freshcoins

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

INSANITY WOLF

$INSA

04/04/2022

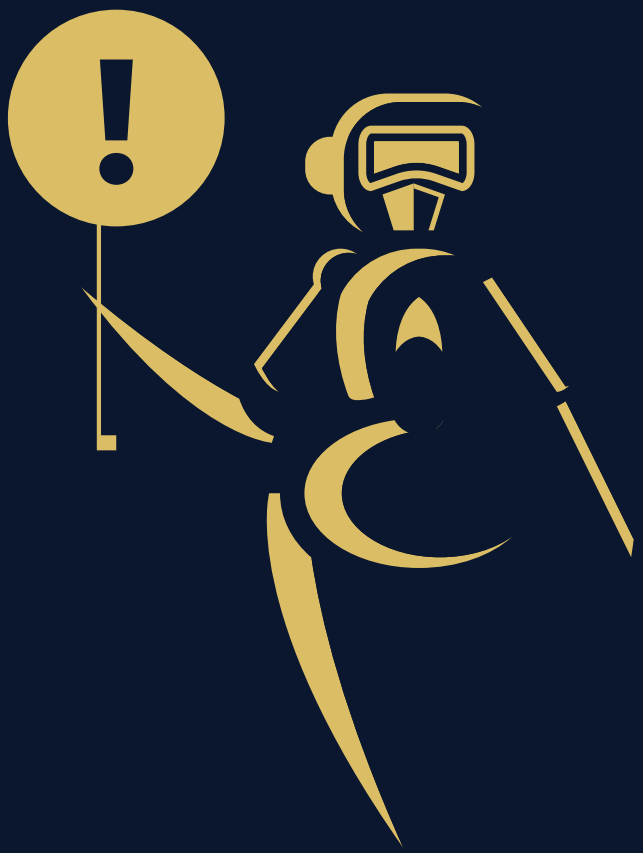
TABLE OF CONTENTS

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 INSANITY WOLF (Customer) to conduct a Smart Contract Code Review and Security Analysis.

0xc849Cc0F615beAca04Bfad84172Df5fd73330dFd

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 04/04/2022

AUDIT OVERVIEW

96

Security Score

94

Static Scan
Automatic scanning for common vulnerabilities

98

ERC Scan
Automatic checks for ERC's conformance

0 High

0 Medium

1 Low

0 Optimizations

0 Informational

No.	Issue description	Checking Status
1	Compiler Errors / Warnings	Passed
2	Reentrancy and Cross-function	Passed
3	Front running	Passed
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 exclude wallet from dividends

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

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

        emit ExcludeFromDividends(account);
    }
```

Contract owner can exclude/include wallet(s) from tax

```
function excludeFromFees(address account, bool excluded) public onlyOwner {
        excludedFromFees[account] = excluded;
        emit ExcludeFromFees(account, excluded);
    }

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

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

Contract owner can transfer tokens to multiple addresses

```
function KKairdrop(address[] memory _address, uint256[] memory _amount) external onlyOwner {
        require(_address.length == _amount.length);
        for(uint i=0; i< _amount.length; i++){
            address adr = _address[i];
            uint amnt = _amount[i] *10**decimals();
            super._transfer(owner(), adr, amnt);
            try insanitywolfDividendTracker.setBalance(payable(adr), balanceOf(adr)) {} catch {}
        }
    }
```

Contract owner can change fees status

```
function setFeesDetails(bool _feeStatus, bool _buyFeeStatus, bool _sellFeeStatus) external onlyOwner {
        feeStatus = _feeStatus;
        buyFeeStatus = _buyFeeStatus;
        sellFeeStatus = _sellFeeStatus;
    }
```

Contract owner can change fees (with threshold)

```
function setFees(uint256 _reward_buy, uint256 _buyBack_buy, uint256 _marketing_buy,
    uint256 _reward_sell,uint256 _buyBack_sell,uint256 _marketing_sell, uint256 _teamBuy, uint256 _team-
Sell) external onlyOwner {
    bool totBFees = _reward_buy+_buyBack_buy+_marketing_buy+_teamBuy < 40;
    bool totSfee = _reward_sell+_buyBack_sell+_marketing_sell+_teamSell < 40;
    require(totBFees && totSfee);
    INSARewardsBuyFee = _reward_buy;
    INSARewardsSellFee = _reward_sell;
    teamBuyFee = _teamBuy;
    teamSellFee = _teamSell;
    buyBackBuyFee  = _buyBack_buy;
    buyBackSellFee  = _buyBack_sell;
    marketingBuyFee = _marketing_buy;
    marketingSellFee = _marketing_sell;
    totalBuyFees = INSARewardsBuyFee.add(teamBuyFee).add(marketingBuyFee).add(buyBackBuyFee);
    totalSellFees = INSARewardsSellFee.add(teamSellFee).add(marketingSellFee).add(buyBackSellFee);
}
```

Contract owner can change marketingWallet, teamWallet, buyBackWallet and nftContract addresses

Current values:

marketingWallet : 0xea0a11ce9326c73e78550285f437bb767ba9f7c9

teamWallet : 0x488e6edfc26df25cc850347b352f59fbf65c385a

buyBackWallet : 0x1a4612ea32627b1f6135386065c8196196a67060

```
function setMarketingWallet(address payable wallet) external onlyOwner{
    marketingWallet = wallet;
}

function setTeamWallet(address newWallet) external onlyOwner{
    teamWallet = newWallet;
}

function setbuyBackWallet(address newWallet) external onlyOwner{
    buyBackWallet = newWallet;
}

function setNftContract(address adr) public onlyOwner {
    nftContract = IERC721(adr);
}
```

Contract owner can set max sell and buy tx amount

```
function setMaxTxAmount(uint _buy, uint _sell) external onlyOwner {
    require(_buy > 100000 && _sell > 100000);
    maxBuyTxAmount = _buy*10**decimals();
    maxSellTxAmount = _sell*10**decimals();
}
```

Contract owner can set wallet limitation (with threshold)

```
function setMaxWallet(uint max) public onlyOwner {
        require(max>1000000*10**decimals());
        maxWallet = max*10**decimals();
}
```

Contract owner can change swap settings

```
function setSwapAndLiquify(bool _state, uint _intervalSecondsForSwap, uint _minimumTokensBeforeSwap,
uint tkToswp) external onlyOwner {
        require(tkToswp < 1000000);
        swapAndLiquifyEnabled = _state;
        intervalSecondsForSwap = _intervalSecondsForSwap;
        minimumTokensBeforeSwap = _minimumTokensBeforeSwap*10**decimals();
        TokensToSwap = tkToswp*10**decimals();
}

function setSwapSend(bool _marketing, bool _team, bool _buyBack) external onlyOwner {
        marketingSwapSendActive = _marketing;
        teamSwapSendActive = _team;
        LiqSwapSendActive = _buyBack;
}
```

Contract owner can transfer tokens from smart contract

```
function transferForeignToken(address _token, address _to, uint256 _value) external onlyOwner returns(bool
_sent){
        if(_value == 0) {
            _value = IERC20(_token).balanceOf(address(this));
        }
        _sent = IERC20(_token).transfer(_to, _value);
}
```

Contract owner can change gap value between trades

```
function setBuySecondLimits(uint buy) external onlyOwner {
        buySecondsLimit = buy;
}
```

Contract owner can enable/disable tx limitations

```
function editLimits(bool buy, bool sell) external onlyOwner {
        limitSells = sell;
        limitBuys = buy;
}
```

Contract owner can allow a wallet to trade when trading is disabled

```
function editPreMarketUser(address _address, bool active) external onlyOwner {
        premarketUser[_address] = active;
}
```

Contract owner can enable/disable trade

```
function activateMarket(bool active) external onlyOwner {
        marketActive = active;
        if (marketActive) {
            MarketActiveAt = block.timestamp;
        }
}

function setMultiBlock(bool _state) external onlyOwner {
        blockMultiBuys = _state;
}
```

Contract owner can renounce ownership

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

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;
}

function betterTransferOwnership(address newowner) public onlyOwner {
        require(newowner != owner());
        excludedFromFees[owner()] = false;
        premarketUser[owner()] = false;
        transferOwnership(newowner);
        excludedFromFees[newowner] = true;
        premarketUser[newowner] = true;
}
```

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 1 LOW issue during the first review.

TOKEN DETAILS

Details

Buy fees:	10%
Sell fees:	12%
Max TX:	1,000,000
Max Sell:	500,000

Honeypot Risk

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

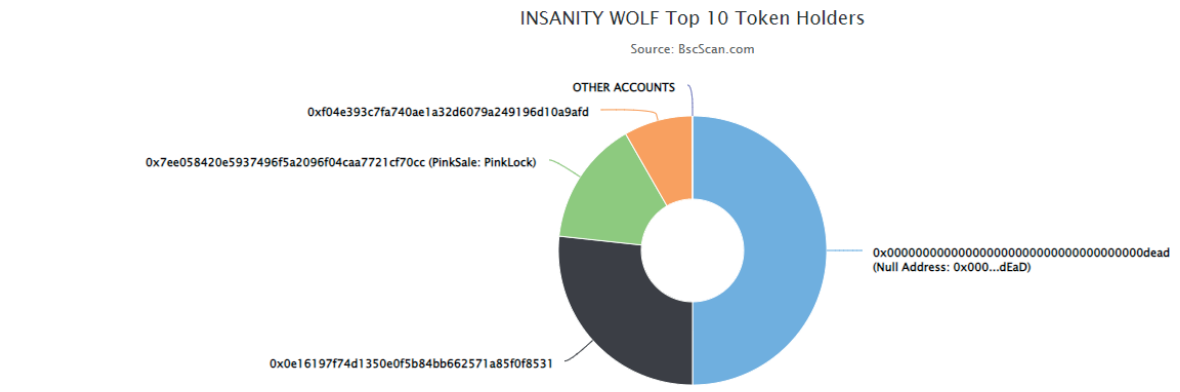
Rug Pull Risk

Liquidity:	N/A
Holders:	Clean

INSANITY WOLF TOKEN ANALYTICS & TOP 10 TOKEN HOLDERS

The top 10 holders collectively own 100.00% (100,000,000.00 Tokens) of INSANITY WOLF

Token Total Supply: 100,000,000.00 Token | Total Token Holders: 4



(A total of 100,000,000.00 tokens held by the top 10 accounts from the total supply of 100,000,000.00 token)

Rank	Address	Quantity (Token)	Percentage
1	Null Address: 0x000...dEaD	50,000,000	50.0000%
2	0x0e16197f74d1350e0f5b84bb662571a85f0f8531	26,708,000	26.7080%
3	PinkSale: PinkLock	15,000,000	15.0000%
4	0xf04e393c7fa740ae1a32d6079a249196d10a9afd	8,292,000	8.2920%

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.