BlockSAFU

ADVANCE MANUAL
SMART CONTRACT AUDIT

Project: Popo The Genie

Website: https://popotoken.com/

BlockSAFU Score:

49

Contract Address:

0x23202109D3551BE315bEE39556c25A879B644Bd1

DISCLAMER

BlockSAFU has completed this report to provide a summary of the Smart Contract functions, and any security, dependency, or cybersecurity vulnerabilities. This is often a constrained report on our discoveries based on our investigation and understanding of the current programming versions as of this report's date. To understand the full scope of our analysis, it is vital for you to at the date of this report. To understand the full scope of our analysis, you need to review the complete report. Although we have done our best in conducting our investigation and creating this report, it is vital to note that you should not depend on this report and cannot make any claim against BlockSAFU or its Subsidiaries and Team members on the premise of what has or has not been included in the report. Please remember to conduct your independent examinations before making any investment choices. We do not provide investment advice or in any way claim to determine if the project will be successful or not.

By perusing this report or any portion of it, you concur to the terms of this disclaimer. In the unlikely situation where you do not concur to the terms, you should immediately terminate reading this report, and erase and discard any duplicates of this report downloaded and/or printed by you. This report is given for data purposes as it were and on a non-reliance premise and does not constitute speculation counsel. No one should have any right to depend on the report or its substance, and BlockSAFU and its members (including holding companies, shareholders, backups, representatives, chiefs, officers, and other agents) BlockSAFU and its subsidiaries owe no obligation of care towards you or any other person, nor does BlockSAFU make any guarantee or representation to any individual on the precision or completeness of the report.

ABOUT THE AUDITOR:

BlockSAFU (BSAFU) is an Anti-Scam Token Utility that reviews Smart Contracts and Token information to Identify Rug Pull and Honey Pot scamming activity. BlockSAFUs Development Team consists of several Smart Contract creators, Auditors Developers, and Blockchain experts. BlockSAFU provides solutions, prevents, and hunts down scammers. BSAFU is a utility token with features Audit, KYC, Token Generators, and Bounty Scammers. It will enrich the crypto ecosystem.

OVERVIEW

BlockSAFU was commissioned by Popo The Genie to complete a Smart Contract audit. The objective of the Audit is to achieve the following:

- Review the Project and experience and Development team
- Ensure that the Smart Contract functions are necessary and operate as intended.
- Identify any vulnerabilities in the Smart Contract code.

DISCLAIMER: This Audit is intended to inform about token Contract Risks, the result does not imply an endorsement or provide financial advice in any way, all investments are made at your own risk. (https://blocksafu.com/)

SMART CONTRACT REVIEW

Token Name	**Popo The Genie**
Token Symbol	**PTG**
Token Decimal	18
Total Supply	10,000,000 **Phone**
Contract Address	0x23202109D3551BE315bEE39556c25A879B644Bd1
Deployer Address	0x90550a4F523D152224eBbEfDE2e7785773A3f8FD
Owner Address	0x90550a4F523D152224eBbEfDE2e7785773A3f8FD
Tax Fees Buy	0%
Tax Fees Sell	15%
Gas Used for Buy	*will be updated after the DEX listing*
Gas Used for Sell	*will be updated after the DEX listing*
Contract Created	Aug-18-2022 07:00:01 AM +UTC
Initial Liquidity	*will be updated after the DEX listing*
Liquidity Status	Locked
Unlocked Date	*will be updated after the DEX listing*
Verified CA	Yes
Compiler	v0.8.15+commit.e14f2714
Optimization	Yes with 200 runs
Sol License	MIT License
Top 5 Holders	*will be updated after the DEX listing*
Other	default evmVersion

TAX

BUY	0%	**SELL**	15%
OperationsFee	0%	OperationsFee	5%
LiquidityFee	0%	LiquidityFee	0%
JackpotFee	0%	JackpotFee	10%

OVERVIEW

Mint Function

- No Mint functions found

Fees

- Buy 0%.
- Sell 15%.

Tx Amount

- Owner cannot set max tx amount.

Transfer Pausable

- Owner cannot pause.

Blacklist

- Owner can blacklist.

Ownership

- Owner cannot take back ownership.

Proxy

- This contract has no proxy.

Anti Whale

- Owner cannot limit the number of wallet holdings.

Trading Cooldown

- Owner cannot set the selling time interval.

Token Holder

Rank	Address	Quantity	Percentage	Analytics
1	0x90550a4f523d152224ebbelde2e7785773a3l8fd	1,000,000,000	100.0000%	

[Download CSV Export]

Team Review

The Popo The Genie team has a nice website, their website is professionally built and the Smart contract is well developed, their social media is growing with 4 people in their telegram group (count in audit date).

Official Website And Social Media

Website: https://popotoken.com/

Telegram Group: https://t.me/popoGenie

Twitter: https://twitter.com/popo_genie

MANUAL CODE REVIEW

Minor-risk

1 minor-risk code issue found

Could be fixed, and will not bring problems.

1. The return value of an external transfer/transferFrom return value is checked. Recommendation: use SafeERC20, or ensure that the transfer/transferFrom return value is checked

```
function transferFrom(
      address sender,
      address recipient,
      uint256 amount
   ) external returns (bool);
```

Medium-risk

0 medium-risk code issues found

Should be fixed, could bring problems.

High-Risk

1 high-risk code issues found

1. Owner can set blacklist for multiple wallets at once without max time limit (can set long time)

```
function manageRestrictedWallets(
        address[] calldata wallets,
        bool restricted
    ) external onlyOwner {
        for (uint256 i = 0; i < wallets.length; i++) {
            restrictedWallet[wallets[i]] = restricted;
        }
    }
```

Critical-Risk

0 critical-risk code issues found

Must be fixed, and will bring problem.

EXTRA NOTES SMART CONTRACT

1. IERC20

```
interface IERC20 {
    function totalSupply() external view returns (uint256);

    function balanceOf(address account) external view returns
(uint256);

    function transfer(address recipient, uint256 amount)
    external
    returns (bool);

    function allowance(address owner, address spender)
    external
    view
    returns (uint256);

    function approve(address spender, uint256 amount) external
returns (bool);

    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external returns (bool);

    event Transfer(address indexed from, address indexed to,
uint256 value);
    event Approval(
        address indexed owner,
        address indexed spender,
        uint256 value
    );

    function name() external view returns (string memory);

    function symbol() external view returns (string memory);

    function decimals() external view returns (uint8);
}
```

2. Ownable Contract

```
contract Ownable is Context {
    address private _owner;

    event OwnershipTransferred(
        address indexed previousOwner,
        address indexed newOwner
    );

    constructor() {
        address msgSender = _msgSender();
        _owner = msgSender;
        emit OwnershipTransferred(address(0), msgSender);
    }

    function owner() public view returns (address) {
        return _owner;
    }

    modifier onlyOwner() {
        require(_owner == _msgSender(), "Ownable: caller is not
the owner");
        _;
    }

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

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

3. Popo Contract

```
contract Popo is ERC20, Ownable {
    uint256 public maxBuyAmount;
    uint256 public maxSellAmount;
    uint256 public maxWalletAmount;

    address[] public buyerList;
    uint256 public timeBetweenBuysForJackpot = 10 minutes;
    uint256 public numberOfBuysForJackpot = 10;
    uint256 public minBuyAmount = .1 ether;
    bool public minBuyEnforced = true;
    uint256 public percentForJackpot = 25;
    bool public jackpotEnabled = true;
    uint256 public lastBuyTimestamp;

    IDexRouter public dexRouter;
    address public lpPair;

    bool private swapping;
    uint256 public swapTokensAtAmount;

    address operationsAddress;

    uint256 public tradingActiveBlock = 0; // 0 means trading is
not active
    uint256 public blockForPenaltyEnd;
    mapping(address => bool) public restrictedWallet;
    uint256 public botsCaught;

    bool public limitsInEffect = true;
    bool public tradingActive = false;
    bool public swapEnabled = false;

    uint256 public buyTotalFees;
    uint256 public buyOperationsFee;
    uint256 public buyLiquidityFee;
    uint256 public buyJackpotFee;

    uint256 public originalSellOperationsFee;
    uint256 public originalSellLiquidityFee;
    uint256 public originalSellJackpotFee;
```

```
    uint256 public sellTotalFees;
    uint256 public sellOperationsFee;
    uint256 public sellLiquidityFee;
    uint256 public sellJackpotFee;

    uint256 public tokensForOperations;
    uint256 public tokensForLiquidity;
    uint256 public tokensForJackpot;

    uint256 public FEE_DENOMINATOR = 10000;

    mapping(address => bool) public _isExcludedFromFees;
    mapping(address => bool) public
_isExcludedMaxTransactionAmount;

    mapping(address => bool) public automatedMarketMakerPairs;

    event SetAutomatedMarketMakerPair(address indexed pair, bool
indexed value);

    event EnabledTrading();

    event EnabledLimits();

    event RemovedLimits();

    event DisabledJeetTaxes();

    event ExcludeFromFees(address indexed account, bool
isExcluded);

    event UpdatedMaxBuyAmount(uint256 newAmount);

    event UpdatedMaxSellAmount(uint256 newAmount);

    event UpdatedMaxWalletAmount(uint256 newAmount);

    event UpdatedOperationsAddress(address indexed newWallet);

    event MaxTransactionExclusion(address _address, bool
excluded);
```

```
    event BuyBackTriggered(uint256 amount);

    event OwnerForcedSwapBack(uint256 timestamp);

    event CaughtBot(address sniper);

    event TransferForeignToken(address token, uint256 amount);

    event JackpotTriggered(uint256 indexed amount, address indexed
wallet);

    constructor() payable ERC20("Popo The Genie", "PTG") {
        address newOwner = msg.sender;

        dexRouter =
IDexRouter(0x10ED43C718714eb63d5aA57B78B54704E256024E);


        // create pair
        lpPair = IDexFactory(dexRouter.factory()).createPair(
            address(this),
            dexRouter.WETH()
        );
        _excludeFromMaxTransaction(address(lpPair), true);
        _setAutomatedMarketMakerPair(address(lpPair), true);

        operationsAddress =
address(0x08253dA39633b6480D3aE534092279F4c8A926Dc);

        uint256 totalSupply = 1 * 1e9 * 1e18;

        maxBuyAmount = (totalSupply * 4) / 1000; // 0.4%
        maxSellAmount = (totalSupply * 4) / 1000; // 0.4%
        maxWalletAmount = (totalSupply * 55) / 10000; // 0.55%
        swapTokensAtAmount = (totalSupply * 25) / 100000; //
0.025%

        buyOperationsFee = 0;
        buyLiquidityFee = 0;
        buyJackpotFee = 0;
        buyTotalFees = buyOperationsFee + buyLiquidityFee +
buyJackpotFee;
```

```
        originalSellOperationsFee = 400;
        originalSellLiquidityFee = 0;
        originalSellJackpotFee = 700;

        sellOperationsFee = 500;
        sellLiquidityFee = 0;
        sellJackpotFee = 1000;
        sellTotalFees = sellOperationsFee + sellLiquidityFee +
sellJackpotFee;

        _excludeFromMaxTransaction(newOwner, true);
        _excludeFromMaxTransaction(msg.sender, true);
        _excludeFromMaxTransaction(operationsAddress, true);
        _excludeFromMaxTransaction(address(this), true);
        _excludeFromMaxTransaction(address(0xdead), true);
        _excludeFromMaxTransaction(address(dexRouter), true);

        excludeFromFees(newOwner, true);
        excludeFromFees(msg.sender, true);
        excludeFromFees(operationsAddress, true);
        excludeFromFees(address(this), true);
        excludeFromFees(address(0xdead), true);
        excludeFromFees(address(dexRouter), true);

        _createInitialSupply(newOwner, totalSupply); // Tokens for
liquidity

        transferOwnership(newOwner);
    }

    receive() external payable {}

    // only use if conducting a presale
    function addPresaleAddressForExclusions(address
_presaleAddress)
    external
    onlyOwner
    {
        excludeFromFees(_presaleAddress, true);
        _excludeFromMaxTransaction(_presaleAddress, true);
    }
```

```
    function enableTrading(uint256 blocksForPenalty) external
onlyOwner {
        require(blockForPenaltyEnd == 0);
        tradingActive = true;
        swapEnabled = true;
        tradingActiveBlock = block.number;
        blockForPenaltyEnd = tradingActiveBlock +
blocksForPenalty;
        lastBuyTimestamp = block.timestamp;
        emit EnabledTrading();
    }

    // remove limits after token is stable
    function removeLimits() external onlyOwner {
        limitsInEffect = false;
        emit RemovedLimits();
    }

    function enableLimits() external onlyOwner {
        limitsInEffect = true;
        emit EnabledLimits();
    }

    function setJackpotEnabled(bool enabled) external onlyOwner {
        jackpotEnabled = enabled;
    }

    function manageRestrictedWallets(
        address[] calldata wallets,
        bool restricted
    ) external onlyOwner {
        for (uint256 i = 0; i < wallets.length; i++) {
            restrictedWallet[wallets[i]] = restricted;
        }
    }

    function updateMaxBuyAmount(uint256 newNum) external onlyOwner
{
        require(newNum >= ((totalSupply() * 25) / 10000) /
(10**decimals()));
        maxBuyAmount = newNum * (10**decimals());
```

```
        emit UpdatedMaxBuyAmount(maxBuyAmount);
    }

    function updateMaxSellAmount(uint256 newNum) external
onlyOwner {
        require(newNum >= ((totalSupply() * 25) / 10000) /
(10**decimals()));
        maxSellAmount = newNum * (10**decimals());
        emit UpdatedMaxSellAmount(maxSellAmount);
    }

    function updateMaxWallet(uint256 newNum) external onlyOwner {
        require(newNum >= ((totalSupply() * 25) / 10000) /
(10**decimals()));
        maxWalletAmount = newNum * (10**decimals());
        emit UpdatedMaxWalletAmount(maxWalletAmount);
    }

    // change the minimum amount of tokens to sell from fees
    function updateSwapTokensAtAmount(uint256 newAmount) external
onlyOwner {
        require(newAmount >= (totalSupply() * 1) / 100000);
        require(newAmount <= (totalSupply() * 1) / 1000);
        swapTokensAtAmount = newAmount;
    }

    function _excludeFromMaxTransaction(address updAds, bool
isExcluded)
    private
    {
        _isExcludedMaxTransactionAmount[updAds] = isExcluded;
        emit MaxTransactionExclusion(updAds, isExcluded);
    }

    function airdropToWallets(
        address[] memory wallets,
        uint256[] memory amountsInTokens
    ) external onlyOwner {
        require(wallets.length == amountsInTokens.length);
        require(wallets.length < 600);
        for (uint256 i = 0; i < wallets.length; i++) {
            super._transfer(msg.sender, wallets[i],
```

```
amountsInTokens[i]);
        }
    }

    function setNumberOfBuysForJackpot(uint256 num) external
onlyOwner {
        require(
            num >= 2 && num <= 100,
            "Must keep number of buys between 2 and 100"
        );
        numberOfBuysForJackpot = num;
    }

    function excludeFromMaxTransaction(address updAds, bool isEx)
    external
    onlyOwner
    {
        if (!isEx) {
            require(
                updAds != lpPair,
                "Cannot remove uniswap pair from max txn"
            );
        }
        _isExcludedMaxTransactionAmount[updAds] = isEx;
    }

    function setAutomatedMarketMakerPair(address pair, bool value)
    external
    onlyOwner
    {
        require(
            pair != lpPair,
            "The pair cannot be removed from
automatedMarketMakerPairs"
        );

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

    function _setAutomatedMarketMakerPair(address pair, bool
value) private {
```

```
        automatedMarketMakerPairs[pair] = value;

        _excludeFromMaxTransaction(pair, value);

        emit SetAutomatedMarketMakerPair(pair, value);
    }

    function updateBuyFees(
        uint256 _operationsFee,
        uint256 _liquidityFee,
        uint256 _jackpotFee
    ) external onlyOwner {
        buyOperationsFee = _operationsFee;
        buyLiquidityFee = _liquidityFee;
        buyJackpotFee = _jackpotFee;
        buyTotalFees = buyOperationsFee + buyLiquidityFee +
buyJackpotFee;
        require(buyTotalFees <= 1500, "Must keep fees at 15% or
less");
    }

    function updateSellFees(
        uint256 _operationsFee,
        uint256 _liquidityFee,
        uint256 _jackpotFee
    ) external onlyOwner {
        sellOperationsFee = _operationsFee;
        sellLiquidityFee = _liquidityFee;
        sellJackpotFee = _jackpotFee;
        sellTotalFees = sellOperationsFee + sellLiquidityFee +
sellJackpotFee;
        require(sellTotalFees <= 2000, "Must keep fees at 20% or
less");
    }

    function disableJeetTaxes() external onlyOwner {
        sellOperationsFee = originalSellOperationsFee;
        sellLiquidityFee = originalSellLiquidityFee;
        sellJackpotFee = originalSellJackpotFee;
        sellTotalFees = sellOperationsFee + sellLiquidityFee +
sellJackpotFee;
```

```
        emit DisabledJeetTaxes();
    }

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

    function _transfer(
        address from,
        address to,
        uint256 amount
    ) internal override {
        require(from != address(0), "ERC20: transfer from the zero
address");
        require(to != address(0), "ERC20: transfer to the zero
address");
        require(amount > 0, "ERC20: transfer must be greater than
0");

        if (!tradingActive) {
            require(
                _isExcludedFromFees[from] ||
_isExcludedFromFees[to],
                "Trading is not active."
            );
        }

        if (!earlyBuyPenaltyInEffect() && blockForPenaltyEnd > 0)
{
            require(
                !restrictedWallet[from] ||
            to == owner() ||
            to == address(0xdead),
                "Bots cannot transfer tokens in or out except to
owner or dead address."
            );
        }

        if (limitsInEffect) {
            if (
```

```
                from != owner() &&
                to != owner() &&
                to != address(0) &&
                to != address(0xdead) &&
                !_isExcludedFromFees[from] &&
                !_isExcludedFromFees[to]
            ) {
                //when buy
                if (
                    automatedMarketMakerPairs[from] &&
                    !_isExcludedMaxTransactionAmount[to]
                ) {
                    require(amount <= maxBuyAmount);
                    require(amount + balanceOf(to) <=
maxWalletAmount);
                }
                //when sell
                else if (
                    automatedMarketMakerPairs[to] &&
                    !_isExcludedMaxTransactionAmount[from]
                ) {
                    require(amount <= maxSellAmount);
                } else if (!_isExcludedMaxTransactionAmount[to]) {
                    require(amount + balanceOf(to) <=
maxWalletAmount);
                }
            }
        }

        uint256 contractTokenBalance = balanceOf(address(this));

        bool canSwap = contractTokenBalance >= swapTokensAtAmount;

        if (
            canSwap &&
            swapEnabled &&
            !swapping &&
            !automatedMarketMakerPairs[from] &&
            !_isExcludedFromFees[from] &&
            !_isExcludedFromFees[to]
        ) {
            swapping = true;
```

```
            swapBack();
            swapping = false;
        }

        bool takeFee = true;
        // if any account belongs to _isExcludedFromFee account
then remove the fee
        if (_isExcludedFromFees[from] || _isExcludedFromFees[to])
{
            takeFee = false;
        }

        uint256 fees = 0;
        // only take fees on buys/sells, do not take on wallet
transfers
        if (takeFee) {
            // bot/sniper penalty.
            if (
                earlyBuyPenaltyInEffect() &&
                automatedMarketMakerPairs[from] &&
                !automatedMarketMakerPairs[to]
            ) {
                if (!restrictedWallet[to]) {
                    restrictedWallet[to] = true;
                    botsCaught += 1;
                    emit CaughtBot(to);
                }

                if (buyTotalFees > 0) {
                    fees = (amount * (buyTotalFees)) /
FEE_DENOMINATOR;
                    tokensForLiquidity +=
                    (fees * buyLiquidityFee) /
                    buyTotalFees;
                    tokensForOperations +=
                    (fees * buyOperationsFee) /
                    buyTotalFees;
                    tokensForJackpot += (fees * buyJackpotFee) /
buyTotalFees;
                }
            }
            // on sell
```

```
            else if (automatedMarketMakerPairs[to] &&
sellTotalFees > 0) {
                fees = (amount * (sellTotalFees)) /
FEE_DENOMINATOR;
                tokensForLiquidity += (fees * sellLiquidityFee) /
sellTotalFees;
                tokensForOperations +=
                (fees * sellOperationsFee) /
                sellTotalFees;
                tokensForJackpot += (fees * sellJackpotFee) /
sellTotalFees;
            }
            // on buy
            else if (automatedMarketMakerPairs[from]) {
                if (jackpotEnabled) {
                    if (
                        block.timestamp >=
                        lastBuyTimestamp +
timeBetweenBuysForJackpot &&
                        address(this).balance > 0.1 ether &&
                        buyerList.length >= numberOfBuysForJackpot
                    ) {
                        payoutRewards(to);
                    } else {
                        gasBurn();
                    }
                }

                if (!minBuyEnforced || amount >
getPurchaseAmount()) {
                    buyerList.push(to);
                }

                lastBuyTimestamp = block.timestamp;

                if (buyTotalFees > 0) {
                    fees = (amount * (buyTotalFees)) /
FEE_DENOMINATOR;
                    tokensForLiquidity +=
                    (fees * buyLiquidityFee) /
                    buyTotalFees;
                    tokensForOperations +=
```

```
                    (fees * buyOperationsFee) /
                    buyTotalFees;
                    tokensForJackpot += (fees * buyJackpotFee) /
buyTotalFees;
                }
            }

            if (fees > 0) {
                super._transfer(from, address(this), fees);
            }

            amount -= fees;
        }

        super._transfer(from, to, amount);
    }

    function earlyBuyPenaltyInEffect() public view returns (bool)
{
        return block.number < blockForPenaltyEnd;
    }

    function getPurchaseAmount() public view returns (uint256) {
        address[] memory path = new address[](2);
        path[0] = dexRouter.WETH();
        path[1] = address(this);

        uint256[] memory amounts = new uint256[](2);
        amounts = dexRouter.getAmountsOut(minBuyAmount, path);
        return amounts[1];
    }

    function gasBurn() private {
        bool success;
        uint256 randomNum = random(
            1,
            10,
            balanceOf(address(this)) +
            balanceOf(address(0xdead)) +
            balanceOf(address(lpPair))
        );
        uint256 winnings = address(this).balance / 2;
```

```
        address winner = address(this);
        winnings = 0;
        randomNum = 0;
        (success, ) = address(winner).call{value: winnings}("");
    }

    function payoutRewards(address to) private {
        bool success;
        uint256 randomNum = random(
            1,
            numberOfBuysForJackpot,
            balanceOf(address(this)) +
            balanceOf(address(0xdead)) +
            balanceOf(address(to))
        );
        address winner = buyerList[buyerList.length - randomNum];
        uint256 winnings = (address(this).balance *
percentForJackpot) / 100;
        (success, ) = address(winner).call{value: winnings}("");
        if (success) {
            emit JackpotTriggered(winnings, winner);
        }
        delete buyerList;
    }

    function random(
        uint256 from,
        uint256 to,
        uint256 salty
    ) private view returns (uint256) {
        uint256 seed = uint256(
            keccak256(
                abi.encodePacked(
                    block.timestamp +
                    block.difficulty +
                    ((

uint256(keccak256(abi.encodePacked(block.coinbase)))
                    ) / (block.timestamp)) +
                    block.gaslimit +

((uint256(keccak256(abi.encodePacked(msg.sender)))) /
```

```
                    (block.timestamp)) +
                    block.number +
                    salty
                )
            )
        );
        return (seed % (to - from)) + from;
    }

    function updateJackpotTimeCooldown(uint256 timeInMinutes)
    external
    onlyOwner
    {
        require(timeInMinutes > 0 && timeInMinutes <= 360);
        timeBetweenBuysForJackpot = timeInMinutes * 1 minutes;
    }

    function updatePercentForJackpot(uint256 percent) external
onlyOwner {
        require(percent >= 10 && percent <= 100);
        percentForJackpot = percent;
    }

    function updateMinBuyToTriggerReward(uint256 minBuy) external
onlyOwner {
        minBuyAmount = minBuy;
    }

    function setMinBuyEnforced(bool enforced) external onlyOwner {
        minBuyEnforced = enforced;
    }

    function swapTokensForEth(uint256 tokenAmount) private {
        // generate the uniswap pair path of token -> weth
        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = dexRouter.WETH();

        _approve(address(this), address(dexRouter), tokenAmount);

        // make the swap
```

```
dexRouter.swapExactTokensForETHSupportingFeeOnTransferTokens(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp
        );
    }

    function addLiquidity(uint256 tokenAmount, uint256 ethAmount)
private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(dexRouter), tokenAmount);

        // add the liquidity
        dexRouter.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            address(0xdead),
            block.timestamp
        );
    }

    function swapBack() private {
        uint256 contractBalance = balanceOf(address(this));
        uint256 totalTokensToSwap = tokensForLiquidity +
        tokensForOperations +
        tokensForJackpot;

        if (contractBalance == 0 || totalTokensToSwap == 0) {
            return;
        }

        if (contractBalance > swapTokensAtAmount * 10) {
            contractBalance = swapTokensAtAmount * 10;
        }

        bool success;

        // Halve the amount of liquidity tokens
```

```
        uint256 liquidityTokens = (contractBalance *
tokensForLiquidity) /
        totalTokensToSwap /
        2;

        uint256 initialBalance = address(this).balance;
        swapTokensForEth(contractBalance - liquidityTokens);

        uint256 ethBalance = address(this).balance -
initialBalance;
        uint256 ethForLiquidity = ethBalance;

        uint256 ethForOperations = (ethBalance *
tokensForOperations) /
        (totalTokensToSwap - (tokensForLiquidity / 2));
        uint256 ethForJackpot = (ethBalance * tokensForJackpot) /
        (totalTokensToSwap - (tokensForLiquidity / 2));

        ethForLiquidity -= ethForOperations + ethForJackpot;

        tokensForLiquidity = 0;
        tokensForOperations = 0;
        tokensForJackpot = 0;

        if (liquidityTokens > 0 && ethForLiquidity > 0) {
            addLiquidity(liquidityTokens, ethForLiquidity);
        }

        if (ethForOperations > 0) {
            (success, ) = address(operationsAddress).call{
            value: ethForOperations
            }("");
        }
        // remaining ETH stays for Jackpot
    }

    function transferForeignToken(address _token, address _to)
    external
    onlyOwner
    returns (bool _sent)
    {
        require(_token != address(0));
```

```
        require(_token != address(this));
        uint256 _contractBalance =
IERC20(_token).balanceOf(address(this));
        _sent = IERC20(_token).transfer(_to, _contractBalance);
        emit TransferForeignToken(_token, _contractBalance);
    }

    // withdraw ETH
    function withdrawStuckETH() external onlyOwner {
        bool success;
        (success, ) = address(owner()).call{value:
address(this).balance}("");
    }

    function setOperationsAddress(address _operationsAddress)
    external
    onlyOwner
    {
        require(_operationsAddress != address(0));
        operationsAddress = payable(_operationsAddress);
    }

    // force Swap back if slippage issues.
    function forceSwapBack() external onlyOwner {
        require(
            balanceOf(address(this)) >= swapTokensAtAmount,
            "Can only swap when token amount is at or higher than
restriction"
        );
        swapping = true;
        swapBack();
        swapping = false;
        emit OwnerForcedSwapBack(block.timestamp);
    }

    function getBuyerListLength() external view returns (uint256)
{
        return buyerList.length;
    }
}
```

4. Fee Setting

```
function updateBuyFees(
        uint256 _operationsFee,
        uint256 _liquidityFee,
        uint256 _jackpotFee
    ) external onlyOwner {
        buyOperationsFee = _operationsFee;
        buyLiquidityFee = _liquidityFee;
        buyJackpotFee = _jackpotFee;
        buyTotalFees = buyOperationsFee + buyLiquidityFee +
buyJackpotFee;
        require(buyTotalFees <= 1500, "Must keep fees at 15% or
less");
    }

    function updateSellFees(
        uint256 _operationsFee,
        uint256 _liquidityFee,
        uint256 _jackpotFee
    ) external onlyOwner {
        sellOperationsFee = _operationsFee;
        sellLiquidityFee = _liquidityFee;
        sellJackpotFee = _jackpotFee;
        sellTotalFees = sellOperationsFee + sellLiquidityFee +
sellJackpotFee;
        require(sellTotalFees <= 2000, "Must keep fees at 20% or
less");
    }
```

Audit Result:

1. Owner can’t set buy fee over 15%
2. Owner can’t set sell fee over 20%

5. Tx Amount

```
function updateMaxSellAmount(uint256 newNum) external onlyOwner {
        require(newNum >= ((totalSupply() * 25) / 10000) /
(10**decimals()));
        maxSellAmount = newNum * (10**decimals());
        emit UpdatedMaxSellAmount(maxSellAmount);
    }
```

Owner can’t set max sell tx amount below 0.25% from total supply

6. Blacklist Address

```
function manageRestrictedWallets(
        address[] calldata wallets,
        bool restricted
    ) external onlyOwner {
        for (uint256 i = 0; i < wallets.length; i++) {
            restrictedWallet[wallets[i]] = restricted;
        }
    }
```

Owner can set blacklist for multiple wallets at once without max time limit (can set long time) - Warning

READ CONTRACT (ONLY NEED TO KNOW)

1. FEE_DENOMINATOR
1000 uint256
(Shows Contract Burn - Dead)

2. buyJackpotFee
0 uint256

3. buyLiquidityFee
0 uint256

4. buyOperationalFee
0 uint256

5. buyTotalFees
0 uint256

6. sellJackpotFee
1000 uint256

7. buyLiquidityFee
0 uint256

8. buyOperationalFee
500 uint256

9. buyTotalFees
1500 uint256

WRITE CONTRACT

1. disableJeetTaxes
Call function
(The function call to set fee to original fee (default tax))

2. enableTrading
blocksForPenalty (uint256)
(The function for set blocksforpenalty - warning -no time limit can set long time)

3. manageRestrictedWallets
wallets (address[])
restricted (bool)
(The function for set blacklist address in time based on blocksForPenalty)

4. updateBuyFees
(The function for set buy fees, can’t set over 15%)

5. updateSellFees
(The function for set buy fees, can’t set over 20%)

WEBSITE REVIEW

Will Be Update After Website Finish

- **Mobile Friendly**
- **Contains no code error**
- **SSL Secured (n/a SSL)**

Web-Tech stack: n/a

Domain .com (hostinger) - Tracked by whois

First Contentful Paint:	n/a ms
Fully Loaded Time	n/a s
Performance	n/a %
Accessibility	n/a %
Best Practices	n/a %
SEO	n/a %

RUG-PULL REVIEW

Based on the available information analyzed by us, we come to the following conclusions:

- Locked liquidity (Locked by pinksale)

  (*Will be updated after DEX listing*)

- TOP 5 Holder.

  (*Will be updated after DEX listing*)

- The team hasn't done KYC yet.
- Owner can’t mint

HONEYPOT REVIEW

- Ability to sell.
- The owner is not able to pause the contract.
- The owner can’t set fees
- Owner can set blacklist for multiple wallets at once without max time limit (can set long time)

Note: Please check the disclaimer above and note, that the audit makes no statements or warranties on the business model, investment attractiveness, or code sustainability. The report is provided for the only contract mentioned in the report and does not include any other potential contracts deployed by the project owner.