RugFreeCoins Audit

Kira Doge Audit

Smart Contract Security Audit

November 16, 2021

Contents

Audit details

Audited project

Kira Doge Token

Contract Address

0x07236AA0A4886Ae2c097C86cc29954F78165B327

Client contact

Kira Doge Team

Blockchain

Binance smart chain

Project website

https://kiradoge.com/

Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice as at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn't say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

DISCLAIMER: By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and Rugfreecoins and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Rugfreecoins) owe no duty of care towards you or any other person, nor does Rugfreecoins make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties or other terms of any kind except as set out in this disclaimer, and Rugfreecoins hereby excludes all representations, warranties, conditions and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, Rugfreecoins hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Rugfreecoins, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed.

Background

Rugfreecoins was commissioned by Kira Doge to perform an audit of the smart contract.

https://bscscan.com/token/0x07236AA0A4886Ae2c097C86cc29954F78165B327

The focus of this audit is to verify that the smart contract is secure, resilient and working according to the specifications.

The information in this report should be used to understand the risk exposure of the smart contract, project feasibility, long term sustainability and as a guide to improve the security posture of the smart contract by remediating the issues that were identified.

About the project

Kira Doge is a token built on the Binance Smart Chain that is with an innovative investment use case the main purpose of which is to seek out constant revenue sources, which in turn, powers reward combined with lottery system and auto burn. Each transaction, purchase incurs 12% fee, and sale incur a 15% fee.

Features

- The **BNB rewards** will be distributed among every holder proportional to how many tokens each individual holds in values of **1% when buying and 8% when selling**.

- **The lottery fee is 6% when buying and 2% when selling** is what allows Kira Doge to become the most commonly known and recognized lottery token in the crypto sphere. In order for a cryptocurrency to grow and gain traction, especially in the Altcoin market, it must have a 'use-case', which only usually comes with the promise of a better future. The Kira Doge team is motivated by the idea that the coin will have a use-case from day one! The gambling industry has been around for centuries, and there will be an ever-growing crowd of "gamblers" and players in the crypto sphere, as cryptocurrencies slowly become the staple in terms of money transactions around the world. To support this transition, Kira Doge wishes to establish itself as the most competitive and well-known lottery token in the industry.

With Kira Doge, the chances of winning are relative to how many tokens you hold, which means that all holders are incentivized to buy more tokens in the long term if they wish to increase their chances of winning the lottery.

- The **sustainability fee of 4% when buying and 3% when selling for marketing and dev** is what allows Kira Doge to hold the aforementioned promise. Tokens will be swapped into BNB and will be sent to a marketing wallet per transaction. This way, Kira Doge will have enough funds to promote the coin and spend for future development without selling tokens as the traditional way.

- The additional component included under the sustainability section is a **liquidity fee of 1% from buying and selling**, which is a redistribution mechanism that ensures the trading pool always has sufficient liquidity.

- Kira Doge has the burn strategy that **1% fee in each transaction when selling** is getting charged that benefits and rewards those who invest long-term. This feature slowly reduces supply making each Kira Doge more and more valuable.

Tokenomics

12% fee when buying

- 1% of trade goes to holders' pockets in BNB tokens.
- 6% of trade goes to the lottery pool.
- 4% of trade goes to the marketing and Dev wallets.
- 1% of trade goes to the liquidity pool.

15% fee when selling

- 8% of trade goes to holders’ pockets in BNB tokens.
- 2% of trade goes to the lottery pool.
- 3% of trade goes to the marketing and Dev wallets.
- 1% of trade goes to the liquidity pool.
- 1% of trade goes to the burn wallet.

Roadmap

November

- Contract creation and security testing
- Website launch
- Kyc passed
- Audit completed
- Presale on pinksale
- Launch on pancakeswap lp lock 1year
- First cex listing
- 3 trading pairs
- Lottery app launch
- Influencers marketing
- Expand kiradoge partnerships
- Listing on cgk
- Blockfolio listing
- Cmc listing
- Staking option
- Banner and influencers marketing
- Nft examples and launch date

December

- 2nd cex listing
- Poocoing ads
- Roadmap update

Target market and the concept

Target market

- Anyone who's interested in the Crypto space with long-term investment plans.
- Anyone who's ready to earn a passive income in BNB by holding tokens.
- Anyone who's interested in trading tokens.
- Anyone who is ready to hold and be eligible to win in the daily lottery
- Anyone who is ready to hold a large portion of tokens and be eligible to get a high chance of winning in the weekly lottery.
- Anyone who's interested in collecting NFTs or trading NFTs.
- Anyone who's interested in taking part with the future plans of the Kira Doge token.
- Anyone who's interested in making financial transactions with any other party using BNB or Kira doge as the currency.

Core concept

The Kira Doge reward system

1% of each transaction when buying and 8% when selling get converted to BNB and is split amongst all holders. Holders will be eligible to receive tokens everyone hour and rewards are proportional to how many tokens each individual holds.

Sustainable mechanism

The **sustainability fee of 3% when buying and 4% when selling for marketing and dev** is what allows Kira Doge to promote the token and use funds to further the development of the platform. Tokens will be swapped into BNB and will be sent to a marketing wallet per transaction. This way, Kira Doge will have access to the funds without selling tokens as the traditional way, which will enable them to consume funds without hurting the project.

Lottery pool & lottery platform

The concept encourages,

- Investors hold the token for a long time, which makes them believe in the project and keep the hopes high of expecting to win a huge prize at once.
- Investors buy more and more since the chance of winning is higher.
- The concept is revolutionary and certainly can get the attraction of new investors as the project progresses along.
- Project market price and market cap can keep stable if everything goes according to the plan since keeping tokens will seem more profitable than selling.

Weekly lottery drawing

KIRADoge lottery is with a strong use case specifically targeting the gambling industry aiming for any long-term believers and holders to give a chance to be eligible for the weekly lottery and win. The most unique core part of the KIRADoge is that the chances of winning are relative to how many tokens investors hold, which means that all holders are incentivized to buy more tokens in the long term if they wish to increase their chances of winning the lottery.

How chances of winning are calculated

Chances of winning will be calculated in indirect proportion to how many tokens each holder has. This means that having more tokens does increase your chances of winning, but not in a linear fashion. Instead, a logarithmic function will be used to convert the proportion of holdings that each investor has and calculate their chances of winning accordingly. This will lower the discrepancy in the probability of winning between a whale and a small investor while keeping our largest investors at an advantage.

No. of tokens	% chance of winning	Log transformation	% of winning (log transformation)
15	0.50	1.17609	0.36784
07	0.23	0.84510	0.26432
05	0.17	0.69897	0.21861
03	0.10	0.477120	0.14923
Total		**3.19728**	**1**

Lottery Drawing Dapp

The lottery platform will be visible in an interface, where all contract holders are visible with their wallet IDs and the number of tokens they hold. Holders can connect wallets and check the probability of winning against the rest of the holders.

Winners will be chosen on a random draw, live every Friday on video chat. The winners will be populated on the web with wallet IDs and the amounts they won.

Conditions to be eligible for the lottery

- The holder should hold at least 1 billion tokens.
- The holder should be holding tokens for 5 days or more.
- Winners from one lottery will not be eligible for the upcoming lottery and be eligible for the next. (7 days of limitation for participating in the lottery.)
- Team, marketing, and dev wallets will be excluded from winning the lottery.

There will be a threshold where the holders should be eligible to be part of the lottery. Something like 100,000 tokens minimum. And in daily/ once in 3 days video chats, you will run the lottery and select 15 lucky winners. The winners will be selected based on the tokens they hold in a logarithmic algorithm. 1st winner will get a large amount from the daily lottery.

Lottery prize distribution

The token will be distributed manually among the winners. The converted BNB from transactions will be sent to one wallet address, which will be known to the public. The wallet address that has been coded into the KIRADoge contract is the following:

0XD6C87830955FF9EAD5155D85939BCB516389FC14

Everyone will be able to monitor this wallet address and be reassured those tokens are distributed to the randomly chosen winners.

How prize distribution happens

- 15 winners per one lottery draw and the 1st winner will get a large amount from the lottery and the rest in that order.
- Prizes will be distributed right after the lottery drawing.
- 10% tax will be charged from the winning prize and will be allocated for marketing.

The liquidity fee of 1%, which is a redistribution mechanism that ensures the trading pool always has sufficient liquidity.

Kira Doge has the burn strategy that **1% fee in each transaction when selling** is getting charged that benefits and rewards those who invest long-term. This feature slowly reduces supply making each Kira Doge more and more valuable.

Potential to grow with score points

1.	Project efficiency	10/10
2.	Project uniqueness	10/10
3	Information quality	10/10
4	Service quality	10/10
5	System quality	10/10
6	Impact on the community	10/10
7	Impact on the business	10/10
8	Preparing for the future	10/10
Total Points		**10/10**

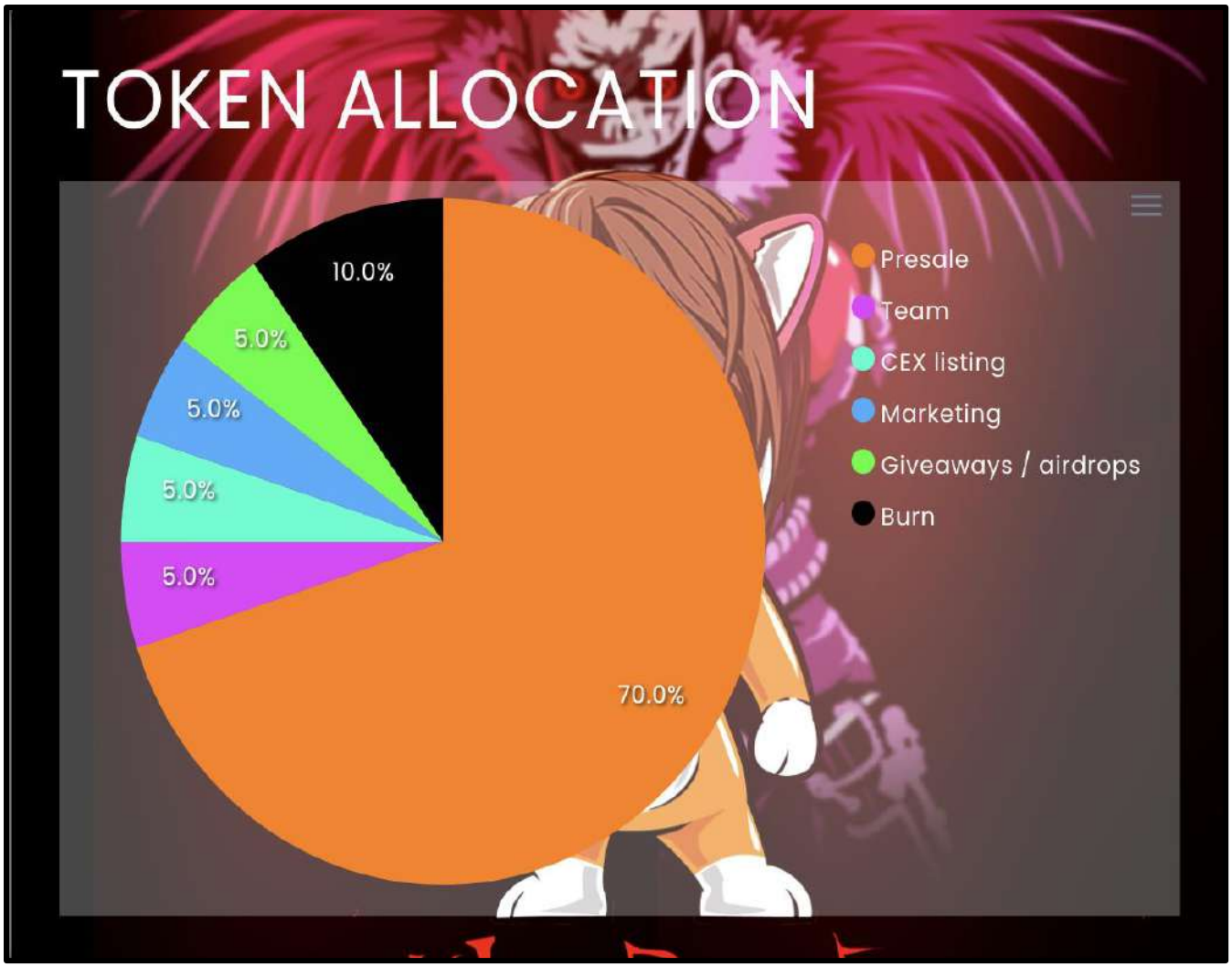
Contract details

Token contract details for 16th November 2021

Contract name	Kiradoge Coin
Contract address	0x07236AA0A4886Ae2c097C86cc29954F78165B327
Token supply	1,000,000,000,000,000
Token ticker	KIRADOGE
Decimals	9
Token holders	5
Transaction count	7
Dev wallet	0x01abb2d65efc6c7120801d50f189d59f57d604db
Dividend tracker	0x0814ef27ee44fdcb7893b8e736606ddd01960a23
Lottery fee receiver	0xd6c87830955ff9ead5155d85939bcb516389fc14
Marketing wallet	0x5cad272a320dab8f83921afe999d66d2f16a99ca
Contract deployer address	0x2840b60e73BaEE5A371cb6C03Cae1AF4311bFb53
Contract's current owner address	x9fe8b1dbe2749c57574f4b1da7068beb54eb044b

Token distribution

Tokens are distributed as follows:



Contract code function details

No	Category	Item	Result
1	Coding conventions	BRC20 Token standards	pass
		compile errors	pass
		Compiler version security	pass
		visibility specifiers	pass
		Gas consumption	pass
		SafeMath features	pass
		Fallback usage	pass
		tx.origin usage	pass
		deprecated items	pass
		Redundant code	pass
		Overriding variables	pass
2	Function call audit	Authorization of function call	pass
		Low level function (call/delegate call) security	pass
		Returned value security	pass
		Selfdestruct function security	pass
3	Business security	Access control of owners	pass
		Business logics	pass
		Business implementations	pass
4	Integer overflow/underflow		pass
5	Reentrancy		pass
6	Exceptional reachable state		pass
7	Transaction ordering dependence		pass
8	Block properties dependence		pass
9	Pseudo random number generator (PRNG)		pass
10	DoS (Denial of Service)		pass
11	Token vesting implementation		pass
12	Fake deposit		pass
13	Event security		pass

Contract description table

Below table represents the summary of the contracts and methods in the token contract. We scanned the whole contract and listed down all the Interfaces, functions and implementations with its visibility and mutability.

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!
IERC20Metadata	**Interface**	**IERC20**		
L	name	External !		NO!
L	symbol	External !		NO!
L	decimals	External !		NO!
Context	**Implementation**			
L	_msgSender	Internal 🔒		

L	_msgData	Internal 🔒		
SafeMath	**Library**			
L	add	Internal 🔒		
L	sub	Internal 🔒		
L	sub	Internal 🔒		
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	div	Internal 🔒		
L	mod	Internal 🔒		
L	mod	Internal 🔒		
ERC20	**Implementation**	**Context, IERC20, IERC20Metadata**		
L		Public ❗	🛑	NO❗
L	name	Public ❗		NO❗
L	symbol	Public ❗		NO❗
L	decimals	Public ❗		NO❗
L	totalSupply	Public ❗		NO❗
L	balanceOf	Public ❗		NO❗

L	transfer	Public ❗	🛑	NO❗
L	allowance	Public ❗		NO❗
L	approve	Public ❗	🛑	NO❗
L	transferFrom	Public ❗	🛑	NO❗
L	increaseAllowance	Public ❗	🛑	NO❗
L	decreaseAllowance	Public ❗	🛑	NO❗
L	_transfer	Internal 🔒	🛑	
L	_mint	Internal 🔒	🛑	
L	_burn	Internal 🔒	🛑	
L	_approve	Internal 🔒	🛑	
L	_beforeTokenTransfer	Internal 🔒	🛑	
SafeMathUint	**Library**			
L	toInt256Safe	Internal 🔒		
SafeMathInt	**Library**			
L	mul	Internal 🔒		
L	div	Internal 🔒		
L	sub	Internal 🔒		

L	add	Internal 🔒		
L	abs	Internal 🔒		
L	toUint256Safe	Internal 🔒		
DividendPayingTokenInterface	**Interface**			
L	dividendOf	External ❗		NO❗
L	distributeDividends	External ❗	💵	NO❗
L	withdrawDividend	External ❗	🛑	NO❗
DividendPayingTokenOptionalInterface	**Interface**			
L	withdrawableDividendOf	External ❗		NO❗
L	withdrawnDividendOf	External ❗		NO❗
L	accumulativeDividendOf	External ❗		NO❗
DividendPayingToken	**Implementation**	**ERC20, DividendPayingTokenInterface, DividendPayingTokenOptionalInterface**		
L		Public ❗	🛑	ERC20
L		External ❗	💵	NO❗
L	distributeDividends	Public ❗	💵	NO❗

L	withdrawDividend	Public ❗	🛑	NO❗
L	_withdrawDividendOfUser	Internal 🔒	🛑	
L	dividendOf	Public ❗		NO❗
L	withdrawableDividendOf	Public ❗		NO❗
L	withdrawnDividendOf	Public ❗		NO❗
L	accumulativeDividendOf	Public ❗		NO❗
L	_transfer	Internal 🔒	🛑	
L	_mint	Internal 🔒	🛑	
L	_burn	Internal 🔒	🛑	
L	_setBalance	Internal 🔒	🛑	
IterableMapping	**Library**			
L	get	Public ❗		NO❗
L	getIndexOfKey	Public ❗		NO❗
L	getKeyAtIndex	Public ❗		NO❗
L	size	Public ❗		NO❗
L	set	Public ❗	🛑	NO❗
L	remove	Public ❗	🛑	NO❗

Ownable	Implementation	Context		
L		Public ❗	◉	NO❗
L	owner	Public ❗		NO❗
L	renounceOwnership	Public ❗	◉	onlyOwner
L	transferOwnership	Public ❗	◉	onlyOwner
IUniswapV2Pair	**Interface**			
L	name	External ❗		NO❗
L	symbol	External ❗		NO❗
L	decimals	External ❗		NO❗
L	totalSupply	External ❗		NO❗
L	balanceOf	External ❗		NO❗
L	allowance	External ❗		NO❗
L	approve	External ❗	◉	NO❗
L	transfer	External ❗	◉	NO❗
L	transferFrom	External ❗	◉	NO❗
L	DOMAIN_SEPARATOR	External ❗		NO❗
L	PERMIT_TYPEHASH	External ❗		NO❗

L	nonces	External ❗		NO❗
L	permit	External ❗	●	NO❗
L	MINIMUM_LIQUIDITY	External ❗		NO❗
L	factory	External ❗		NO❗
L	token0	External ❗		NO❗
L	token1	External ❗		NO❗
L	getReserves	External ❗		NO❗
L	price0CumulativeLast	External ❗		NO❗
L	price1CumulativeLast	External ❗		NO❗
L	kLast	External ❗		NO❗
L	mint	External ❗	●	NO❗
L	burn	External ❗	●	NO❗
L	swap	External ❗	●	NO❗
L	skim	External ❗	●	NO❗
L	sync	External ❗	●	NO❗
L	initialize	External ❗	●	NO❗
IUniswapV2Factory	**Interface**			

L	feeTo	External ❗		NO❗
L	feeToSetter	External ❗		NO❗
L	getPair	External ❗		NO❗
L	allPairs	External ❗		NO❗
L	allPairsLength	External ❗		NO❗
L	createPair	External ❗	🛑	NO❗
L	setFeeTo	External ❗	🛑	NO❗
L	setFeeToSetter	External ❗	🛑	NO❗
IUniswapV2Router01	**Interface**			
L	factory	External ❗		NO❗
L	WETH	External ❗		NO❗
L	addLiquidity	External ❗	🛑	NO❗
L	addLiquidityETH	External ❗	💵	NO❗
L	removeLiquidity	External ❗	🛑	NO❗
L	removeLiquidityETH	External ❗	🛑	NO❗
L	removeLiquidityWithPermit	External ❗	🛑	NO❗
L	removeLiquidityETHWithPermit	External ❗	🛑	NO❗

L	swapExactTokens ForTokens	External ❗	🛑	NO❗
L	swapTokensForEx actTokens	External ❗	🛑	NO❗
L	swapExactETHFor Tokens	External ❗	💶	NO❗
L	swapTokensForEx actETH	External ❗	🛑	NO❗
L	swapExactTokens ForETH	External ❗	🛑	NO❗
L	swapETHForExact Tokens	External ❗	💶	NO❗
L	quote	External ❗		NO❗
L	getAmountOut	External ❗		NO❗
L	getAmountIn	External ❗		NO❗
L	getAmountsOut	External ❗		NO❗
L	getAmountsIn	External ❗		NO❗
IUniswapV2Router02	**Interface**	**IUniswapV2Router01**		
L	removeLiquidityET HSupportingFeeO nTransferTokens	External ❗	🛑	NO❗
L	removeLiquidityET HWithPermitSupp ortingFeeOnTransf erTokens	External ❗	🛑	NO❗
L	swapExactTokens ForTokensSupport ingFeeOnTransfer Tokens	External ❗	🛑	NO❗

L	swapExactETHForTokensSupportingFeeOnTransferTokens	External		NO
L	swapExactTokensForETHSupportingFeeOnTransferTokens	External		NO
KIRADOGE	**Implementation**	**ERC20, Ownable**		
L		Public		ERC20
L		External		NO
L	updateDividendTracker	Public		onlyOwner
L	updateUniswapV2Router	Public		onlyOwner
L	excludeFromFees	Public		onlyOwner
L	excludeMultipleAccountsFromFees	Public		onlyOwner
L	setAutomatedMarketMakerPair	Public		onlyOwner
L	_setAutomatedMarketMakerPair	Private		
L	updatemaxBuyTransactionAmount	Public		onlyOwner
L	updatemaxSellTransactionAmount	Public		onlyOwner
L	updatemaxWalletToken	Public		onlyOwner
L	updateswapTokensAtAmount	Public		onlyOwner
L	updateMarketingWallet	Public		onlyOwner

L	startTrading	Public ❗	◉	onlyOwner
L	liftAllLimits	Public ❗	◉	onlyOwner
L	updateGasForProcessing	Public ❗	◉	onlyOwner
L	updateClaimWait	External ❗	◉	onlyOwner
L	updateDevWallet	External ❗	◉	onlyOwner
L	updateLotteryFeeReceiver	External ❗	◉	onlyOwner
L	updateBuyFees	External ❗	◉	onlyOwner
L	updateSellFees	External ❗	◉	onlyOwner
L	updateSwapPrecentage	External ❗	◉	onlyOwner
L	getClaimWait	External ❗		NO❗
L	getTotalDividendsDistributed	External ❗		NO❗
L	isExcludedFromFees	Public ❗		NO❗
L	withdrawableDividendOf	Public ❗		NO❗
L	dividendTokenBalanceOf	Public ❗		NO❗
L	getAccountDividendsInfo	External ❗		NO❗
L	getAccountDividendsInfoAtIndex	External ❗		NO❗
L	processDividendTracker	External ❗	◉	NO❗
L	claim	External ❗	◉	NO❗

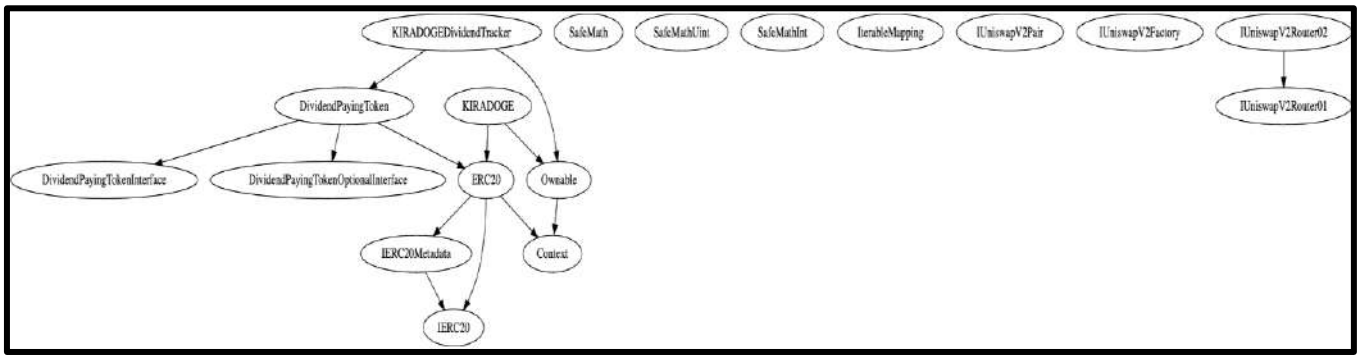
L	getLastProcessedIndex	External !		NO!
L	getNumberOfDividendTokenHolders	External !		NO!
L	getLiquidityIsAdded	Public !		NO!
L	setLiquidityisAdded	External !		onlyOwner
L	_transfer	Internal		
L	swapAndLiquify	Private		
L	swapTokensForEth	Private		
L	addLiquidity	Private		
L	swapAndSendDividends	Private		
KIRADOGEDividendTracker	**Implementation**	**DividendPayingToken, Ownable**		
L		Public !		DividendPayingToken
L	_transfer	Internal		
L	withdrawDividend	Public !		NO!
L	excludeFromDividends	External !		onlyOwner
L	updateClaimWait	External !		onlyOwner
L	getLastProcessedIndex	External !		NO!
L	getNumberOfTokenHolders	External !		NO!

L	getAccount	Public ❗		NO❗
L	getAccountAtIndex	Public ❗		NO❗
L	canAutoClaim	Private 🔐		
L	setBalance	External ❗	🛑	onlyOwner
L	process	Public ❗	🛑	NO❗
L	processAccount	Public ❗	🛑	onlyOwner

Legend

Symbol	Meaning
🛑	**Function can modify state**
💵	**Function is payable**

Inheritance Hierarchy



Security issue checking status

- **High severity issues**
  No high severity issues found.

- **Medium severity issues**
  No medium severity issues found.

- **Low severity issues**
  No low severity issues found.

Owner privileges

- The owner can renounce and transfer ownership.

```
ftrace | funcSig
function renounceOwnership() public virtual onlyOwner {
    emit OwnershipTransferred(_owner, address(0));
    _owner = address(0);
}

/**
 * @dev Transfers ownership of the contract to a new account (`newOwner`).
 * Can only be called by the current owner.
 */
ftrace | funcSig
function transferOwnership(address newOwner) public virtual onlyOwner {
    require(
        newOwner != address(0),
        "Ownable: new owner is the zero address"
    );
    emit OwnershipTransferred(_owner, newOwner);
    _owner = newOwner;
}
```

- The owner can update the dividend tracker.

```
ftrace | funcSig
function updateDividendTracker(address newAddress) public onlyOwner {
    require(
        newAddress != address(dividendTracker),
        "KIRADOGE: The dividend tracker already has that address"
    );

    KIRADOGEDividendTracker newDividendTracker = KIRADOGEDividendTracker(
        payable(newAddress)
    );

    require(
        newDividendTracker.owner() == address(this),
        "KIRADOGE: The new dividend tracker must be owned by the KIRADOGE token contract"
    );

    newDividendTracker.excludeFromDividends(address(newDividendTracker));
    newDividendTracker.excludeFromDividends(address(this));
    newDividendTracker.excludeFromDividends(owner());
    newDividendTracker.excludeFromDividends(address(uniswapV2Router));

    emit UpdateDividendTracker(newAddress, address(dividendTracker));

    dividendTracker = newDividendTracker;
}
```

❖ The owner can update the router address.

```
ftrace | funcSig
function updateUniswapV2Router(address newAddress) public onlyOwner {
    require(
        newAddress != address(uniswapV2Router),
        "KIRADOGE: The router already has that address"
    );
    emit UpdateUniswapV2Router(newAddress, address(uniswapV2Router));
    uniswapV2Router = IUniswapV2Router02(newAddress);
}
```

❖ The owner can exclude wallets from fees.

```
ftrace | funcSig
function excludeFromFees(address account, bool excluded) public onlyOwner {
    require(
        _isExcludedFromFees[account] != excluded,
        "KIRADOGE: Account is already the value of 'excluded'"
    );
    _isExcludedFromFees[account] = excluded;

    emit ExcludeFromFees(account, excluded);
}

ftrace | funcSig
function excludeMultipleAccountsFromFees(
    address[] calldata accounts,
    bool excluded
) public onlyOwner {
    for (uint256 i = 0; i < accounts.length; i++) {
        _isExcludedFromFees[accounts[i]] = excluded;
    }

    emit ExcludeMultipleAccountsFromFees(accounts, excluded);
}
```

❖ The owner can update max buy and sell transaction amount.

```
ftrace | funcSig
function updatemaxBuyTransactionAmount(uint256 newmaxBuyTransactionAmount)
    public
    onlyOwner
{
    require(
        newmaxBuyTransactionAmount != maxBuyTransactionAmount,
        "KIRADOGE: The max buy transaction amount is already this amount"
    );
    emit maxBuyTransactionAmountUpdated(
        newmaxBuyTransactionAmount,
        maxBuyTransactionAmount
    );
    maxBuyTransactionAmount = newmaxBuyTransactionAmount;
}

ftrace | funcSig
function updatemaxSellTransactionAmount(uint256 newmaxSellTransactionAmount)
    public
    onlyOwner
{
    require(
        newmaxSellTransactionAmount != maxSellTransactionAmount,
        "KIRADOGE: The max sell transaction amount is already this amount"
    );
    emit maxSellTransactionAmountUpdated(
        newmaxSellTransactionAmount,
        maxSellTransactionAmount
    );
    maxSellTransactionAmount = newmaxSellTransactionAmount;
}
```

❖ The owner can change the max wallet token amount.

```
ftrace | funcSig
function updatemaxWalletToken(uint256 newmaxWalletToken) public onlyOwner {
    require(
        newmaxWalletToken != maxWalletToken,
        "KIRADOGE: The max wallet token is already this amount"
    );
    emit maxWalletTokenUpdated(newmaxWalletToken, maxWalletToken);
    maxWalletToken = newmaxWalletToken;
}
```

❖ The owner can change the swap point.

```
ftrace | funcSig
function updateswapTokensAtAmount(uint256 newswapTokensAtAmount)
    public
    onlyOwner
{
    require(
        newswapTokensAtAmount != swapTokensAtAmount,
        "KIRADOGE: The swap tokens at amount is already this amount"
    );
    emit setswapTokensAtAmount(newswapTokensAtAmount, swapTokensAtAmount);
    swapTokensAtAmount = newswapTokensAtAmount;
}
```

❖ The owner can update the marketing wallet.

```
ftrace | funcSig
function updateMarketingWallet(address newMarketingWallet)
    public
    onlyOwner
{
    require(
        newMarketingWallet != marketingWallet,
        "KIRADOGE: The marketing wallet is already this address"
    );
    excludeFromFees(newMarketingWallet, true);
    emit MarketingWalletUpdated(newMarketingWallet, marketingWallet);
    marketingWallet = newMarketingWallet;
}
```

❖ The owner can enable/disable trading.

```
ftrace | funcSig
function startTrading(bool _status) public onlyOwner {
    tradingIsEnabled = _status;
}
```

❖ The owner can change the dev wallet and lottery wallet.

```
// update dev wallet
ftrace | funcSig
function updateDevWallet(address newDevWallet) external onlyOwner {
    devWallet = newDevWallet;
}

// update lottery Fee receiver
ftrace | funcSig
function updateLotteryFeeReceiver(address newLotteryFeeReceiver)
    external
    onlyOwner
{
    lotteryFeeReceiver = newLotteryFeeReceiver;
}
```

❖ The owner can change swap percentages.

```
// update swap percentages
ftrace | funcSig
function updateSwapPrecentage(
    uint256 newRewardPrecentage,
    uint256 newLiquidityPrecentage,
    uint256 newMarketingPrecentage,
    uint256 newLotteryPrecentage
) external onlyOwner {
    rewardPrecentage = newRewardPrecentage;
    lotteryPrecentage = newLiquidityPrecentage;
    marketingPrecentage = newMarketingPrecentage;
    liquidityPrecentage = newLotteryPrecentage;
}
```

❖ The owner can change all buy and sell fees.

```
// update total buy fees
ftrace | funcSig
function updateBuyFees(
    uint256 rewardFee,
    uint256 liquidityFee,
    uint256 marketingFee,
    uint256 burnFee,
    uint256 lotteryFee
) external onlyOwner {
    buyRewardFee = rewardFee;
    buyLiquidityFee = liquidityFee;
    buyMarketingFee = marketingFee;
    buyBurnFee = burnFee;
    buyLotteryFee = lotteryFee;

    // calculate total buy fees
    totalBuyFees = buyRewardFee
        .add(buyLiquidityFee)
        .add(buyMarketingFee)
        .add(buyBurnFee)
        .add(buyLotteryFee);
}

// update total sell fees
ftrace | funcSig
function updateSellFees(
    uint256 rewardFee,
    uint256 liquidityFee,
    uint256 marketingFee,
    uint256 burnFee,
    uint256 lotteryFee
) external onlyOwner {
    sellRewardFee = rewardFee;
    sellLiquidityFee = liquidityFee;
    sellMarketingFee = marketingFee;
    sellBurnFee = burnFee;
    sellLotteryFee = lotteryFee;

    // calculate total sell fees
    totalSellFees = sellRewardFee
        .add(sellLiquidityFee)
        .add(sellMarketingFee)
        .add(sellBurnFee)
        .add(sellLotteryFee);
}
```

Audit conclusion

While conducting the audit of the Kira Doge smart contract, it was observed that there is nothing alarming with the code.