Rodeo Solutions

Develop — Audit — Coach

$KNIGHTDOGE Token Smart Contract Audit

Rodeo Studios
July 2021

https://github.com/RodeoSolutions

Contents

Commission

Audited Project	**KNIGHTDOGE Token**
Project website	**Not Provided**
Contract Owner	**0x59a092d876a2ff4a049ea4cb8aa69a95ca107a3c**
Smart Contract Address	**0x3D6c5Fa41c7740D6f1c1b48eBf069cCe4980529E**
Blockchain	**Binance Main Smart Chain**

Rodeo Solutions was commissioned by KNIGHDOGE Token owners to perform an audit of their main smart contract. The purpose of the audit was to achieve the following:

- Ensure that the smart contract functions as intended.
- Identify potential security issues with the smart contract.

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

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 Rodeo Solutions and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers and other representatives) (Rodeo Solutions) owe no duty of care towards you or any other person, nor does Rodeo Solutions 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 Rodeo Solutions 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, Rodeo Solutions hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against Rodeo Solutions, 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.

$KNIGHTDOGE Properties

Contract name	KNIGHTDOGE Token
Contract address	**0xBe4628d7E02e9257875149FA4981C400a01A49A3**
Total supply	1T
Token ticker	$KNIGHTDOGE
Decimals	18
Token holders	1
Transactions count	0
Top 100 holder's dominance	100%
Liquidity fee	0
Tax fee	0
Total fees	0
Mintable	No
Burnable	Yes
Uniswap V2 pair	0x719e3b1365d3b87e166512079ed9153f7b462b03
Contract deployer address	0x59A092d876A2FF4A049eA4cB8aA69A95cA107A3c
Contract's current owner address	0x59a092d876a2ff4a049ea4cb8aa69a95ca107a3c
Marketing Address	0xec70292cf5c5d46dbb37bc72319de3bcd05394e3

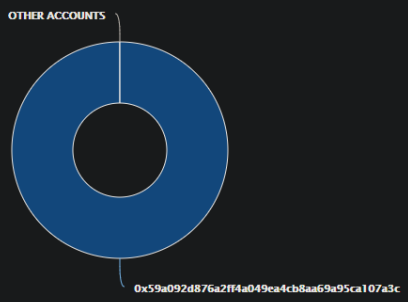
As of 07/31/2021

Token Analytics

The top 100 holders collectively own 100.00% (1,000,000,000,000.00 Tokens) of KNIGHT DOGE | Token Total Supply: 1,000,000,000,000.00 Token | Total Token Holders: 1

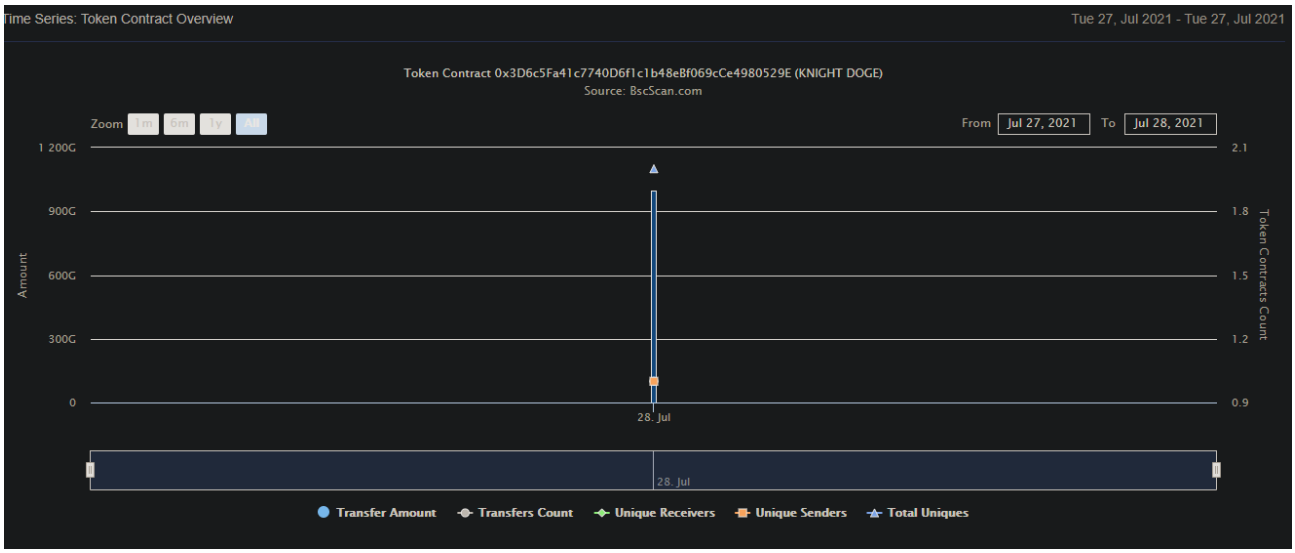
KNIGHT DOGE Top 100 Token Holders

Source: BscScan.com



(A total of 1,000,000,000,000.00 tokens held by the top 100 accounts from the total supply of 1,000,000,000,000.00 token)

Rank	Address	Quantity (Token)	Percentage
1	0x59a092d876a2ff4a049ea4cb8aa69a95ca107a3c	1,000,000,000,000	100.0000%



Contract Functions

Public

View

```
owner()
name()
symbol()
decimals()
totalSupply()
balanceOf(address account)
allowance(address owner, address spender)
dividendOf(address _owner)
withdrawableDividendOf(address _owner)
withdrawnDividendOf(address _owner)
accumulativeDividendOf(address _owner)
getIsExcludedFromFees(address account)
getAccount(address _account)
getAccountAtIndex(uint256 index)
```

Virtual

```
withdrawDividend()
increaseAllowance(address spender, uint256 addedValue)
decreaseAllowance(address spender, uint256 subtractedValue)
```

Executables

```
transfer(address recipient, uint256 amount)
approve(address spender, uint256 amount)
transferFrom(address sender, address recipient, uint256 amount)
deliver(uint256 tAmount)
withdrawDividend()
process(uint256 gas)
```

Owner Executables

```
setDividendTokenAddress(address newToken)
whitelistDxSale(address _presaleAddress, address _routerAddress)
prepareForPartherOrExchangeListing(address _partnerOrExchangeAddress)
setMaxBuyTransaction(uint256 _maxTxn)
setMaxSellTransaction(uint256 _maxTxn)
updateDogeDividendToken(address _newContract)
updateMarketingWallet(address _newWallet)
setMaxWalletTokend(uint256 _maxToken)
setSwapTokensAtAmount(uint256 _swapAmount)
setSellTransactionMultiplier(uint256 _multiplier)
afterPreSale()
setTradingIsEnabled(bool _enabled)
setBuyBackAndLiquifyEnabled(bool _enabled)
setDogeDividendEnabled(bool _enabled)
setMarketingEnabled(bool _enabled)
```

```
updateDogeDividendTracker(address newAddress)
updateDogeDividendRewardFee(uint8 newFee)
updateMarketingFee(uint8 newFee)
updateBuyBackAndLiquidityFee(uint8 newFee)
updateUniswapV2Router(address newAddress)
excludeFromFees(address account, bool excluded)
excludeFromDividend(address account)
excludeMultipleAccountsFromFees(address[] calldata accounts, bool excluded)
setAutomatedMarketMakerPair(address pair, bool value)
_setAutomatedMarketMakerPair(address pair, bool value)
updateGasForProcessing(uint256 newValue)
updateMinimumBalanceForDividends(uint256 newMinimumBalance)
updateClaimWait(uint256 claimWait)
processDividendTracker(uint256 gas)
setDividendTokenAddress(address newToken)
updateMinimumTokenBalanceForDividends(uint256 _newMinimumBalance)
excludeFromDividends(address account)
updateClaimWait(uint256 newClaimWait)
setBalance(address payable account, uint256 newBalance)
processAccount(address payable account, bool automatic)
```

Checklist

Compiler errors.	Passed
Possible delays in data delivery.	Passed
Timestamp dependence.	Low Severity
Integer Overflow and Underflow.	Passed
DoS with Revert.	Passed
DoS with block gas limit.	Passed
Methods execution permissions.	Mid Severity
Economy model of the contract.	Passed
Private user data leaks.	Passed
Malicious Events Log.	Passed
Scoping and Declarations.	Passed
Uninitialized storage pointers.	Passed
Arithmetic accuracy.	Passed
Design Logic.	Passed
Cross-function race conditions.	Passed
Fallback function security.	Passed
Safe Open Zeppelin contracts implementation and usage.	Passed
Whitepaper-Website-Contract correlation.	Low Severity

Owner privileges

KightDoge Contract

- The owner can set when the presale ends by excluding the Dividend contract and presale address

```
function whitelistDxSale(address _presaleAddress, address _routerAddress) external onlyOwner {
    presaleAddress = _presaleAddress;
    dogeDividendTracker.excludeFromDividends(_presaleAddress);
    excludeFromFees(_presaleAddress, true);

    dogeDividendTracker.excludeFromDividends(_routerAddress);
    excludeFromFees(_routerAddress, true);
}
```

- The owner can exclude from fees the Dividend contract and the partnering address

```
function prepareForPartherOrExchangeListing(address _partnerOrExchangeAddress) external onlyOwner {
    dogeDividendTracker.excludeFromDividends(_partnerOrExchangeAddress);
    excludeFromFees(_partnerOrExchangeAddress, true);
}
```

- The Owner can limit the Max Buy amount

```
function setMaxBuyTransaction(uint256 _maxTxn) external onlyOwner {
    maxBuyTranscationAmount = _maxTxn * (10**18);
}
```

- The owner can limit the Max Sell amount

```
function setMaxSellTransaction(uint256 _maxTxn) external onlyOwner {
    maxSellTransactionAmount = _maxTxn * (10**18);
}
```

- The owner can change the Dividend contract address at anytime

```
function updateDogeDividendToken(address _newContract) external onlyOwner {
    dogeDividendToken = _newContract;
    dogeDividendTracker.setDividendTokenAddress(_newContract);
}
```

- The owner can update the Marketing Wallet at anytime

```
function updateMarketingWallet(address _newWallet) external onlyOwner {
    require(_newWallet != marketingWallet, "The marketing wallet is already this address");
    excludeFromFees(_newWallet, true);
    emit MarketingWalletUpdated(marketingWallet, _newWallet);
    marketingWallet = _newWallet;
}
```

- The owner can set the Max amount of tokens per wallet at anytime

```
function setMaxWalletTokend(uint256 _maxToken) external onlyOwner {
    maxWalletToken = _maxToken * (10**18);
}
```

- The owner can set the max swap amount at anytime

```
function setSwapTokensAtAmount(uint256 _swapAmount) external onlyOwner {
    swapTokensAtAmount = _swapAmount * (10**18);
}
```

- The owner can set the transaction multiplier at anytime

```
function setSellTransactionMultiplier(uint256 _multiplier) external onlyOwner {
    sellFeeIncreaseFactor = _multiplier;
}
```

- The owner can set the token properties to the following amount at anytime

```
function afterPreSale() external onlyOwner {
    dogeDividendRewardsFee = 5;
    marketingFee = 3;
    buyBackAndLiquidityFee = 2;
    totalFees = 10;
    marketingEnabled = true;
    buyBackAndLiquifyEnabled = true;
    dogeDividendEnabled = true;
    swapTokensAtAmount = 20000000 * (10**18);
    maxBuyTranscationAmount = 1000000000000 * (10**18);
    maxSellTransactionAmount = 1000000000 * (10**18);
    maxWalletToken = 1000000000000 * (10**18);
}
```

- The owner can stop the trading at anytime

```
function setTradingIsEnabled(bool _enabled) external onlyOwner {
    tradingIsEnabled = _enabled;
}
```

- The owner can disable or enable Buy Back for Liquidity at anytime

```
function setBuyBackAndLiquifyEnabled(bool _enabled) external onlyOwner {
    require(buyBackAndLiquifyEnabled != _enabled, "Can't set flag to same status");
    if (_enabled == false) {
        previousBuyBackAndLiquidityFee = buyBackAndLiquidityFee;
        buyBackAndLiquidityFee = 0;
        buyBackAndLiquifyEnabled = _enabled;
    } else {
        buyBackAndLiquidityFee = previousBuyBackAndLiquidityFee;
        totalFees = buyBackAndLiquidityFee.add(marketingFee).add(dogeDividendRewardsFee);
        buyBackAndLiquifyEnabled = _enabled;
    }

    emit BuyBackAndLiquifyEnabledUpdated(_enabled);
}
```

- The owner can disable the Dividends at anytime

```
function setDogeDividendEnabled(bool _enabled) external onlyOwner {
    require(dogeDividendEnabled != _enabled, "Can't set flag to same status");
    if (_enabled == false) {
        previousDogeDividendRewardsFee = dogeDividendRewardsFee;
        dogeDividendRewardsFee = 0;
        dogeDividendEnabled = _enabled;
    } else {
        dogeDividendRewardsFee = previousDogeDividendRewardsFee;
        totalFees = dogeDividendRewardsFee.add(marketingFee).add(buyBackAndLiquidityFee);
        dogeDividendEnabled = _enabled;
    }

    emit DogeDividendEnabledUpdated(_enabled);
}
```

- The owner can enable or disable the marketing fees at anytime

```
function setMarketingEnabled(bool _enabled) external onlyOwner {
    require(marketingEnabled != _enabled, "Can't set flag to same status");
    if (_enabled == false) {
        previousMarketingFee = marketingFee;
        marketingFee = 0;
        marketingEnabled = _enabled;
    } else {
        marketingFee = previousMarketingFee;
        totalFees = marketingFee.add(dogeDividendRewardsFee).add(buyBackAndLiquidityFee);
        marketingEnabled = _enabled;
    }

    emit MarketingEnabledUpdated(_enabled);
}
```

- The owner can update the dividend tracker contract at anytime

```
function updateDogeDividendTracker(address newAddress) external onlyOwner {
    require(newAddress != address(dogeDividendTracker), "The dividend tracker already has that address");

    DogeDividendTracker newDogeDividendTracker = DogeDividendTracker(payable(newAddress));

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

    newDogeDividendTracker.excludeFromDividends(address(newDogeDividendTracker));
    newDogeDividendTracker.excludeFromDividends(address(this));
    newDogeDividendTracker.excludeFromDividends(address(uniswapV2Router));
    newDogeDividendTracker.excludeFromDividends(address(deadAddress));

    emit UpdateDogeDividendTracker(newAddress, address(dogeDividendTracker));

    dogeDividendTracker = newDogeDividendTracker;
}
```

- The owner can change fees at anytime

```
function updateDogeDividendRewardFee(uint8 newFee) external onlyOwner {
    require(newFee <= 6, "Fee must be less than 6%");
    dogeDividendRewardsFee = newFee;
    totalFees = dogeDividendRewardsFee.add(marketingFee).add(buyBackAndLiquidityFee);
}


function updateMarketingFee(uint8 newFee) external onlyOwner {
    require(newFee <= 6, "Fee must be less than 6%");
    marketingFee = newFee;
    totalFees = marketingFee.add(dogeDividendRewardsFee).add(buyBackAndLiquidityFee);
}

function updateBuyBackAndLiquidityFee(uint8 newFee) external onlyOwner {
    require(newFee <= 6, "Fee must be less than 6%");
    buyBackAndLiquidityFee = newFee;
    totalFees = buyBackAndLiquidityFee.add(dogeDividendRewardsFee).add(marketingFee);
}
```

- The owner can exclude accounts from fees at anytime

```
function excludeFromFees(address account, bool excluded) public onlyOwner {
    require(isExcludedFromFees[account] != excluded, "Account is already exluded from fees");
    isExcludedFromFees[account] = excluded;

    emit ExcludeFromFees(account, excluded);
}

function excludeFromDividend(address account) public onlyOwner {
    dogeDividendTracker.excludeFromDividends(address(account));
}

function excludeMultipleAccountsFromFees(address[] calldata accounts, bool excluded) external onlyOwner {
    for(uint256 i = 0; i < accounts.length; i++) {
        isExcludedFromFees[accounts[i]] = excluded;
    }

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

- The owner can update the UniSwap Router at anytime

```
function updateUniswapV2Router(address newAddress) external onlyOwner {
    require(newAddress != address(uniswapV2Router), "The router already has that address");
    emit UpdateUniswapV2Router(newAddress, address(uniswapV2Router));
    uniswapV2Router = IUniswapV2Router02(newAddress);
}
```

- The owner can prepare the token for new AMM pairs

```
function setAutomatedMarketMakerPair(address pair, bool value) public onlyOwner {
    require(pair != uniswapV2Pair, "The PancakeSwap pair cannot be removed from automatedMarketMakerPairs");

    _setAutomatedMarketMakerPair(pair, value);
}

function _setAutomatedMarketMakerPair(address pair, bool value) private onlyOwner {
    require(automatedMarketMakerPairs[pair] != value, "Automated market maker pair is already set to that value");
    automatedMarketMakerPairs[pair] = value;

    if(value) {
        dogeDividendTracker.excludeFromDividends(pair);
    }

    emit SetAutomatedMarketMakerPair(pair, value);
}
```

- The owner can set gas for processing at anytime

```
function updateGasForProcessing(uint256 newValue) external onlyOwner {
    require(newValue != gasForProcessing, "Cannot update gasForProcessing to same value");
    gasForProcessing = newValue;
    emit GasForProcessingUpdated(newValue, gasForProcessing);
}
```

- The owner can update the minimum balance to be allowed to claim dividends as well as the time to wait between claims

```
function updateMinimumBalanceForDividends(uint256 newMinimumBalance) external onlyOwner {
    dogeDividendTracker.updateMinimumTokenBalanceForDividends(newMinimumBalance);
}

function updateClaimWait(uint256 claimWait) external onlyOwner {
    dogeDividendTracker.updateClaimWait(claimWait);
}
```

- The owner can process the dividends at anytime

```
function processDividendTracker(uint256 gas) external onlyOwner {
    (uint256 ethIterations, uint256 ethClaims, uint256 ethLastProcessedIndex) = dogeDividendTracker.process(gas);
    emit ProcessedDogeDividendTracker(ethIterations, ethClaims, ethLastProcessedIndex, false, gas, tx.origin);
}
```

DogeDividendTracker Contract

- The owner can update the Dividend Tracker contract at anytime

```
function setDividendTokenAddress(address newToken) external override onlyOwner {
  dividendToken = newToken;
}
```

- The owner can update the minimum balance needed to claim balance at anytime

```
function updateMinimumTokenBalanceForDividends(uint256 _newMinimumBalance) external onlyOwner {
    require(_newMinimumBalance != minimumTokenBalanceForDividends, "New mimimum balance for dividend cannot be same as current minimum balar
    minimumTokenBalanceForDividends = _newMinimumBalance * (10**18);
}
```

- The owner can exclude addresses for Dividends at anytime

```
function excludeFromDividends(address account) external onlyOwner {
    require(!excludedFromDividends[account]);
    excludedFromDividends[account] = true;

    _setBalance(account, 0);
    tokenHoldersMap.remove(account);

    emit ExcludeFromDividends(account);
}
```

- The owner can update the claim time at anytime

```
function updateClaimWait(uint256 newClaimWait) external onlyOwner {
    require(newClaimWait >= 3600 && newClaimWait <= 86400, "KnightDoge_Doge_Dividend_Tracker: claimWait must be updated to between 1 and 24
    require(newClaimWait != claimWait, "KnightDoge_Doge_Dividend_Tracker: Cannot update claimWait to same value");
    emit ClaimWaitUpdated(newClaimWait, claimWait);
    claimWait = newClaimWait;
}
```

- The owner can set the dividend balance of an address at anytime

```
function setBalance(address payable account, uint256 newBalance) external onlyOwner {
    if(excludedFromDividends[account]) {
        return;
    }

    if(newBalance >= minimumTokenBalanceForDividends) {
        _setBalance(account, newBalance);
        tokenHoldersMap.set(account, newBalance);
    }
    else {
        _setBalance(account, 0);
        tokenHoldersMap.remove(account);
    }

    processAccount(account, true);
}
```

- The owner can process any address at anytime

```
function processAccount(address payable account, bool automatic) public onlyOwner returns (bool) {
    uint256 amount = _withdrawDividendOfUser(account);

    if(amount > 0) {
        lastClaimTimes[account] = block.timestamp;
        emit Claim(account, amount, automatic);
        return true;
    }

    return false;
}
```

Potential Issues

[Low] Smart Contract presented before DxSale

As of 07/31/2021 when the Audit took place the contract has all fees set at 0% as well as the allowed amounts are still set to zero.

The reason behind this is that the DxSale hasn't go live yet and during the presales the fees have to be set at 0%.

After that the owner will be able to run the afterPresaSale() function in order to set them to the following amounts. Note that this function can be ran at any time rendering this a way to make the contract return to the initial values.

```
function afterPreSale() external onlyOwner {
    dogeDividendRewardsFee = 5;
    marketingFee = 3;
    buyBackAndLiquidityFee = 2;
    totalFees = 10;
    marketingEnabled = true;
    buyBackAndLiquifyEnabled = true;
    dogeDividendEnabled = true;
    swapTokensAtAmount = 20000000 * (10**18);
    maxBuyTranscationAmount = 1000000000000 * (10**18);
    maxSellTransactionAmount = 1000000000 * (10**18);
    maxWalletToken = 1000000000000 * (10**18);
}
```

[Low] Dividend's logic can change at any time

Even though at this time the Dividends contract is written in the same file as the Token contract giving a clear visibility of the Dividend's logic, this can change at any time.

By updating the Dividends tracker address the logic inside the current address would be rendered useless.

This wouldn't present much of an issue as the new address will be public under the owner's address but it must be taken into consideration to prevent future issues.

```
function updateDogeDividendTracker(address newAddress) external onlyOwner {
    require(newAddress != address(dogeDividendTracker), "The dividend tracker already has that address");

    DogeDividendTracker newDogeDividendTracker = DogeDividendTracker(payable(newAddress));

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

    newDogeDividendTracker.excludeFromDividends(address(newDogeDividendTracker));
    newDogeDividendTracker.excludeFromDividends(address(this));
    newDogeDividendTracker.excludeFromDividends(address(uniswapV2Router));
    newDogeDividendTracker.excludeFromDividends(address(deadAddress));

    emit UpdateDogeDividendTracker(newAddress, address(dogeDividendTracker));

    dogeDividendTracker = newDogeDividendTracker;
}
```

```
function setDividendTokenAddress(address newToken) external override onlyOwner {
  dividendToken = newToken;
}
```

[Low] Dividends can be disabled at any time

It must be noted that Dividends can be disabled at any time rendering a big part of the Tokenomics/Properties of the Token useless.

```
function setDogeDividendEnabled(bool _enabled) external onlyOwner {
    require(dogeDividendEnabled != _enabled, "Can't set flag to same status");
    if (_enabled == false) {
        previousDogeDividendRewardsFee = dogeDividendRewardsFee;
        dogeDividendRewardsFee = 0;
        dogeDividendEnabled = _enabled;
    } else {
        dogeDividendRewardsFee = previousDogeDividendRewardsFee;
        totalFees = dogeDividendRewardsFee.add(marketingFee).add(buyBackAndLiquidityFee);
        dogeDividendEnabled = _enabled;
    }

    emit DogeDividendEnabledUpdated(_enabled);
}
```

[Mid] Dividends Contract can set the Balance

Inside the DogeDividendTracker contract, the owner, is able to set the balance of any account to over the minimum value in order to take part of the dividends.

This can pose an issue as the token is not mintable but the dividends can effectively be set by hand. While this is only set to external, this cannot be called from the BSCScan site but it could be interacted with from other sources.

```
function setBalance(address payable account, uint256 newBalance) external onlyOwner {
    if(excludedFromDividends[account]) {
        return;
    }

    if(newBalance >= minimumTokenBalanceForDividends) {
        _setBalance(account, newBalance);
        tokenHoldersMap.set(account, newBalance);
    }
    else {
        _setBalance(account, 0);
        tokenHoldersMap.remove(account);
    }

    processAccount(account, true);
}
```

Conclusions

The Smart Contract code passed the audit successfully on the Binance Mainnet with some considerations to take.

There were three low severity warnings raised meaning that they should be taken into consideration but if the confidence in the owner is good, they can be dismissed.

- Smart contract current Fees and amount states (pre DxSale), will change after the presale. This is not part of the audit.
- Dividends Tracker logic can change at any time by pointing to another contract.
- Dividends can be disabled at any time with no time limit.

One mid severity warning was raised:

- There’s a function inside the DogeDividendTracker which could be interacted with to set the Dividends balance of an account to any value over the minimum limit.

  This can be fixed by setting the function to internal instead of external, meaning that it can be called only from inside the contract.

The last change is advisable in order to provide more security to new holders. Nonetheless this is not necessary if the holders and/or investors feel confident with the contract owners.