BlockSAFU

ADVANCE MANUAL
SMART CONTRACT AUDIT

Project: DogeIpa

Website: https://dogeipa.online/

PASSED

BlockSAFU Score:

82

Contract Address:

0x7a45d24aFfE81e98a03eE68D10cB2Dc1f857676b

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 Doge Ipa 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	**Doge Ipa**
Token Symbol	**Dipa**
Token Decimal	18
Total Supply	250,000,000 **Dipa**
Contract Address	0x7a45d24aFfE81e98a03eE68D10cB2Dc1f857676b
Deployer Address	0x2c293c3393cD63D53A794CdE657D51D673B144AA
Owner Address	0x2c293c3393cD63D53A794CdE657D51D673B144AA
Tax Fees Buy	9%
Tax Fees Sell	9%
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-27-2022 09:28:48 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	v0.8.4+commit.c7e474f2
Optimization	Yes with 200 runs
Sol License	MIT License
Top 5 Holders	*will be updated after the DEX listing*
Other	default evmVersion

TAX

BUY	9%	**SELL**	9%
Liquidity Fee	2%	Liquidity Fee	2%
Marketing Fee	2%	Marketing Fee	2%
Reward Fee	5%	Reward Fee	5%

OVERVIEW

Mint Function

- No mint functions.

Fees

- Buy 9% (owner cannot set fees over 25%).
- Sell 9% (owner can’t set fees over 25%).

Tx Amount

- Owner cannot set max tx amount.

Transfer Pausable

- Owner cannot pause.

Blacklist

- Owner cannot 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	0x2c293c3393cd63d53a794cde657d51d673b144ae	250,000,000	100.0000%	

[Download CSV Export]

Team Review

The Doge Ipa team has a nice website, their website is professionally built and the Smart contract is well developed, their social media is growing with over 456 people in their telegram group (count in audit date).

Official Website And Social Media

Website: https://dogeipa.online/

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

Twitter: https://twitter.com/DogeipaArmy?t=T6DQ55ubADXhXQfF35PMqA&s=35

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

0 high-risk code issues found

Must be fixed, and will bring problem.

Critical-Risk

0 critical-risk code issues found

Must be fixed, and will bring problem.

EXTRA NOTES SMART CONTRACT

1. IERC20

```
interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns
(uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to
`recipient`.
     *
     * Returns a boolean value indicating whether the operation
succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external
returns (bool);

    /**
     * @dev Returns the remaining number of tokens that `spender`
will be
     * allowed to spend on behalf of `owner` through
{transferFrom}. This is
     * zero by default.
     *
     * This value changes when {approve} or {transferFrom} are
called.
     */
    function allowance(address owner, address spender) external
view returns (uint256);

    /**
     * @dev Sets `amount` as the allowance of `spender` over the
```

```
caller's tokens.
     *
     * Returns a boolean value indicating whether the operation
succeeded.
     *
     * IMPORTANT: Beware that changing an allowance with this
method brings the risk
     * that someone may use both the old and the new allowance by
unfortunate
     * transaction ordering. One possible solution to mitigate this
race
     * condition is to first reduce the spender's allowance to 0
and set the
     * desired value afterwards:
     *
https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external
returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient`
using the
     * allowance mechanism. `amount` is then deducted from the
caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation
succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(
        address sender,
        address recipient,
        uint256 amount
    ) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account
```

```
(`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to,
uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an
`owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender,
uint256 value);
}
```

IERC20 Normal Base Template

2. SafeMath Contract

```
library SafeMath {
...
    function add(uint256 a, uint256 b) internal pure returns
(uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");
        return c;
    }
...
    function sub(uint256 a, uint256 b, string memory errorMessage)
internal pure returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    /**
     * @dev Returns the multiplication of two unsigned integers,
reverting on
     * overflow.
     *
     * Counterpart to Solidity's `*` operator.
     *
     * Requirements:
     *
     * - Multiplication cannot overflow.
     */
  ...
    function mod(
        uint256 a,
        uint256 b,
        string memory errorMessage
    ) internal pure returns (uint256) {
        unchecked {
            require(b > 0, errorMessage);
            return a % b;
        }
    }
}
```

Standard Safemath contract

3. Doge Ipa Contract

```
contract AntiBotBABYTOKEN is ERC20, Ownable, BaseToken {
    using SafeMath for uint256;

    uint256 public constant VERSION = 1;

    IUniswapV2Router02 public uniswapV2Router;
    address public uniswapV2Pair;

    bool private swapping;

    BABYTOKENDividendTracker public dividendTracker;

    address public rewardToken;

    uint256 public swapTokensAtAmount;

    uint256 public tokenRewardsFee;
    uint256 public liquidityFee;
    uint256 public marketingFee;
    uint256 public totalFees;

    address public _marketingWalletAddress;

    uint256 public gasForProcessing;

    // exlcude from fees and max transaction amount
    mapping(address => bool) private _isExcludedFromFees;

    // store addresses that a automatic market maker pairs. Any
transfer *to* these addresses
    // could be subject to a maximum transfer amount
    mapping(address => bool) public automatedMarketMakerPairs;

    IPinkAntiBot public pinkAntiBot;
    bool public enableAntiBot;

    event UpdateDividendTracker(
        address indexed newAddress,
        address indexed oldAddress
    );
```

```
    event UpdateUniswapV2Router(
        address indexed newAddress,
        address indexed oldAddress
    );

    event ExcludeFromFees(address indexed account, bool
isExcluded);
    event ExcludeMultipleAccountsFromFees(address[] accounts, bool
isExcluded);

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

    event LiquidityWalletUpdated(
        address indexed newLiquidityWallet,
        address indexed oldLiquidityWallet
    );

    event GasForProcessingUpdated(
        uint256 indexed newValue,
        uint256 indexed oldValue
    );

    event SwapAndLiquify(
        uint256 tokensSwapped,
        uint256 ethReceived,
        uint256 tokensIntoLiqudity
    );

    event SendDividends(uint256 tokensSwapped, uint256 amount);

    event ProcessedDividendTracker(
        uint256 iterations,
        uint256 claims,
        uint256 lastProcessedIndex,
        bool indexed automatic,
        uint256 gas,
        address indexed processor
    );

    constructor(
        string memory name_,
```

```
        string memory symbol_,
        uint256 totalSupply_,
        address[5] memory addrs, // reward, router, marketing
wallet, dividendTracker, anti bot
        uint256[3] memory feeSettings, // rewards, liquidity,
marketing
        uint256 minimumTokenBalanceForDividends_,
        address serviceFeeReceiver_,
        uint256 serviceFee_
    ) payable ERC20(name_, symbol_) {
        rewardToken = addrs[0];
        _marketingWalletAddress = addrs[2];
        require(
            msg.sender != _marketingWalletAddress,
            "Owner and marketing wallet cannot be the same"
        );

        pinkAntiBot = IPinkAntiBot(addrs[4]);
        pinkAntiBot.setTokenOwner(owner());
        enableAntiBot = true;

        tokenRewardsFee = feeSettings[0];
        liquidityFee = feeSettings[1];
        marketingFee = feeSettings[2];
        totalFees =
tokenRewardsFee.add(liquidityFee).add(marketingFee);
        require(totalFees <= 25, "Total fee is over 25%");
        swapTokensAtAmount = totalSupply_.mul(2).div(10**6); //
0.002%

        // use by default 300,000 gas to process auto-claiming
dividends
        gasForProcessing = 300000;

        dividendTracker = BABYTOKENDividendTracker(
            payable(Clones.clone(addrs[3]))
        );
        dividendTracker.initialize(
            rewardToken,
            minimumTokenBalanceForDividends_
        );
```

```
        IUniswapV2Router02 _uniswapV2Router =
IUniswapV2Router02(addrs[1]);
        // Create a uniswap pair for this new token
        address _uniswapV2Pair =
IUniswapV2Factory(_uniswapV2Router.factory())
            .createPair(address(this), _uniswapV2Router.WETH());
        uniswapV2Router = _uniswapV2Router;
        uniswapV2Pair = _uniswapV2Pair;
        _setAutomatedMarketMakerPair(_uniswapV2Pair, true);

        // exclude from receiving dividends

dividendTracker.excludeFromDividends(address(dividendTracker));
        dividendTracker.excludeFromDividends(address(this));
        dividendTracker.excludeFromDividends(owner());
        dividendTracker.excludeFromDividends(address(0xdead));

dividendTracker.excludeFromDividends(address(_uniswapV2Router));
        // exclude from paying fees or having max transaction
amount
        excludeFromFees(owner(), true);
        excludeFromFees(_marketingWalletAddress, true);
        excludeFromFees(address(this), true);
        /*
            _mint is an internal function in ERC20.sol that is
only called here,
            and CANNOT be called ever again
        */
        _mint(owner(), totalSupply_);

        emit TokenCreated(
            owner(),
            address(this),
            TokenType.antiBotBaby,
            VERSION
        );

        payable(serviceFeeReceiver_).transfer(serviceFee_);
    }

    function setEnableAntiBot(bool _enable) external onlyOwner {
        enableAntiBot = _enable;
```

```
    }

    receive() external payable {}

    function setSwapTokensAtAmount(uint256 amount) external
onlyOwner {
        swapTokensAtAmount = amount;
    }

    function updateDividendTracker(address newAddress) public
onlyOwner {
        require(
            newAddress != address(dividendTracker),
            "BABYTOKEN: The dividend tracker already has that
address"
        );

        BABYTOKENDividendTracker newDividendTracker =
BABYTOKENDividendTracker(
            payable(newAddress)
        );

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


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

newDividendTracker.excludeFromDividends(address(uniswapV2Router));

        emit UpdateDividendTracker(newAddress,
address(dividendTracker));

        dividendTracker = newDividendTracker;
    }
```

```
    function updateUniswapV2Router(address newAddress) public
onlyOwner {
        require(
            newAddress != address(uniswapV2Router),
            "BABYTOKEN: The router already has that address"
        );
        emit UpdateUniswapV2Router(newAddress,
address(uniswapV2Router));
        uniswapV2Router = IUniswapV2Router02(newAddress);
        address _uniswapV2Pair =
IUniswapV2Factory(uniswapV2Router.factory())
            .createPair(address(this), uniswapV2Router.WETH());
        uniswapV2Pair = _uniswapV2Pair;
    }

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

        emit ExcludeFromFees(account, excluded);
    }

    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);
    }

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

```
    function setTokenRewardsFee(uint256 value) external onlyOwner
{
        tokenRewardsFee = value;
        totalFees =
tokenRewardsFee.add(liquidityFee).add(marketingFee);
        require(totalFees <= 25, "Total fee is over 25%");
    }

    function setLiquiditFee(uint256 value) external onlyOwner {
        liquidityFee = value;
        totalFees =
tokenRewardsFee.add(liquidityFee).add(marketingFee);
        require(totalFees <= 25, "Total fee is over 25%");
    }

    function setMarketingFee(uint256 value) external onlyOwner {
        marketingFee = value;
        totalFees =
tokenRewardsFee.add(liquidityFee).add(marketingFee);
        require(totalFees <= 25, "Total fee is over 25%");
    }

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

        _setAutomatedMarketMakerPair(pair, value);
    }

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

```
        );
        automatedMarketMakerPairs[pair] = value;

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

        emit SetAutomatedMarketMakerPair(pair, value);
    }

    function updateGasForProcessing(uint256 newValue) public
onlyOwner {
        require(
            newValue >= 200000 && newValue <= 500000,
            "BABYTOKEN: gasForProcessing must be between 200,000
and 500,000"
        );
        require(
            newValue != gasForProcessing,
            "BABYTOKEN: Cannot update gasForProcessing to same
value"
        );
        emit GasForProcessingUpdated(newValue, gasForProcessing);
        gasForProcessing = newValue;
    }

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

    function getClaimWait() external view returns (uint256) {
        return dividendTracker.claimWait();
    }

    function updateMinimumTokenBalanceForDividends(uint256 amount)
        external
        onlyOwner
    {

dividendTracker.updateMinimumTokenBalanceForDividends(amount);
    }
```

```
    function getMinimumTokenBalanceForDividends()
        external
        view
        returns (uint256)
    {
        return dividendTracker.minimumTokenBalanceForDividends();
    }

    function getTotalDividendsDistributed() external view returns
(uint256) {
        return dividendTracker.totalDividendsDistributed();
    }

    function isExcludedFromFees(address account) public view
returns (bool) {
        return _isExcludedFromFees[account];
    }

    function withdrawableDividendOf(address account)
        public
        view
        returns (uint256)
    {
        return dividendTracker.withdrawableDividendOf(account);
    }

    function dividendTokenBalanceOf(address account)
        public
        view
        returns (uint256)
    {
        return dividendTracker.balanceOf(account);
    }

    function excludeFromDividends(address account) external
onlyOwner {
        dividendTracker.excludeFromDividends(account);
    }

    function isExcludedFromDividends(address account)
        public
```

```
        view
        returns (bool)
    {
        return dividendTracker.isExcludedFromDividends(account);
    }

    function getAccountDividendsInfo(address account)
        external
        view
        returns (
            address,
            int256,
            int256,
            uint256,
            uint256,
            uint256,
            uint256,
            uint256
        )
    {
        return dividendTracker.getAccount(account);
    }

    function getAccountDividendsInfoAtIndex(uint256 index)
        external
        view
        returns (
            address,
            int256,
            int256,
            uint256,
            uint256,
            uint256,
            uint256,
            uint256
        )
    {
        return dividendTracker.getAccountAtIndex(index);
    }

    function processDividendTracker(uint256 gas) external {
        (
```

```
            uint256 iterations,
            uint256 claims,
            uint256 lastProcessedIndex
        ) = dividendTracker.process(gas);
        emit ProcessedDividendTracker(
            iterations,
            claims,
            lastProcessedIndex,
            false,
            gas,
            tx.origin
        );
    }

    function claim() external {
        dividendTracker.processAccount(payable(msg.sender),
false);
    }

    function getLastProcessedIndex() external view returns
(uint256) {
        return dividendTracker.getLastProcessedIndex();
    }

    function getNumberOfDividendTokenHolders() external view
returns (uint256) {
        return dividendTracker.getNumberOfTokenHolders();
    }

    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");

        if (enableAntiBot) {
            pinkAntiBot.onPreTransferCheck(from, to, amount);
        }
```

```
        if (amount == 0) {
            super._transfer(from, to, 0);
            return;
        }

        uint256 contractTokenBalance = balanceOf(address(this));

        bool canSwap = contractTokenBalance >= swapTokensAtAmount;

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

            uint256 marketingTokens = contractTokenBalance
                .mul(marketingFee)
                .div(totalFees);
            swapAndSendToFee(marketingTokens);

            uint256 swapTokens =
contractTokenBalance.mul(liquidityFee).div(
                totalFees
            );
            swapAndLiquify(swapTokens);

            uint256 sellTokens = balanceOf(address(this));
            swapAndSendDividends(sellTokens);

            swapping = false;
        }

        bool takeFee = !swapping;

        // if any account belongs to _isExcludedFromFee account
then remove the fee
        if (_isExcludedFromFees[from] || _isExcludedFromFees[to])
{
```

```
            takeFee = false;
        }

        if (takeFee) {
            uint256 fees = amount.mul(totalFees).div(100);
            if (automatedMarketMakerPairs[to]) {
                fees += amount.mul(1).div(100);
            }
            amount = amount.sub(fees);

            super._transfer(from, address(this), fees);
        }

        super._transfer(from, to, amount);

        try
            dividendTracker.setBalance(payable(from),
balanceOf(from))
        {} catch {}
        try dividendTracker.setBalance(payable(to), balanceOf(to))
{} catch {}

        if (!swapping) {
            uint256 gas = gasForProcessing;

            try dividendTracker.process(gas) returns (
                uint256 iterations,
                uint256 claims,
                uint256 lastProcessedIndex
            ) {
                emit ProcessedDividendTracker(
                    iterations,
                    claims,
                    lastProcessedIndex,
                    true,
                    gas,
                    tx.origin
                );
            } catch {}
        }
    }
```

```
    function swapAndSendToFee(uint256 tokens) private {
        uint256 initialCAKEBalance =
IERC20(rewardToken).balanceOf(
            address(this)
        );

        swapTokensForCake(tokens);
        uint256 newBalance =
(IERC20(rewardToken).balanceOf(address(this))).sub(
            initialCAKEBalance
        );
        IERC20(rewardToken).transfer(_marketingWalletAddress,
newBalance);
    }

    function swapAndLiquify(uint256 tokens) private {
        // split the contract balance into halves
        uint256 half = tokens.div(2);
        uint256 otherHalf = tokens.sub(half);

        // capture the contract's current ETH balance.
        // this is so that we can capture exactly the amount of
ETH that the
        // swap creates, and not make the liquidity event include
any ETH that
        // has been manually sent to the contract
        uint256 initialBalance = address(this).balance;

        // swap tokens for ETH
        swapTokensForEth(half); // <- this breaks the ETH -> HATE
swap when swap+liquify is triggered

        // how much ETH did we just swap into?
        uint256 newBalance =
address(this).balance.sub(initialBalance);

        // add liquidity to uniswap
        addLiquidity(otherHalf, newBalance);

        emit SwapAndLiquify(half, newBalance, otherHalf);
    }
```

```
    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] = uniswapV2Router.WETH();

        _approve(address(this), address(uniswapV2Router),
tokenAmount);

        // make the swap

uniswapV2Router.swapExactTokensForETHSupportingFeeOnTransferTokens
(
            tokenAmount,
            0, // accept any amount of ETH
            path,
            address(this),
            block.timestamp
        );
    }

    function swapTokensForCake(uint256 tokenAmount) private {
        address[] memory path = new address[](3);
        path[0] = address(this);
        path[1] = uniswapV2Router.WETH();
        path[2] = rewardToken;

        _approve(address(this), address(uniswapV2Router),
tokenAmount);

        // make the swap

uniswapV2Router.swapExactTokensForTokensSupportingFeeOnTransferTok
ens(
            tokenAmount,
            0,
            path,
            address(this),
            block.timestamp
        );
    }
```

```
    function addLiquidity(uint256 tokenAmount, uint256 ethAmount)
private {
        // approve token transfer to cover all possible scenarios
        _approve(address(this), address(uniswapV2Router),
tokenAmount);

        // add the liquidity
        uniswapV2Router.addLiquidityETH{value: ethAmount}(
            address(this),
            tokenAmount,
            0, // slippage is unavoidable
            0, // slippage is unavoidable
            address(0),
            block.timestamp
        );
    }

    function swapAndSendDividends(uint256 tokens) private {
        swapTokensForCake(tokens);
        uint256 dividends =
IERC20(rewardToken).balanceOf(address(this));
        bool success = IERC20(rewardToken).transfer(
            address(dividendTracker),
            dividends
        );

        if (success) {
            dividendTracker.distributeCAKEDividends(dividends);
            emit SendDividends(tokens, dividends);
        }
    }
}
```

4. Tax Fee contract

```
function setTokenRewardsFee(uint256 value) external onlyOwner {
        tokenRewardsFee = value;
        totalFees =
tokenRewardsFee.add(liquidityFee).add(marketingFee);
        require(totalFees <= 25, "Total fee is over 25%");
    }

    function setLiquiditFee(uint256 value) external onlyOwner {
        liquidityFee = value;
        totalFees =
tokenRewardsFee.add(liquidityFee).add(marketingFee);
        require(totalFees <= 25, "Total fee is over 25%");
    }

    function setMarketingFee(uint256 value) external onlyOwner {
        marketingFee = value;
        totalFees =
tokenRewardsFee.add(liquidityFee).add(marketingFee);
        require(totalFees <= 25, "Total fee is over 25%");
    }
```

The owner can't set fees over 25%

5. PinkAntiBot

```
interface IPinkAntiBot {
  function setTokenOwner(address owner) external;

  function onPreTransferCheck(
    address from,
    address to,
    uint256 amount
  ) external;
}
...
 function setEnableAntiBot(bool _enable) external onlyOwner {
        enableAntiBot = _enable;
    }
```

The owner can set antibot to enable or not.

READ CONTRACT (ONLY NEED TO KNOW)

1. Version
1 uint256
(Shows Contract Versions)

2. _marketingWalletAddress
0xe4eb981b21691d6c50ca7b0cbd83a4d020f25519 address
(Shows marketing wallet address)

3. enableAntiBot
True bool
(Function for read anti bot active or not)

4. liquidityFee
2 uint256
(Function for read liquidity fee)

5. marketingFee
2 uint256
(Function for read marketing fee)

6. tokenRewardsFee
2 uint256
(Function for read reward fee)

7. name
Doge Ipa string
(Function for read Token name)

WRITE CONTRACT

1. setEnableAntiBot
_enable (bool)
(The form is filled with the true or false for active or deactivate anti bot)

2. renounceOwnership
(Renouncing ownership will leave the contract without an owner, thereby removing any functionality that is only available to the owner)

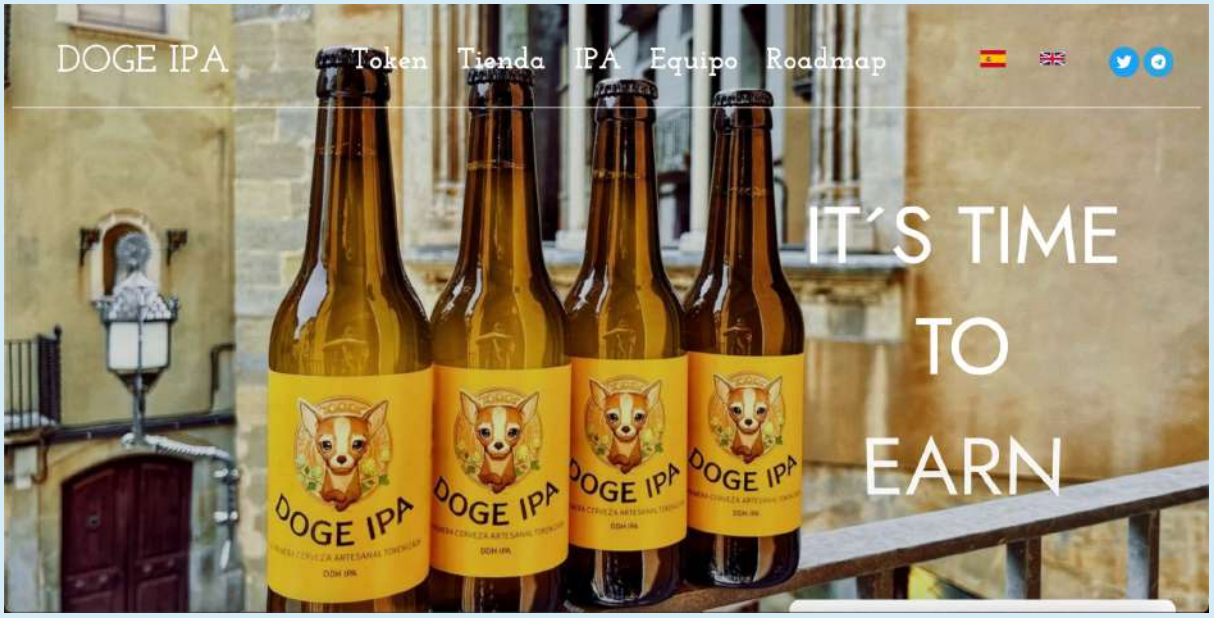
3. transferOwnership
newOwner (address)
(Its function is to change the owner)

4. setLiquiditFee (cannot set over 25%)
value (uint 256)
(The form is filled with new fee, for change liquidity fee)

5. setMarketingFee (cannot set over 25%)
value (uint 256)
(The form is filled with new fee, for change marketing fee)

6. setTokenRewardsFee
value (uint 256)
(The form is filled with new fee, for change Token Rewards fee)

WEBSITE REVIEW



- **Mobile Friendly**
- **Contains no code error**
- **SSL Secured (By R3 SSL)**

Web-Tech stack: Wordpress, PHP, Hostinger

Domain .online (hostinger) - Tracked by whois

First Contentful Paint:	1.0s
Fully Loaded Time	3.0s
Performance	56%
Accessibility	86%
Best Practices	83%
SEO	92%

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 No KYC on Blocksafu

HONEYPOT REVIEW

- Ability to sell.
- The owner is not able to pause the contract.
- The owner can’t set fees over 25%
- PinkAntiBot

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.