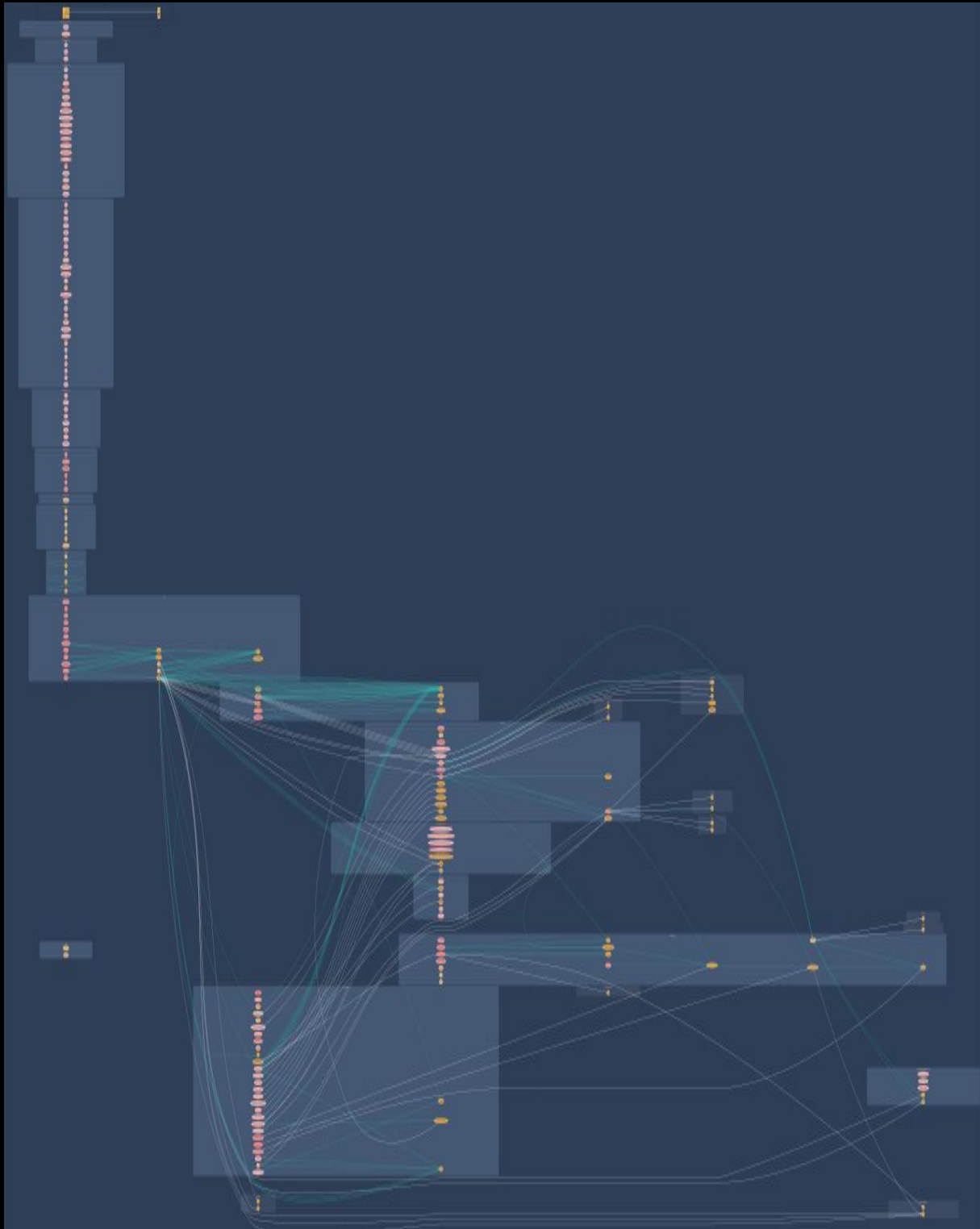
Function	<p>The transferFrom() method overrides current allowance regardless of whether the spender already used it or not, so there is no way to increase or decrease allowance by a certain value atomically unless the token owner is a smart contract, not an account.</p> <p>This can be abused by a token 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 transferFrom can be front-run by abusing the <code>_approve</code> function.</p>
Remediation	<p>Only use the approve function of the ERC/BEP standard to change the allowed amount to 0 or from 0 (wait till transaction is mined and approved).</p> <p>Token owner just needs to make sure that the first transaction actually changed allowance from N to 0, i.e., that the spender didn't manage to transfer some of N allowed tokens before the first transaction was mined. Such checking is possible using advanced blockchain explorers such as [Etherscan.io](https://etherscan.io/)</p> <p>Another way to mitigate the threat is to approve token transfers only to smart contracts with verified source code that does not contain logic for performing attacks like described above, and to accounts owned by the people you may trust.</p>

```

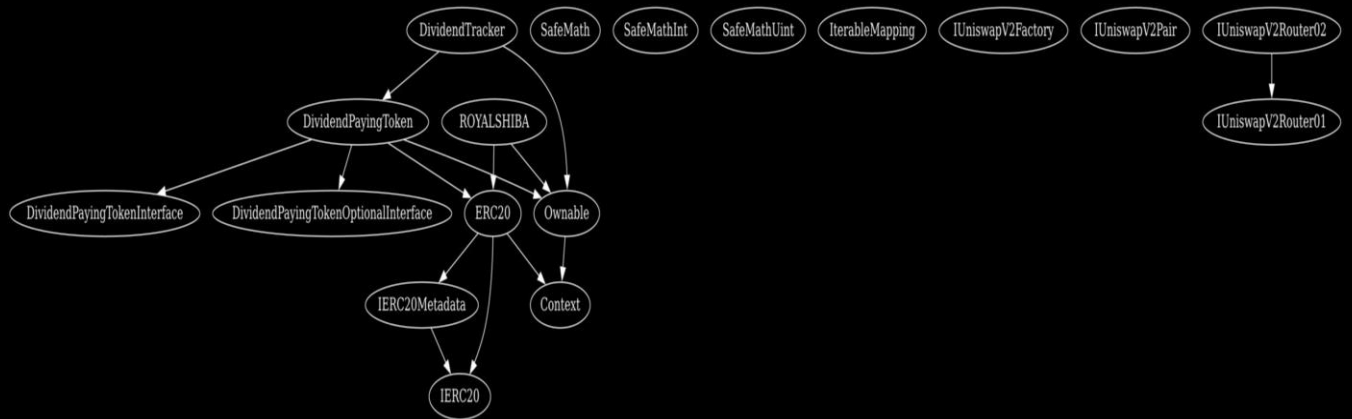
473  function transferFrom(
474      address sender!,
475      address recipient!,
476      uint256 amount!
477  ) public virtual override returns (bool) {
478      _transfer(sender!, recipient!, amount!);
479      _approve(sender!, msgSender(), allowances[sender!][msgSender()].sub(amount!, "ERC20: transfer amount exceeds allowance"));
480      return true;
481  }
482

```

Contract Flow Graph





















Inheritance Graph




























Contract Descriptions




















Contract	Type	Bases		
		Visibility	Mutability	Modifiers
	Function Name			
Context	Implementation			
	_msgSender	Internal 🔒		
	_msgData	Internal 🔒		
Ownable	Implementation	Context		
		Public !	🛑	NO!
	owner	Public !		NO!
	renounceOwnership	Public !	🛑	onlyOwner
	transferOwnership	Public !	🛑	onlyOwner
	_transferOwnership	Internal 🔒	🛑	
SafeMath	Library			
	add	Internal 🔒		
	sub	Internal 🔒		
	sub	Internal 🔒		
	mul	Internal 🔒		
	div	Internal 🔒		
	div	Internal 🔒		
	mod	Internal 🔒		
	mod	Internal 🔒		
SafeMathInt	Library			
	mul	Internal 🔒		
	div	Internal 🔒		
	sub	Internal 🔒		
	add	Internal 🔒		
	abs	Internal 🔒		
	toUint256Safe	Internal 🔒		

SafeMathUint	Library			
	toInt256Safe	Internal 🔒		
IterableMapping	Library			
	get	Public !		NO !
	getIndexOfKey	Public !		NO !
	getKeyAtIndex	Public !		NO !
	size	Public !		NO !
	set	Public !	🔴	NO !
	remove	Public !	🔴	NO !
IUniswapV2Factory	Interface			
	feeTo	External !		NO !
	feeToSetter	External !		NO !
	getPair	External !		NO !
	allPairs	External !		NO !
	allPairsLength	External !		NO !
	createPair	External !	🔴	NO !
	setFeeTo	External !	🔴	NO !
	setFeeToSetter	External !	🔴	NO !
IUniswapV2Pair	Interface			
	name	External !		NO !
	symbol	External !		NO !
	decimals	External !		NO !
	totalSupply	External !		NO !
	balanceOf	External !		NO !
	allowance	External !		NO !
	approve	External !	🔴	NO !
	transfer	External !	🔴	NO !
	transferFrom	External !	🔴	NO !
	DOMAIN_SEPARATOR	External !		NO !
	PERMIT_TYPEHASH	External !		NO !
	nonces	External !		NO !
	permit	External !	🔴	NO !
	MINIMUM_LIQUIDITY	External !		NO !



	factory	External !		NO !
	token0	External !		NO !
	token1	External !		NO !
	getReserves	External !		NO !
	price0CumulativeLast	External !		NO !
	price1CumulativeLast	External !		NO !
	kLast	External !		NO !
	mint	External !		NO !
	burn	External !		NO !
	swap	External !		NO !
	skim	External !		NO !
	sync	External !		NO !
	initialize	External !		NO !
IUniswapV2Route r01	Interface			
	factory	External !		NO !
	WETH	External !		NO !
	addLiquidity	External !		NO !
	addLiquidityETH	External !		NO !
	removeLiquidity	External !		NO !
	removeLiquidityETH	External !		NO !
	removeLiquidityWithPermit	External !		NO !
	removeLiquidityETHWithPermit	External !		NO !
	swapExactTokensForTokens	External !		NO !
	swapTokensForExactTokens	External !		NO !
	swapExactETHForTokens	External !		NO !
	swapTokensForExactETH	External !		NO !
	swapExactTokensForETH	External !		NO !
	swapETHForExactTokens	External !		NO !
	quote	External !		NO !

	getAmountOut	External !		NO !
	getAmountIn	External !		NO !
	getAmountsOut	External !		NO !
	getAmountsIn	External !		NO !
IUniswapV2Router02	Interface	IUniswapV2Router01		
	removeLiquidityETHSupportingFeeOnTransferTokens	External !		NO !
	removeLiquidityETHWithPermitSupportingFeeOnTransferTokens	External !		NO !
	swapExactTokensForTokensSupportingFeeOnTransferTokens	External !		NO !
	swapExactETHForTokensSupportingFeeOnTransferTokens	External !		NO !
	swapExactTokensForETHSupportingFeeOnTransferTokens	External !		NO !
IERC20	Interface			
	totalSupply	External !		NO !
	balanceOf	External !		NO !
	allowance	External !		NO !
	transfer	External !		NO !
	approve	External !		NO !
	transferFrom	External !		NO !
IERC20Metadata	Interface	IERC20		
	name	External !		NO !
	symbol	External !		NO !
	decimals	External !		NO !

ERC20	Implementation	Context, IERC20, IERC20Metadata		
		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 !
	transferFrom	Public !		NO !
	increaseAllowance	Public !		NO !
	decreaseAllowance	Public !		NO !
	_transfer	Internal 		
	_mint	Internal 		
	_burn	Internal 		
	_approve	Internal 		
	_beforeTokenTransfer	Internal 		
DividendPayingTokenInterface	Interface			
	dividendOf	External !		NO !
	withdrawDividend	External !		NO !
DividendPayingTokenOptionalInterface	Interface			
	withdrawableDividendOf	External !		NO !
	withdrawnDividendOf	External !		NO !
	accumulativeDividendOf	External !		NO !

DividendPayingToken	Implementation	ERC20, Ownable, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface		
		Public !		ERC20
	distributeDividends	Public !		onlyOwner
	withdrawDividend	Public !		NO!
	_withdrawDividendOfUser	Internal 		
	dividendOf	Public !		NO!
	withdrawableDividendOf	Public !		NO!
	withdrawnDividendOf	Public !		NO!
	accumulativeDividendOf	Public !		NO!
	_transfer	Internal 		
	_mint	Internal 		
	_burn	Internal 		
	_setBalance	Internal 		
DividendTracker	Implementation	Ownable, DividendPayingToken		
		Public !		DividendPayingToken
	_transfer	Internal 		
	withdrawDividend	Public !		NO!
	updateMinimumTokenBalanceForDividends	External !		onlyOwner
	excludeFromDividends	External !		onlyOwner
	updateClaimWait	External !		onlyOwner
	setLastProcessedIndex	External !		onlyOwner

	getLastProcessedIndex	External !		NO !
	getNumberOfTokenHolders	External !		NO !
	getAccount	Public !		NO !
	getAccountAtIndex	Public !		NO !
	canAutoClaim	Private 🔒		
	setBalance	External !	🔴	onlyOwner
	process	Public !	🔴	NO !
	processAccount	Public !	🔴	onlyOwner
ROYALSHIBA	Implementation	ERC20, Ownable		
		Public !	🟢	ERC20
		External !	🟢	NO !
	setAutomatedMarketMakerPair	External !	🔴	onlyOwner
	claimStuckTokens	External !	🔴	onlyOwner
	isContract	Internal 🔒		
	sendBNB	Internal 🔒	🔴	
	_setAutomatedMarketMakerPair	Private 🔒	🔴	
	excludeFromFees	External !	🔴	onlyOwner
	isExcludedFromFees	Public !		NO !
	_transfer	Internal 🔒	🔴	
	swapAndSendDividends	Private 🔒	🔴	
	setSwapTokensAtAmount	External !	🔴	onlyOwner
	updateClaimWait	External !	🔴	onlyOwner
	getClaimWait	External !		NO !
	getTotalDividendsDistributed	External !		NO !
	withdrawableDividendOf	Public !		NO !
	dividendTokenBalanceOf	Public !		NO !
	totalRewardsEarned	Public !		NO !

	excludeFromDividends	External !		onlyOwner
	getAccountDividendsInfo	External !		NO !
	getAccountDividendsInfoAtIndex	External !		NO !
	processDividendTracker	External !		NO !
	claim	External !		NO !
	claimAddress	External !		onlyOwner
	getLastProcessedIndex	External !		NO !
	setLastProcessedIndex	External !		onlyOwner
	getNumberOfDividendTokenHolders	External !		NO !



Function
can modify
state



Function
is payable

Source:

File Name

SHA-1 Hash

c:\Solidity\royalshiba.sol
1

757f58312cba52ffc3246765f40e2bbce12943
63

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>

