SKELETON ECOSYSTEM

SMART CONTRACT AUDIT

RICHIE RICH

$RICHIE

RICHIE RICH [RICHIE]

BEP 20

`0x7e62bDBe547883feaBB6299Af5FFf041715e5800`

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.

RICHIE RICH BEP20

Overview

Contract Name	RICHIE RICH
Ticker/Simbol	RICHIE
Blockchain	Binance Smart Chain Bep20
Contract Address	0x7e62bDBe547883feaBB6299Af5FFf041715e5800
Creator Address	0x669ACf46050d40c78946768BBe99bb8A6044d1B6
Current Owner Address	Renounced
Contract Explorer	https://bscscan.com/token/0x7e62bdbe547883feabb6299af5fff041715e5800
Compiler Version	v0.8.19+commit.7dd6d404
License	None
Optimisation	Yes with 200 Runs
Total Supply	991,644,290.514097 RICHIE
Decimals	9

Creation/Audit

Contract Deployed	4 Aug 2023
Audit Created	29-Aug-23 20:00:00 UTC
Audit Update	V 0.1

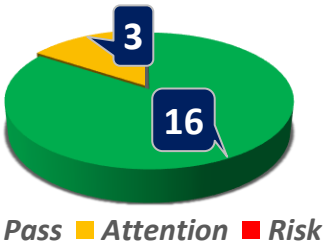
Verified Socials

Website	https://richierich.vip
Telegram	https://t.me/RichieRichVIP
Twitter (X)	https://twitter.com/RichieRichVIP

SKELETON AUDIT REPORT

RICHIE RICH BEP20

Contract Function Analysis

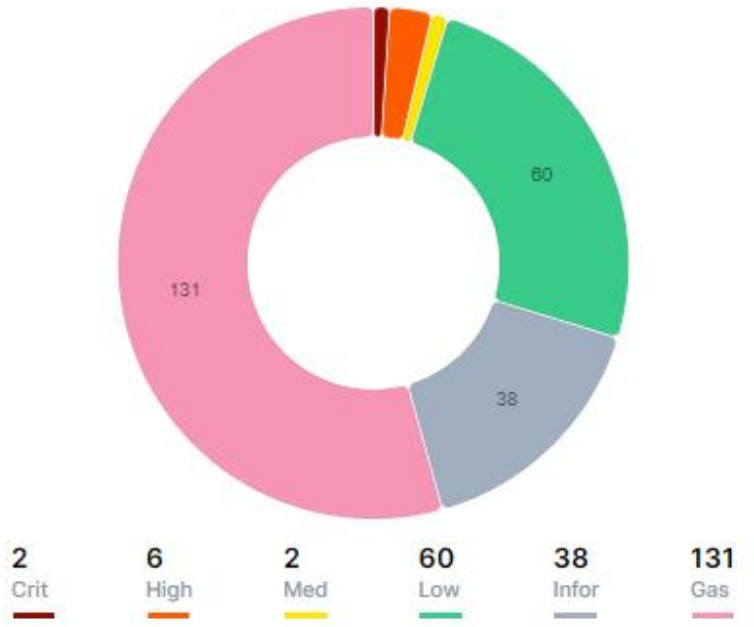


Pass Attention Item Risky Item

Contract Verified	✅	The contract source code is uploaded to blockchain explorer and is open source, so everybody can read it.
Contract Ownership	✅	The ownership of the contract was sent to dead address. With this the owner eliminates he's rights to modify the contract. The owner can not set any of the functions anymore.
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. Contract renounced so tax rate is fixed. ✅
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. Contract renounced so tax rate is fixed. ✅
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	✅	Locked on 29.08.2023: 87% % for 3628 days on DxSale Note! Initial liqudity tokens scanned. For new LP Lockers allways re-check with skeleton scanner on telegram.
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. ⚠ If there is authorised hidden owner, or there is Retrieve Ownership Function, the trading disable function may be used!
Set Fees function	✅	No 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). If contract is renounced this function can't be used. ⚠ If there is authorised hidden owner, or there is Retrieve Ownership Function, the set fees function may be used! ✅ Renounced, this function can not be used.
Proxy Contract	✅	Not a Proxy Contract. The proxy contract means contract owner can modifiy 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. If contract is renounced this function can't be used.

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. ⚠ If contract is renounced this function still can be used as auto self Destruct
Whitelist Function	⚠	Whitelist Function Found. 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) If there is a whitelist, some addresses may not be able to trade normally (honeypot risk). ✅ Renounced, this function can not be used.
Hidden Owner Analysis	✅	No authorised hidden owner found. For contract with a hidden owner, developer can still manipulate the contract even if the ownership has been abandoned. Fake renounce.
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	⚠	Specific Tax Changing Functions found. ✅ Renounced, this function can not be used. 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	⚠	Trading Cooldown Function found. ✅ Renounced, this function can not be modified. 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. ✅ Renounced, this function can not be used. If there is a function for this, the maximum trading amount or maximum position can be modified. Can cause honeypot
Transaction Limiting Function	✅	Transaction Limiter Function Found. ✅ Renounced, this function can not be used. The number of overall token transactions may be limited (honeypot risk)

Contract Safety and Weakness



PUBLIC BURN	2
UNCHECKED TRANSFER	3
APPROVE FRONT-RUNNING ATTACK	2
REENTRANCY	1
DANGEROUS UNARY EXPRESSIONS	1
RETURN VALUE OF LOW LEVEL CALLS	1
USE OF MULTIPLE PRAGMA VERSIONS	3
EMPTY TRY/CATCH BLOCK	1
USE OWNABLE2STEP	3
MISCONFIGURED BEFORETOKENTRAN...	4

LONG NUMBER LITERALS	6
OUTDATED COMPILER VERSION	11
USE OF FLOATING PRAGMA	11
MISSING EVENTS	19
COMPILER VERSION TOO RECENT	1
EVENT BASED REENTRANCY	1
MISSING PAYABLE IN CALL FUNCTION	1
VARIABLES SHOULD BE IMMUTABLE	2
NAME MAPPING PARAMETERS	3
MISSING UNDERSCORE IN NAMING VA...	7

REQUIRE WITH EMPTY MESSAGE	3
MISSING INDEXED KEYWORDS IN EVE...	8
BLOCK VALUES AS A PROXY FOR TIME	7
UNUSED RECEIVE FALLBACK	1
HARD-CODED ADDRESS DETECTED	6
CONSTANT STATE VARIABLES	1
CODE OPTIMIZATION BY USING MAX ...	1
MAKE PUBLIC LIBRARY FUNCTIONS IN...	6
DEFINE CONSTRUCTOR AS PAYABLE	4
MULTIPLICATION/DIVISION BY TWO S...	1

FUNCTION SHOULD BE EXTERNAL	7
SPLITTING REQUIRE STATEMENTS	6
INTERNAL FUNCTIONS NEVER USED	1
FUNCTION SHOULD RETURN STRUCT	4
CHEAPER INEQUALITIES IN IF()	8
GAS OPTIMIZATION FOR STATE VARIA...	1
CHEAPER CONDITIONAL OPERATORS	7
CHEAPER INEQUALITIES IN REQUIRE()	15
LONG REQUIRE/REVERT STRINGS	22
STORAGE VARIABLE CACHING IN MEM...	47

⚠ Public Burn (2 item)

```
16      * @dev Destroys `amount` tokens from the caller.
17      *
18      * See {ERC20-_burn}.
19      */
20     function burn(uint256 amount) public virtual {
21         _burn(_msgSender(), amount);
22     }
23
24     /**
25      * @dev Destroys `amount` tokens from `account`, deducting from the caller's
26      * allowance.
27      *
```

```
33      * `amount`.
34      */
35     function burnFrom(address account, uint256 amount) public virtual {
36         _spendAllowance(account, _msgSender(), amount);
37         _burn(account, amount);
38     }
39 }
40
```

Function	Severity	Remedation
The contract was found to be using public or an external burn function. The function was missing access control to prevent another user from burning their tokens. Also, the burn function was found to be using a different address than msg.sender.	⚙ Severity : Critical	Consider adding access control modifiers to the burn function to prevent another user from burning their tokens. The burn function should use msg.sender in the _from argument.

⚠ Unchecked Transfer (3 Item)

```
357             }
358
359             if (fees > 0) {
360                 super._transfer(from, address(this), fees);
361             }
362         }
363
364         super._transfer(from, to, amount);
365
366         dividendTracker.setBalance(from, balanceOf(from));
```

```
361             }
362         }
363
364         super._transfer(from, to, amount);
365
366         dividendTracker.setBalance(from, balanceOf(from));
367         dividendTracker.setBalance(to, balanceOf(to));
368
```

```
71    function distributeDividends(uint256 amount) public {
72      require(totalSupply() > 0);
73
74      uint256 balBefore = IERC20(rewardToken).balanceOf(address(this));
75      IERC20(rewardToken).transferFrom(msg.sender, address(this), amount);
76      uint256 received = IERC20(rewardToken).balanceOf(address(this)) - balBefore;
77
78      if (received > 0) {
79        magnifiedDividendPerShare = magnifiedDividendPerShare + (received * magnitude / totalSupply());
80
```

Function	Severity	Remedation
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.	Severity : High	Use OpenZeppelin SafeERC20's safetransfer and safetransferFrom functions.

⚠ Approve Front Running Attack (2 Items)

```
134         * - `spender` cannot be the zero address.
135         */
136        function approve(address spender, uint256 amount) public virtual override returns (bool) {
137            address owner = _msgSender();
138            _approve(owner, spender, amount);
139            return true;
140        }
141
142        /**
143         * @dev See {IERC20-transferFrom}.
144         *
145         * Emits an {Approval} event indicating the updated allowance. This is not
```

```
321         *
322         * Might emit an {Approval} event.
323         */
324        function _spendAllowance(address owner, address spender, uint256 amount) internal virtual {
325            uint256 currentAllowance = allowance(owner, spender);
326            if (currentAllowance != type(uint256).max) {
327                require(currentAllowance >= amount, "ERC20: insufficient allowance");
328                unchecked {
329                    _approve(owner, spender, currentAllowance - amount);
330                }
331            }
332        }
333
334        /**
335         * @dev Hook that is called before any transfer of tokens. This includes
```

Function	Severity	Remedation
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	Severity : High	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). 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/) Another way to mitigate the threat is to approve token transfers only to smart contracts with verified source

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.		code that does not contain logic for performing attacks like described above, and to accounts owned by the people you may trust.

⚠ Reentrancy (1 item)

```
67    constructor(address _rewardToken, string memory _name, string memory _symbol) ERC20(_name, _symbol) {
68      rewardToken = _rewardToken;
69    }
70
71    function distributeDividends(uint256 amount) public {
72      require(totalSupply() > 0);
73
74      uint256 balBefore = IERC20(rewardToken).balanceOf(address(this));
75      IERC20(rewardToken).transferFrom(msg.sender, address(this), amount);
76      uint256 received = IERC20(rewardToken).balanceOf(address(this)) - balBefore;
77
78      if (received > 0) {
79        magnifiedDividendPerShare = magnifiedDividendPerShare + (received * magnitude / totalSupply());
80
81        emit DividendsDistributed(msg.sender, received);
82
83        totalDividendsDistributed = totalDividendsDistributed + received;
84      }
85    }
86
87    function _withdrawDividend(address account) internal returns(uint256) {
88      uint256 withdrawableDividend = withdrawableDividendOf(account);
89
```

Function	Severity	Remedation
In a Re-entrancy attack, a malicious contract calls back into the calling contract before the first invocation of the function is finished. This may cause the different invocations of the function to interact in undesirable ways, especially in cases where the function is updating state variables after the external calls. This may lead to loss of funds, improper value updates, token loss, etc.	⚙ Severity : High	It is recommended to add a [Re-entrancy Guard] to the functions making external calls. The functions should use a Checks-Effects-Interactions pattern. The external calls should be executed at the end of the function and all the state-changing must happen before the call.

⚠ Return Value of Low level call (1 item)

```
300                 uint256 coinsReceived = address(this).balance;
301
302                 uint256 marketingPortion = coinsReceived * _marketingPending / token2Swap;
303                 if (marketingPortion > 0) {
304                     (success,) = payable(address(marketingAddress)).call{value: marketingPortion}("");
305                     require(success, "TaxesDefaultRouterWalletCoin: Fee transfer error");
306                     emit marketingFeeSent(marketingAddress, marketingPortion);
307                 }
308                 _marketingPending = 0;
309
```

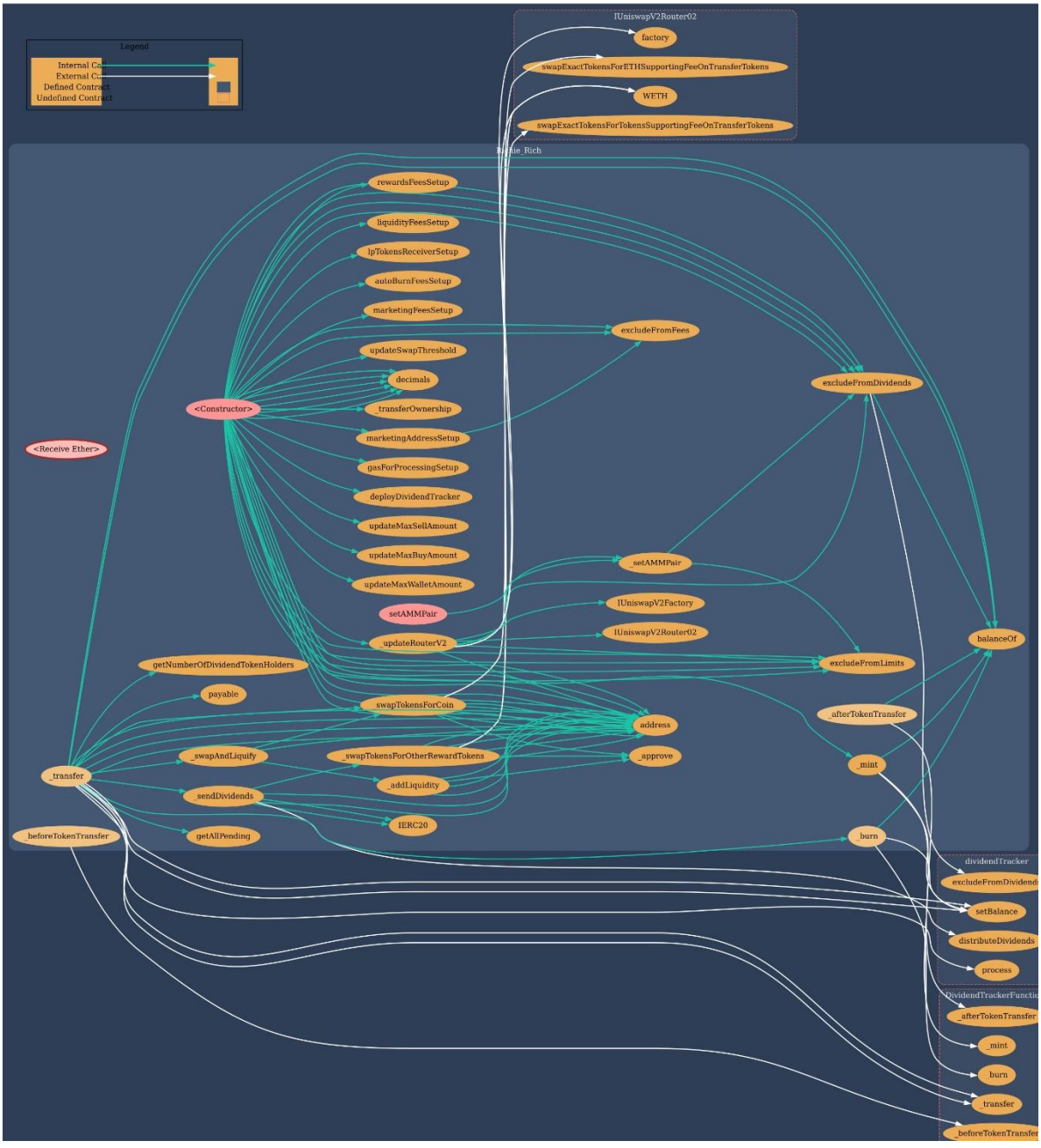
Function	Severity	Remedation
The functions do not check the return value of low-level calls. This can lock Ether in the contract if the call fails or may compromise the contract if the ownership is being changed. The following calls were detected without return value validations - call	Severity : Medium	Ensure return value is checked using conditional statements for low-level calls. We should also ensure that we log failed calls using events.

⚠ Dangerous Unary Expression (1 item)

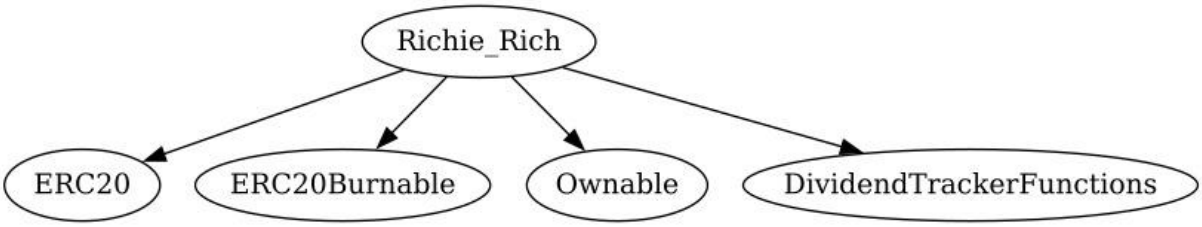
```
262   {
263     account = _account;
264     index = tokenHoldersMap.getIndexOfKey(account);
265     iterationsUntilProcessed = -1;
266
267     if (index >= 0) {
268       if (uint256(index) > lastProcessedIndex) {
269         iterationsUntilProcessed = index - int256(lastProcessedIndex);
270       } else {
```

Function	Severity	Remedation
Mathematical operations in the smart contracts forms the base for all the arithmetic logic in the code. Care should be taken when implementing these because they may control critical function logic, tokens, ether, etc. The contract was found to be incorrectly implementing arithmetic expressions assuming the contract wanted to add the value of a to b and store it back in a. Correct usage: a += b; Incorrect usage: a =+b;	Severity : Medium	Developers should exercise caution when writing these expressions because a simple mistake may cause loss of funds or compromise of the contract. Make sure that the expressions used are in the correct format, i.e., a += b; and not a =+b;.

Contract Flow Graph



Inheritance Graph



Contract Descriptions

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
Richie_Rich	Implementation	ERC20, ERC20Burnable, Ownable, DividendTrackerFunctions		
		Public		ERC20
		External		NO
	decimals	Public		NO
	_swapTokensForCoin	Private		
	updateSwapThreshold	Public		onlyOwner
	getAllPending	Public		NO
	marketingAddressSetup	Public		onlyOwner
	marketingFeesSetup	Public		onlyOwner
	autoBurnFeesSetup	Public		onlyOwner
	_swapAndLiquify	Private		
	_addLiquidity	Private		
	lpTokensReceiverSetup	Public		onlyOwner
	liquidityFeesSetup	Public		onlyOwner
	_swapTokensForOtherRewardTokens	Private		
	_sendDividends	Private		
	excludeFromDividends	Public		onlyOwner
	rewardsFeesSetup	Public		onlyOwner
	_burn	Internal		
	_mint	Internal		

	excludeFromFees	Public		onlyOwner
	_transfer	Internal		
	_updateRouterV2	Private		
	setAMMPair	Public		onlyOwner
	_setAMMPair	Private		
	excludeFromLimits	Public		onlyOwner
	updateMaxWalletAmount	Public		onlyOwner
	updateMaxBuyAmount	Public		onlyOwner
	updateMaxSellAmount	Public		onlyOwner
	_beforeTokenTransfer	Internal		
	_afterTokenTransfer	Internal		

Function can modify state

Function is payable

Source:

File Name SHA-1 Hash

c:\Solidity\richierich.sol f7a4996563f9cae3ef03072c3579ef4fab1f726d

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