freshcoins

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

GROK QUEEN

$GROKQ2

11/01/2024

TOKEN OVERVIEW

Fees

• Buy fees:	7%
• Sell fees:	7%

Fees privileges

• Can change buy fees up to 30% and sell fees up to 30%

Ownership

• Owned

Minting

• No mint function

Max Tx Amount / Max Wallet Amount

• Can change max tx amount and max wallet amount (with threshold)

Blacklist

• Blacklist function not detected

Other privileges

• Can exclude / include from fees

• Contract owner has to call enableTrading function to enable trade

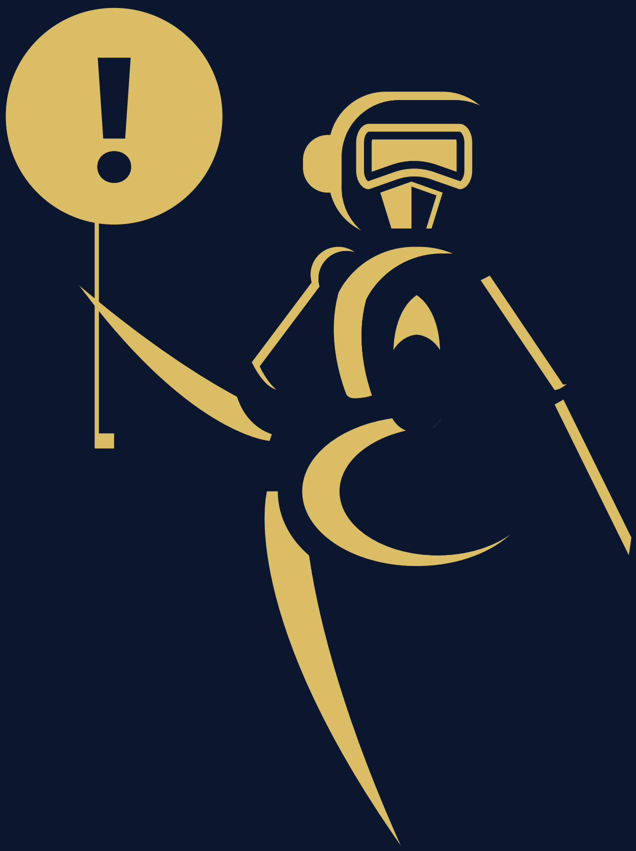
TABLE OF CONTENTS

DISCLAIMER

The information provided on this analysis document is only for general information and should not be used as a reason to invest.

FreshCoins Team will take no payment for manipulating the results of this audit.

The score and the result will stay on this project page information on our website https://freshcoins.io
FreshCoins Team does not guarantees that a project will not sell off team supply, or any other scam strategy (RUG or Honeypot etc)



INTRODUCTION

FreshCoins (Consultant) was contracted by
GROK QUEEN (Customer) to conduct a Smart Contract Code Review and Security Analysis.

0x6A3df69787fCB463Ed5eBbB9caf6445d6dc0E1e7

Network: Binance Smart Chain (BSC)

This report presents the findings of the security assessment of Customer's smart contract and its code review conducted on 11/01/2024

WEBSITE DIAGNOSTIC

https://grokqueen.site/

🟥	🟨	🟩
0-49	50-89	90-100

Performance	Accessibility	Best Practices	SEO	Progressive Web App
92	94	95	91	NA

Socials

Twitter	Telegram
https://twitter.com/grokqueen/	https://t.me/grokqueenofficial

AUDIT OVERVIEW

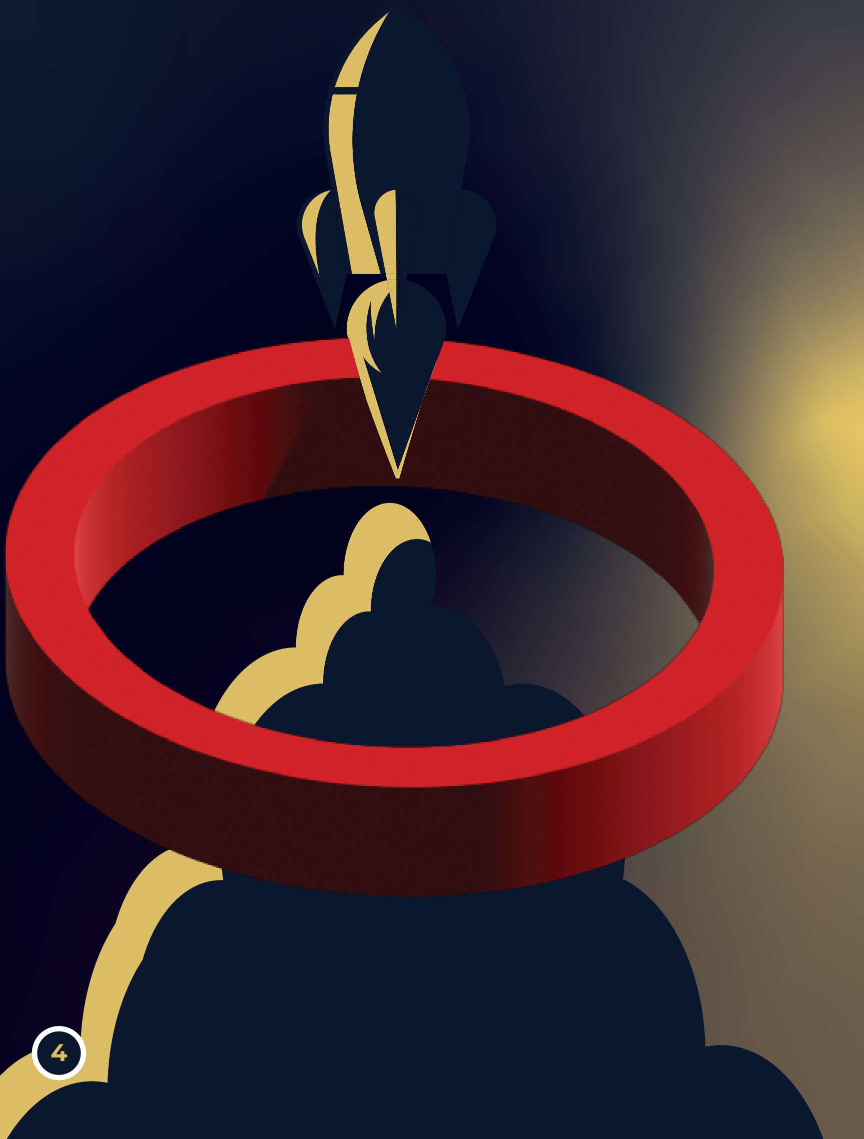
71

Security Score
HIGH RISK
Audit FAIL

93 **Static Scan**
Automatic scanning for common vulnerabilities

86 **ERC Scan**
Automatic checks for ERC's conformance

2	High
1	Medium
0	Low
0	Optimizations
0	Informational



No.	Issue description	Checking Status
1	Compiler Errors / Warnings	Passed
2	Reentrancy and Cross-function	Passed
3	Front running	Low
4	Timestamp dependence	Passed
5	Integer Overflow and Underflow	Passed
6	Reverted DoS	Passed
7	DoS with block gas limit	Low
8	Methods execution permissions	Passed
9	Exchange rate impact	Passed
10	Malicious Event	Passed
11	Scoping and Declarations	Passed
12	Uninitialized storage pointers	Passed
13	Design Logic	Passed
14	Safe Zeppelin module	Passed

OWNER PRIVILEGES

- Contract owner can't mint tokens after initial contract deploy

- Contract owner can't exclude an address from transactions

- Contract owner can exclude/include address from tax

```
function excludeFromFees(address account, bool excluded) external onlyOwner {
    require(_isExcludedFromFees[account] != excluded,"Account is already the value of 'excluded'");
    _isExcludedFromFees[account] = excluded;
    emit ExcludeFromFees(account, excluded);
}
```

- Contract owner can exclude/include address from tx limitations

```
function setExcludeFromMaxTransactionLimit(address account, bool excluded) external onlyOwner {
    require(
        _isExcludedFromMaxTxLimit[account] != exclude,
        "Account is already set to that state"
    );
    _isExcludedFromMaxTxLimit[account] = exclude;
    emit ExcludedFromMaxTransactionLimit(account, exclude);
}
```

- Contract owner can exclude/include address from rewards

```
function excludeFromReward(address account) public onlyOwner {
    require(!_isExcluded[account], "Account is already excluded");
    if (_rOwned[account] > 0) {
        _tOwned[account] = tokenFromReflection(_rOwned[account]);
    }
    _isExcluded[account] = true;
    _excluded.push(account);
}

function includeInReward(address account) external onlyOwner {
    require(_isExcluded[account], "Account is already included");
    for (uint256 i = 0; i < _excluded.length; i++) {
        if (_excluded[i] == account) {
            _excluded[i] = _excluded[_excluded.length - 1];
            _tOwned[account] = 0;
            _isExcluded[account] = false;
            _excluded.pop();
            break;
        }
    }
}
```

Contract owner can exclude/include address from wallet limitations

```
function excludeFromMaxWallet(address account, bool exclude) external onlyOwner {
      require(_isExcludedFromMaxWalletLimit[account] != exclude,"Account is already set to that state");
      require(account != address(this), "Can't set this address.");
      _isExcludedFromMaxWalletLimit[account] = exclude;
      emit ExcludedFromMaxWalletLimit(account, exclude);
}
```

Contract owner has ability to retrieve any token held by the contract

Native tokens excluded

```
function claimStuckTokens(address token) external onlyOwner {
      require(token != address(this), "Owner cannot claim native tokens");
      if (token == address(0x0)) {
         payable(msg.sender).sendValue(address(this).balance);
         return;
      }
      IERC20 ERC20token = IERC20(token);
      uint256 balance = ERC20token.balanceOf(address(this));
      ERC20token.transfer(msg.sender, balance);
}
```

Contract owner can enable/disable tx limitations

```
function setEnableMaxTransactionLimit(bool enable) external onlyOwner {
      require(enable != maxTransactionLimitEnabled, "Max transaction limit is already set to that state");
      maxTransactionLimitEnabled = enable;
      emit MaxTransactionLimitStateChanged(maxTransactionLimitEnabled);
}

function setEnableMaxWalletLimit(bool enable) external onlyOwner {
      require(enable != maxWalletLimitEnabled, "Max wallet limit is already set to that state");
      maxWalletLimitEnabled = enable;
      emit MaxWalletLimitStateChanged(maxWalletLimitEnabled);
}
```

Contract owner can enable/disable wallet to wallet fee

```
function enableWalletToWalletTransferWithoutFee(
      bool enable
   ) external onlyOwner {
      require(
         walletToWalletTransferWithoutFee != enable,
         "Wallet to wallet transfer without fee is already set to that value"
      );
      walletToWalletTransferWithoutFee = enable;
      emit WalletToWalletTransferWithoutFeeEnabled(enable);
}
```

● Contract owner can change marketingWallet address

Current value:

marketingWallet: 0xc2F6CC314f27E38B444a81D91293291147a86A47

```
function changeMarketingWallet(
      address _marketingWallet
   ) external onlyOwner {
      require(
         _marketingWallet != marketingWallet,
         "Marketing wallet is already that address"
      );
      require(
         _marketingWallet != address(0),
         "Marketing wallet is the zero address"
      );
      marketingWallet = _marketingWallet;
      emit MarketingWalletChanged(marketingWallet);
}
```

● Contract owner can change buy fees up to 30% and sell fees up to 30%

```
function setBuyFeePercentages(uint _ReflectionFeeonBuy, uint _liquidityFeeonBuy, uint _marketingFeeon-
Buy, uint _burnFeeOnBuy) external onlyOwner {
      ReflectionFeeonBuy = _ReflectionFeeonBuy;
      liquidityFeeonBuy = _liquidityFeeonBuy;
      marketingFeeonBuy = _marketingFeeonBuy;
      burnFeeOnBuy = _burnFeeOnBuy;

      totalBuyFees = ReflectionFeeonBuy + liquidityFeeonBuy + marketingFeeonBuy + burnFeeOnBuy;
      require(totalBuyFees <= 300, "Buy fees cannot be greater than 30%");

      emit BuyFeesChanged(ReflectionFeeonBuy, liquidityFeeonBuy, marketingFeeonBuy);
}

function setSellFeePercentages(uint _ReflectionFeeonSell, uint _liquidityFeeonSell, uint _marketingFeeonSell,
uint _burnFeeOnSell) external onlyOwner {
      ReflectionFeeonSell = _ReflectionFeeonSell;
      liquidityFeeonSell = _liquidityFeeonSell;
      marketingFeeonSell = _marketingFeeonSell;
      burnFeeOnSell = _burnFeeOnSell;

      totalSellFees = ReflectionFeeonSell + liquidityFeeonSell + marketingFeeonSell + burnFeeOnSell;
      require(totalSellFees <= 300, "Sell fees cannot be greater than 30%");

      emit SellFeesChanged(ReflectionFeeonSell, liquidityFeeonSell, marketingFeeonSell);
}
```

● Contract owner can change swap settings

```
function setSwapTokensAtAmount(uint256 newAmount) external onlyOwner {
      require(
         newAmount > totalSupply() / 1e5,
         "SwapTokensAtAmount must be greater than 0.001% of total supply"
      );
      swapTokensAtAmount = newAmount;
      emit SwapTokensAtAmountUpdated(newAmount);
}

function setSwapEnabled(bool _enabled) external onlyOwner {
      swapEnabled = _enabled;
      emit SwapEnabledUpdated(_enabled);
}
```

● Contract owner has to call enableTrading function to enable trade

Please note that any wallet excluded from fees retains the ability to engage in trading, even in situations where trading has been disabled

```
function enableTrading() external onlyOwner {
      require(tradingEnabled == false, "Trading is already enabled");
      tradingEnabled = true;
}

_transferFrom function line 1089
.
.
.
if (!_isExcludedFromFees[from] && !_isExcludedFromFees[to]) {
         require(tradingEnabled, "Trading is not enabled yet");
}
.
.
.
```

● Contract owner can change tx limitations (with threshold)

```
function setMaxTransactionAmounts(uint256 _maxTransactionAmountBuy, uint256 _maxTransaction-
AmountSell) external onlyOwner {
      require(_maxTransactionAmountBuy >= totalSupply() / 1000 && _maxTransactionAmountSell >=
totalSupply() / 1000, "Max Transaction limis cannot be lower than 0.1% of total supply");
      maxTransactionAmountBuy = _maxTransactionAmountBuy;
      maxTransactionAmountSell = _maxTransactionAmountSell;
      emit MaxTransactionLimitAmountChanged(maxTransactionAmountBuy, maxTransactionAmountSell);
}

function setMaxWalletAmount(uint256 _maxWalletAmount) external onlyOwner {
      require(
         _maxWalletAmount >= totalSupply() / 1000,
         "Max wallet percentage cannot be lower than 0.1%"
      );
      maxWalletAmount = _maxWalletAmount;

      emit MaxWalletLimitAmountChanged(maxWalletAmount);
}
```

● Contract owner can transfer ownership

```
function transferOwnership(address newOwner) public virtual onlyOwner {
        require(
            newOwner != address(0),
            "Ownable: new owner is the zero address"
        );
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
}
```

● Contract owner can renounce ownership

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

Recommendation:

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. The risk can be prevented by temporarily locking the contract or renouncing ownership.

CONCLUSION AND ANALYSIS

Smart Contracts within the scope were manually reviewed and analyzed with static tools.

Audit report overview contains all found security vulnerabilities and other issues in the reviewed code.

Found 2 HIGH issues during the first review.

TOKEN DETAILS

Details

Buy fees:	7%
Sell fees:	7%
Max TX:	420,000,000,000,000,000
Max Sell:	420,000,000,000,000,000

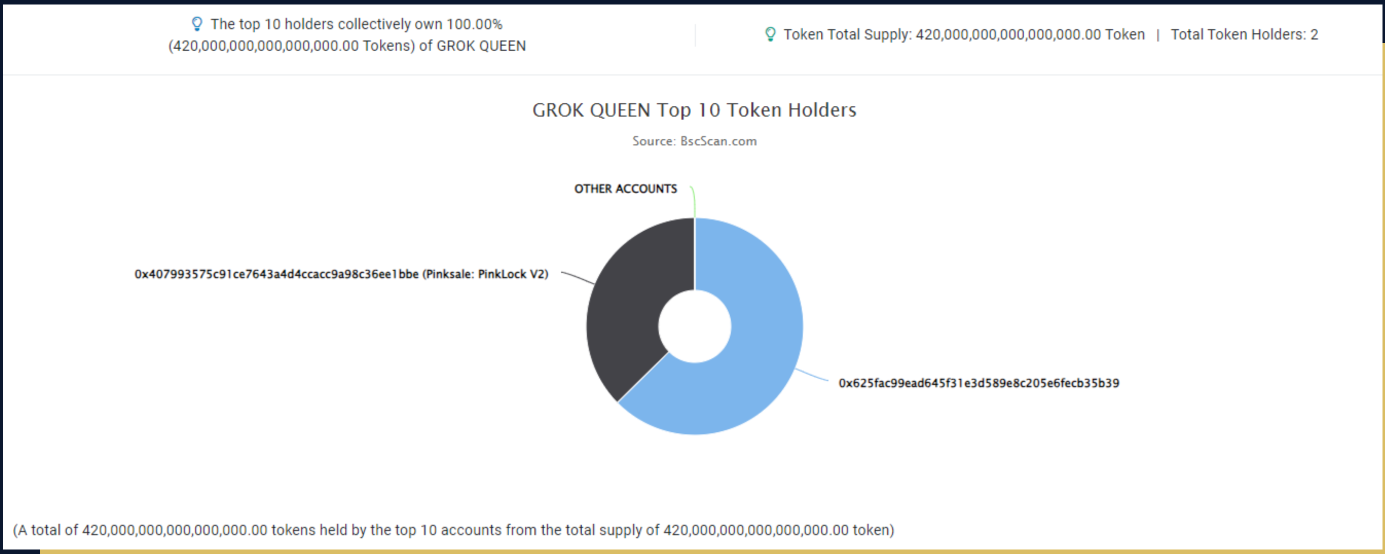
Honeypot Risk

Ownership:	Owned
Blacklist:	Not detected
Modify Max TX:	Detected
Modify Max Sell:	Detected
Disable Trading:	Not detected

Rug Pull Risk

Liquidity:	N/A
Holders:	Clean

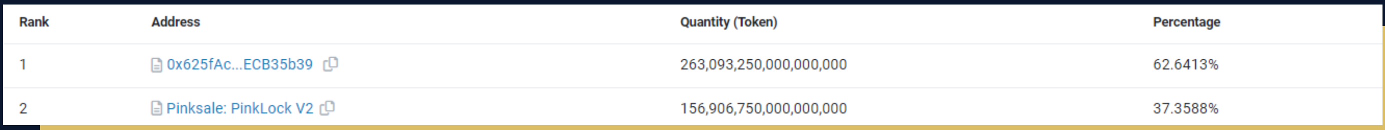
GROKQ2 TOKEN ANALYTICS & TOP 10 TOKEN HOLDERS



The top 10 holders collectively own 100.00% (420,000,000,000,000,000.00 Tokens) of GROK QUEEN

Token Total Supply: 420,000,000,000,000,000.00 Token | Total Token Holders: 2

(A total of 420,000,000,000,000,000.00 tokens held by the top 10 accounts from the total supply of 420,000,000,000,000,000.00 token)



Rank	Address	Quantity (Token)	Percentage
1	0x625fAc...ECB35b39	263,093,250,000,000,000	62.6413%
2	Pinksale: PinkLock V2	156,906,750,000,000,000	37.3588%

TECHNICAL DISCLAIMER

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. The audit can't guarantee the explicit security of the audited project / smart contract.