/\*\*

\*Submitted for verification at BscScan.com on 2021-09-17

\*/

//Contract by MiniThunder and Team - built on others previous work w/ a splash of DevTeamSix magic, DeFiSCI...

// SPDX-License-Identifier: Unlicensed

pragma solidity ^0.8.4;

abstract contract Context {

function \_msgSender() internal view virtual returns (address payable) {

return payable(msg.sender);

}

function \_msgData() internal view virtual returns (bytes memory) {

this; // silence state mutability warning without generating bytecode - see https://github.com/ethereum/solidity/issues/2691

return msg.data;

}

}

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);

}

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) internal pure returns (uint256) {

return sub(a, b, "SafeMath: subtraction overflow");

}

function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

require(b <= a, errorMessage);

uint256 c = a - b;

return c;

}

function mul(uint256 a, uint256 b) internal pure returns (uint256) {

if (a == 0) {

return 0;

}

uint256 c = a \* b;

require(c / a == b, "SafeMath: multiplication overflow");

return c;

}

function div(uint256 a, uint256 b) internal pure returns (uint256) {

return div(a, b, "SafeMath: division by zero");

}

function div(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

require(b > 0, errorMessage);

uint256 c = a / b;

// assert(a == b \* c + a % b); // There is no case in which this doesn't hold

return c;

}

function mod(uint256 a, uint256 b) internal pure returns (uint256) {

return mod(a, b, "SafeMath: modulo by zero");

}

function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {

require(b != 0, errorMessage);

return a % b;

}

}

library Address {

function isContract(address account) internal view returns (bool) {

// According to EIP-1052, 0x0 is the value returned for not-yet created accounts

// and 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470 is returned

// for accounts without code, i.e. `keccak256('')`

bytes32 codehash;

bytes32 accountHash = 0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;

// solhint-disable-next-line no-inline-assembly

assembly { codehash := extcodehash(account) }

return (codehash != accountHash && codehash != 0x0);

}

function sendValue(address payable recipient, uint256 amount) internal {

require(address(this).balance >= amount, "Address: insufficient balance");

// solhint-disable-next-line avoid-low-level-calls, avoid-call-value

(bool success, ) = recipient.call{ value: amount }("");

require(success, "Address: unable to send value, recipient may have reverted");

}

function functionCall(address target, bytes memory data) internal returns (bytes memory) {

return functionCall(target, data, "Address: low-level call failed");

}

function functionCall(address target, bytes memory data, string memory errorMessage) internal returns (bytes memory) {

return \_functionCallWithValue(target, data, 0, errorMessage);

}

function functionCallWithValue(address target, bytes memory data, uint256 value) internal returns (bytes memory) {

return functionCallWithValue(target, data, value, "Address: low-level call with value failed");

}

function functionCallWithValue(address target, bytes memory data, uint256 value, string memory errorMessage) internal returns (bytes memory) {

require(address(this).balance >= value, "Address: insufficient balance for call");

return \_functionCallWithValue(target, data, value, errorMessage);

}

function \_functionCallWithValue(address target, bytes memory data, uint256 weiValue, string memory errorMessage) private returns (bytes memory) {

require(isContract(target), "Address: call to non-contract");

(bool success, bytes memory returndata) = target.call{ value: weiValue }(data);

if (success) {

return returndata;

} else {

if (returndata.length > 0) {

assembly {

let returndata\_size := mload(returndata)

revert(add(32, returndata), returndata\_size)

}

} else {

revert(errorMessage);

}

}

}

}

contract Ownable is Context {

address private \_owner;

address private \_previousOwner;

uint256 private \_lockTime;

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() public 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;

}

function getUnlockTime() public view returns (uint256) {

return \_lockTime;

}

function getTime() public view returns (uint256) {

return block.timestamp;

}

function lock(uint256 time) public virtual onlyOwner {

\_previousOwner = \_owner;

\_owner = address(0);

\_lockTime = block.timestamp + time;

emit OwnershipTransferred(\_owner, address(0));

}

function unlock() public virtual {

require(\_previousOwner == msg.sender, "You don't have permission to unlock");

require(block.timestamp > \_lockTime , "Contract is locked until 7 days");

emit OwnershipTransferred(\_owner, \_previousOwner);

\_owner = \_previousOwner;

}

}

// pragma solidity >=0.5.0;

interface IUniswapV2Factory {

event PairCreated(address indexed token0, address indexed token1, address pair, uint);

function feeTo() external view returns (address);

function feeToSetter() external view returns (address);

function getPair(address tokenA, address tokenB) external view returns (address pair);

function allPairs(uint) external view returns (address pair);

function allPairsLength() external view returns (uint);

function createPair(address tokenA, address tokenB) external returns (address pair);

function setFeeTo(address) external;

function setFeeToSetter(address) external;

}

// pragma solidity >=0.5.0;

interface IUniswapV2Pair {

event Approval(address indexed owner, address indexed spender, uint value);

event Transfer(address indexed from, address indexed to, uint value);

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

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

function decimals() external pure returns (uint8);

function totalSupply() external view returns (uint);

function balanceOf(address owner) external view returns (uint);

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

function approve(address spender, uint value) external returns (bool);

function transfer(address to, uint value) external returns (bool);

function transferFrom(address from, address to, uint value) external returns (bool);

function DOMAIN\_SEPARATOR() external view returns (bytes32);

function PERMIT\_TYPEHASH() external pure returns (bytes32);

function nonces(address owner) external view returns (uint);

function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external;

event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);

event Swap(

address indexed sender,

uint amount0In,

uint amount1In,

uint amount0Out,

uint amount1Out,

address indexed to

);

event Sync(uint112 reserve0, uint112 reserve1);

function MINIMUM\_LIQUIDITY() external pure returns (uint);

function factory() external view returns (address);

function token0() external view returns (address);

function token1() external view returns (address);

function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);

function price0CumulativeLast() external view returns (uint);

function price1CumulativeLast() external view returns (uint);

function kLast() external view returns (uint);

function burn(address to) external returns (uint amount0, uint amount1);

function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;

function skim(address to) external;

function sync() external;

function initialize(address, address) external;

}

// pragma solidity >=0.6.2;

interface IUniswapV2Router01 {

function factory() external pure returns (address);

function WETH() external pure returns (address);

function addLiquidity(

address tokenA,

address tokenB,

uint amountADesired,

uint amountBDesired,

uint amountAMin,

uint amountBMin,

address to,

uint deadline

) external returns (uint amountA, uint amountB, uint liquidity);

function addLiquidityETH(

address token,

uint amountTokenDesired,

uint amountTokenMin,

uint amountETHMin,

address to,

uint deadline

) external payable returns (uint amountToken, uint amountETH, uint liquidity);

function removeLiquidity(

address tokenA,

address tokenB,

uint liquidity,

uint amountAMin,

uint amountBMin,

address to,

uint deadline

) external returns (uint amountA, uint amountB);

function removeLiquidityETH(

address token,

uint liquidity,

uint amountTokenMin,

uint amountETHMin,

address to,

uint deadline

) external returns (uint amountToken, uint amountETH);

function removeLiquidityWithPermit(

address tokenA,

address tokenB,

uint liquidity,

uint amountAMin,

uint amountBMin,

address to,

uint deadline,

bool approveMax, uint8 v, bytes32 r, bytes32 s

) external returns (uint amountA, uint amountB);

function removeLiquidityETHWithPermit(

address token,

uint liquidity,

uint amountTokenMin,

uint amountETHMin,

address to,

uint deadline,

bool approveMax, uint8 v, bytes32 r, bytes32 s

) external returns (uint amountToken, uint amountETH);

function swapExactTokensForTokens(

uint amountIn,

uint amountOutMin,

address[] calldata path,

address to,

uint deadline

) external returns (uint[] memory amounts);

function swapTokensForExactTokens(

uint amountOut,

uint amountInMax,

address[] calldata path,

address to,

uint deadline

) external returns (uint[] memory amounts);

function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)

external

payable

returns (uint[] memory amounts);

function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to, uint deadline)

external

returns (uint[] memory amounts);

function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)

external

returns (uint[] memory amounts);

function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)

external

payable

returns (uint[] memory amounts);

function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);

function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint amountOut);

function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint amountIn);

function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memory amounts);

function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memory amounts);

}

// pragma solidity >=0.6.2;

interface IUniswapV2Router02 is IUniswapV2Router01 {

function removeLiquidityETHSupportingFeeOnTransferTokens(

address token,

uint liquidity,

uint amountTokenMin,

uint amountETHMin,

address to,

uint deadline

) external returns (uint amountETH);

function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(

address token,

uint liquidity,

uint amountTokenMin,

uint amountETHMin,

address to,

uint deadline,

bool approveMax, uint8 v, bytes32 r, bytes32 s

) external returns (uint amountETH);

function swapExactTokensForTokensSupportingFeeOnTransferTokens(

uint amountIn,

uint amountOutMin,

address[] calldata path,

address to,

uint deadline

) external;

function swapExactETHForTokensSupportingFeeOnTransferTokens(

uint amountOutMin,

address[] calldata path,

address to,

uint deadline

) external payable;

function swapExactTokensForETHSupportingFeeOnTransferTokens(

uint amountIn,

uint amountOutMin,

address[] calldata path,

address to,

uint deadline

) external;

}

contract MiniThunder is Context, IERC20, Ownable {

using SafeMath for uint256;

using Address for address;

address payable public marketingAddress = payable(0x6628979fF520C4c770722567CC14028fdAD5F5BB); // Marketing Address

address public immutable deadAddress = 0x000000000000000000000000000000000000dEaD;

mapping (address => uint256) private \_rOwned;

mapping (address => uint256) private \_tOwned;

mapping (address => mapping (address => uint256)) private \_allowances;

mapping (address => bool) private \_isExcludedFromFee;

mapping (address => bool) private \_isExcluded;

address[] private \_excluded;

uint256 private constant MAX = ~uint256(0);

uint256 private \_tTotal = 1000000000 \* 10\*\*6 \* 10\*\*9;

uint256 private \_rTotal = (MAX - (MAX % \_tTotal));

uint256 private \_tFeeTotal;

string private \_name = "MiniThunder";

string private \_symbol = "Thunder";

uint8 private \_decimals = 9;

struct AddressFee {

bool enable;

uint256 \_taxFee;

uint256 \_liquidityFee;

uint256 \_buyTaxFee;

uint256 \_buyLiquidityFee;

uint256 \_sellTaxFee;

uint256 \_sellLiquidityFee;

}

struct SellHistories {

uint256 time;

uint256 bnbAmount;

}

uint256 public \_taxFee = 2;

uint256 private \_previousTaxFee = \_taxFee;

uint256 public \_liquidityFee = 10;

uint256 private \_previousLiquidityFee = \_liquidityFee;

uint256 public \_buyTaxFee = 2;

uint256 public \_buyLiquidityFee = 10;

uint256 public \_sellTaxFee = 7;

uint256 public \_sellLiquidityFee = 11;

uint256 public \_startTimeForSwap;

uint256 public \_intervalMinutesForSwap = 1 \* 1 minutes;

uint256 public \_buyBackRangeRate = 80;

// Fee per address

mapping (address => AddressFee) public \_addressFees;

uint256 public marketingDivisor = 4;

uint256 public \_maxTxAmount = 3000000 \* 10\*\*6 \* 10\*\*9;

uint256 private minimumTokensBeforeSwap = 200000 \* 10\*\*6 \* 10\*\*9;

uint256 public buyBackSellLimit = 1 \* 10\*\*14;

// LookBack into historical sale data

SellHistories[] public \_sellHistories;

bool public \_isAutoBuyBack = true;

uint256 public \_buyBackDivisor = 10;

uint256 public \_buyBackTimeInterval = 5 minutes;

uint256 public \_buyBackMaxTimeForHistories = 24 \* 60 minutes;

IUniswapV2Router02 public uniswapV2Router;

address public uniswapV2Pair;

bool inSwapAndLiquify;

bool public swapAndLiquifyEnabled = false;

bool public buyBackEnabled = true;

bool public \_isEnabledBuyBackAndBurn = true;

event RewardLiquidityProviders(uint256 tokenAmount);

event BuyBackEnabledUpdated(bool enabled);

event AutoBuyBackEnabledUpdated(bool enabled);

event SwapAndLiquifyEnabledUpdated(bool enabled);

event SwapAndLiquify(

uint256 tokensSwapped,

uint256 ethReceived,

uint256 tokensIntoLiqudity

);

event SwapETHForTokens(

uint256 amountIn,

address[] path

);

event SwapTokensForETH(

uint256 amountIn,

address[] path

);

modifier lockTheSwap {

inSwapAndLiquify = true;

\_;

inSwapAndLiquify = false;

}

constructor () {

\_rOwned[\_msgSender()] = \_rTotal;

// Pancake Router Testnet v1

// IUniswapV2Router02 \_uniswapV2Router = IUniswapV2Router02(0xD99D1c33F9fC3444f8101754aBC46c52416550D1);

// uniswap Router Testnet v2 - Not existing I guess

IUniswapV2Router02 \_uniswapV2Router = IUniswapV2Router02(0x10ED43C718714eb63d5aA57B78B54704E256024E);

uniswapV2Pair = IUniswapV2Factory(\_uniswapV2Router.factory())

.createPair(address(this), \_uniswapV2Router.WETH());

uniswapV2Router = \_uniswapV2Router;

\_isExcludedFromFee[owner()] = true;

\_isExcludedFromFee[address(this)] = true;

\_startTimeForSwap = block.timestamp;

emit Transfer(address(0), \_msgSender(), \_tTotal);

}

function name() public view returns (string memory) {

return \_name;

}

function symbol() public view returns (string memory) {

return \_symbol;

}

function decimals() public view returns (uint8) {

return \_decimals;

}

function totalSupply() public view override returns (uint256) {

return \_tTotal;

}

function balanceOf(address account) public view override returns (uint256) {

if (\_isExcluded[account]) return \_tOwned[account];

return tokenFromReflection(\_rOwned[account]);

}

function transfer(address recipient, uint256 amount) public override returns (bool) {

\_transfer(\_msgSender(), recipient, amount);

return true;

}

function allowance(address owner, address spender) public view override returns (uint256) {

return \_allowances[owner][spender];

}

function approve(address spender, uint256 amount) public override returns (bool) {

\_approve(\_msgSender(), spender, amount);

return true;

}

function transferFrom(address sender, address recipient, uint256 amount) public override returns (bool) {

\_transfer(sender, recipient, amount);

\_approve(sender, \_msgSender(), \_allowances[sender][\_msgSender()].sub(amount, "ERC20: transfer amount exceeds allowance"));

return true;

}

function increaseAllowance(address spender, uint256 addedValue) public virtual returns (bool) {

\_approve(\_msgSender(), spender, \_allowances[\_msgSender()][spender].add(addedValue));

return true;

}

function decreaseAllowance(address spender, uint256 subtractedValue) public virtual returns (bool) {

\_approve(\_msgSender(), spender, \_allowances[\_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below zero"));

return true;

}

function isExcludedFromReward(address account) public view returns (bool) {

return \_isExcluded[account];

}

function totalFees() public view returns (uint256) {

return \_tFeeTotal;

}

function minimumTokensBeforeSwapAmount() public view returns (uint256) {

return minimumTokensBeforeSwap;

}

function buyBackSellLimitAmount() public view returns (uint256) {

return buyBackSellLimit;

}

function deliver(uint256 tAmount) public {

address sender = \_msgSender();

require(!\_isExcluded[sender], "Excluded addresses cannot call this function");

(uint256 rAmount,,,,,) = \_getValues(tAmount);

\_rOwned[sender] = \_rOwned[sender].sub(rAmount);

\_rTotal = \_rTotal.sub(rAmount);

\_tFeeTotal = \_tFeeTotal.add(tAmount);

}

function reflectionFromToken(uint256 tAmount, bool deductTransferFee) public view returns(uint256) {

require(tAmount <= \_tTotal, "Amount must be less than supply");

if (!deductTransferFee) {

(uint256 rAmount,,,,,) = \_getValues(tAmount);

return rAmount;

} else {

(,uint256 rTransferAmount,,,,) = \_getValues(tAmount);

return rTransferAmount;

}

}

function tokenFromReflection(uint256 rAmount) public view returns(uint256) {

require(rAmount <= \_rTotal, "Amount must be less than total reflections");

uint256 currentRate = \_getRate();

return rAmount.div(currentRate);

}

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 not excluded");

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;

}

}

}

function \_approve(address owner, address spender, uint256 amount) private {

require(owner != address(0), "ERC20: approve from the zero address");

require(spender != address(0), "ERC20: approve to the zero address");

\_allowances[owner][spender] = amount;

emit Approval(owner, spender, amount);

}

function \_transfer(

address from,

address to,

uint256 amount

) private {

require(from != address(0), "ERC20: transfer from the zero address");

require(to != address(0), "ERC20: transfer to the zero address");

require(amount > 0, "Transfer amount must be greater than zero");

if(from != owner() && to != owner()) {

require(amount <= \_maxTxAmount, "Transfer amount exceeds the maxTxAmount.");

}

uint256 contractTokenBalance = balanceOf(address(this));

bool overMinimumTokenBalance = contractTokenBalance >= minimumTokensBeforeSwap;

if (to == uniswapV2Pair && balanceOf(uniswapV2Pair) > 0) {

SellHistories memory sellHistory;

sellHistory.time = block.timestamp;

sellHistory.bnbAmount = \_getSellBnBAmount(amount);

\_sellHistories.push(sellHistory);

}

// Sell tokens for ETH

if (!inSwapAndLiquify && swapAndLiquifyEnabled && balanceOf(uniswapV2Pair) > 0) {

if (to == uniswapV2Pair) {

if (overMinimumTokenBalance && \_startTimeForSwap + \_intervalMinutesForSwap <= block.timestamp) {

\_startTimeForSwap = block.timestamp;

contractTokenBalance = minimumTokensBeforeSwap;

swapTokens(contractTokenBalance);

}

if (buyBackEnabled) {

uint256 balance = address(this).balance;

uint256 \_bBSLimitMax = buyBackSellLimit;

if (\_isAutoBuyBack) {

uint256 sumBnbAmount = 0;

uint256 startTime = block.timestamp - \_buyBackTimeInterval;

uint256 cnt = 0;

for (uint i = 0; i < \_sellHistories.length; i ++) {

if (\_sellHistories[i].time >= startTime) {

sumBnbAmount = sumBnbAmount.add(\_sellHistories[i].bnbAmount);

cnt = cnt + 1;

}

}

if (cnt > 0 && \_buyBackDivisor > 0) {

\_bBSLimitMax = sumBnbAmount.div(cnt).div(\_buyBackDivisor);

}

\_removeOldSellHistories();

}

uint256 \_bBSLimitMin = \_bBSLimitMax.mul(\_buyBackRangeRate).div(100);

uint256 \_bBSLimit = \_bBSLimitMin + uint256(keccak256(abi.encodePacked(block.timestamp, block.difficulty))) % (\_bBSLimitMax - \_bBSLimitMin + 1);

if (balance > \_bBSLimit) {

buyBackTokens(\_bBSLimit);

}

}

}

}

bool takeFee = true;

// If any account belongs to \_isExcludedFromFee account then remove the fee

if(\_isExcludedFromFee[from] || \_isExcludedFromFee[to]){

takeFee = false;

}

else{

// Buy

if(from == uniswapV2Pair){

removeAllFee();

\_taxFee = \_buyTaxFee;

\_liquidityFee = \_buyLiquidityFee;

}

// Sell

if(to == uniswapV2Pair){

removeAllFee();

\_taxFee = \_sellTaxFee;

\_liquidityFee = \_sellLiquidityFee;

}

// If send account has a special fee

if(\_addressFees[from].enable){

removeAllFee();

\_taxFee = \_addressFees[from].\_taxFee;

\_liquidityFee = \_addressFees[from].\_liquidityFee;

// Sell

if(to == uniswapV2Pair){

\_taxFee = \_addressFees[from].\_sellTaxFee;

\_liquidityFee = \_addressFees[from].\_sellLiquidityFee;

}

}

else{

// If buy account has a special fee

if(\_addressFees[to].enable){

//buy

removeAllFee();

if(from == uniswapV2Pair){

\_taxFee = \_addressFees[to].\_buyTaxFee;

\_liquidityFee = \_addressFees[to].\_buyLiquidityFee;

}

}

}

}

\_tokenTransfer(from,to,amount,takeFee);

}

function swapTokens(uint256 contractTokenBalance) private lockTheSwap {

uint256 initialBalance = address(this).balance;

swapTokensForEth(contractTokenBalance);

uint256 transferredBalance = address(this).balance.sub(initialBalance);

// Send to Marketing address//50%

transferToAddressETH(marketingAddress, transferredBalance.mul(50).div(100));

}

function buyBackTokens(uint256 amount) private lockTheSwap {

if (amount > 0) {

swapETHForTokens(amount);

}

}

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), // The contract

block.timestamp

);

emit SwapTokensForETH(tokenAmount, path);

}

function swapETHForTokens(uint256 amount) private {

// Generate the uniswap pair path of token -> WETH

address[] memory path = new address[](2);

path[0] = uniswapV2Router.WETH();

path[1] = address(this);

// Make the swap

uniswapV2Router.swapExactETHForTokensSupportingFeeOnTransferTokens{value: amount}(

0, // Accept any amount of Tokens

path,

deadAddress, // Burn address

block.timestamp.add(300)

);

emit SwapETHForTokens(amount, path);

}

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

owner(),

block.timestamp

);

}

function \_tokenTransfer(address sender, address recipient, uint256 amount,bool takeFee) private {

if(!takeFee)

removeAllFee();

if (\_isExcluded[sender] && !\_isExcluded[recipient]) {

\_transferFromExcluded(sender, recipient, amount);

} else if (!\_isExcluded[sender] && \_isExcluded[recipient]) {

\_transferToExcluded(sender, recipient, amount);

} else if (\_isExcluded[sender] && \_isExcluded[recipient]) {

\_transferBothExcluded(sender, recipient, amount);

} else {

\_transferStandard(sender, recipient, amount);

}

if(!takeFee)

restoreAllFee();

}

function \_transferStandard(address sender, address recipient, uint256 tAmount) private {

(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = \_getValues(tAmount);

\_rOwned[sender] = \_rOwned[sender].sub(rAmount);

\_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount);

\_takeLiquidity(tLiquidity);

\_reflectFee(rFee, tFee);

emit Transfer(sender, recipient, tTransferAmount);

}

function \_transferToExcluded(address sender, address recipient, uint256 tAmount) private {

(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = \_getValues(tAmount);

\_rOwned[sender] = \_rOwned[sender].sub(rAmount);

\_tOwned[recipient] = \_tOwned[recipient].add(tTransferAmount);

\_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount);

\_takeLiquidity(tLiquidity);

\_reflectFee(rFee, tFee);

emit Transfer(sender, recipient, tTransferAmount);

}

function \_transferFromExcluded(address sender, address recipient, uint256 tAmount) private {

(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = \_getValues(tAmount);

\_tOwned[sender] = \_tOwned[sender].sub(tAmount);

\_rOwned[sender] = \_rOwned[sender].sub(rAmount);

\_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount);

\_takeLiquidity(tLiquidity);

\_reflectFee(rFee, tFee);

emit Transfer(sender, recipient, tTransferAmount);

}

function \_transferBothExcluded(address sender, address recipient, uint256 tAmount) private {

(uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = \_getValues(tAmount);

\_tOwned[sender] = \_tOwned[sender].sub(tAmount);

\_rOwned[sender] = \_rOwned[sender].sub(rAmount);

\_tOwned[recipient] = \_tOwned[recipient].add(tTransferAmount);

\_rOwned[recipient] = \_rOwned[recipient].add(rTransferAmount);

\_takeLiquidity(tLiquidity);

\_reflectFee(rFee, tFee);

emit Transfer(sender, recipient, tTransferAmount);

}

function \_reflectFee(uint256 rFee, uint256 tFee) private {

\_rTotal = \_rTotal.sub(rFee);

\_tFeeTotal = \_tFeeTotal.add(tFee);

}

function \_getValues(uint256 tAmount) private view returns (uint256, uint256, uint256, uint256, uint256, uint256) {

(uint256 tTransferAmount, uint256 tFee, uint256 tLiquidity) = \_getTValues(tAmount);

(uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = \_getRValues(tAmount, tFee, tLiquidity, \_getRate());

return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee, tLiquidity);

}

function \_getTValues(uint256 tAmount) private view returns (uint256, uint256, uint256) {

uint256 tFee = calculateTaxFee(tAmount);

uint256 tLiquidity = calculateLiquidityFee(tAmount);

uint256 tTransferAmount = tAmount.sub(tFee).sub(tLiquidity);

return (tTransferAmount, tFee, tLiquidity);

}

function \_getRValues(uint256 tAmount, uint256 tFee, uint256 tLiquidity, uint256 currentRate) private pure returns (uint256, uint256, uint256) {

uint256 rAmount = tAmount.mul(currentRate);

uint256 rFee = tFee.mul(currentRate);

uint256 rLiquidity = tLiquidity.mul(currentRate);

uint256 rTransferAmount = rAmount.sub(rFee).sub(rLiquidity);

return (rAmount, rTransferAmount, rFee);

}

function \_getRate() private view returns(uint256) {

(uint256 rSupply, uint256 tSupply) = \_getCurrentSupply();

return rSupply.div(tSupply);

}

function \_getCurrentSupply() private view returns(uint256, uint256) {

uint256 rSupply = \_rTotal;

uint256 tSupply = \_tTotal;

for (uint256 i = 0; i < \_excluded.length; i++) {

if (\_rOwned[\_excluded[i]] > rSupply || \_tOwned[\_excluded[i]] > tSupply) return (\_rTotal, \_tTotal);

rSupply = rSupply.sub(\_rOwned[\_excluded[i]]);

tSupply = tSupply.sub(\_tOwned[\_excluded[i]]);

}

if (rSupply < \_rTotal.div(\_tTotal)) return (\_rTotal, \_tTotal);

return (rSupply, tSupply);

}

function \_takeLiquidity(uint256 tLiquidity) private {

uint256 currentRate = \_getRate();

uint256 rLiquidity = tLiquidity.mul(currentRate);

\_rOwned[address(this)] = \_rOwned[address(this)].add(rLiquidity);

if(\_isExcluded[address(this)])

\_tOwned[address(this)] = \_tOwned[address(this)].add(tLiquidity);

}

function calculateTaxFee(uint256 \_amount) private view returns (uint256) {

return \_amount.mul(\_taxFee).div(

10\*\*2

);

}

function calculateLiquidityFee(uint256 \_amount) private view returns (uint256) {

return \_amount.mul(\_liquidityFee).div(

10\*\*2

);

}

function removeAllFee() private {

if(\_taxFee == 0 && \_liquidityFee == 0) return;

\_previousTaxFee = \_taxFee;

\_previousLiquidityFee = \_liquidityFee;

\_taxFee = 0;

\_liquidityFee = 0;

}

function restoreAllFee() private {

\_taxFee = \_previousTaxFee;

\_liquidityFee = \_previousLiquidityFee;

}

function isExcludedFromFee(address account) public view returns(bool) {

return \_isExcludedFromFee[account];

}

function excludeFromFee(address account) public onlyOwner {

\_isExcludedFromFee[account] = true;

}

function includeInFee(address account) public onlyOwner {

\_isExcludedFromFee[account] = false;

}

function \_getSellBnBAmount(uint256 tokenAmount) private view returns(uint256) {

address[] memory path = new address[](2);

path[0] = address(this);

path[1] = uniswapV2Router.WETH();

uint[] memory amounts = uniswapV2Router.getAmountsOut(tokenAmount, path);

return amounts[1];

}

function \_removeOldSellHistories() private {

uint256 i = 0;

uint256 maxStartTimeForHistories = block.timestamp - \_buyBackMaxTimeForHistories;

for (uint256 j = 0; j < \_sellHistories.length; j ++) {

if (\_sellHistories[j].time >= maxStartTimeForHistories) {

\_sellHistories[i].time = \_sellHistories[j].time;

\_sellHistories[i].bnbAmount = \_sellHistories[j].bnbAmount;

i = i + 1;

}

}

uint256 removedCnt = \_sellHistories.length - i;

for (uint256 j = 0; j < removedCnt; j ++) {

\_sellHistories.pop();

}

}

function SetBuyBackMaxTimeForHistories(uint256 newMinutes) external onlyOwner {

\_buyBackMaxTimeForHistories = newMinutes \* 1 minutes;

}

function SetBuyBackDivisor(uint256 newDivisor) external onlyOwner {

\_buyBackDivisor = newDivisor;

}

function GetBuyBackTimeInterval() public view returns(uint256) {

return \_buyBackTimeInterval.div(60);

}

function SetBuyBackTimeInterval(uint256 newMinutes) external onlyOwner {

\_buyBackTimeInterval = newMinutes \* 1 minutes;

}

function SetBuyBackRangeRate(uint256 newPercent) external onlyOwner {

require(newPercent <= 100, "The value must not be larger than 100.");

\_buyBackRangeRate = newPercent;

}

function GetSwapMinutes() public view returns(uint256) {

return \_intervalMinutesForSwap.div(60);

}

function SetSwapMinutes(uint256 newMinutes) external onlyOwner {

\_intervalMinutesForSwap = newMinutes \* 1 minutes;

}

function setTaxFeePercent(uint256 taxFee) external onlyOwner() {

\_taxFee = taxFee;

}

function setBuyFee(uint256 buyTaxFee, uint256 buyLiquidityFee) external onlyOwner {

\_buyTaxFee = buyTaxFee;

\_buyLiquidityFee = buyLiquidityFee;

}

function setSellFee(uint256 sellTaxFee, uint256 sellLiquidityFee) external onlyOwner {

\_sellTaxFee = sellTaxFee;

\_sellLiquidityFee = sellLiquidityFee;

}

function setLiquidityFeePercent(uint256 liquidityFee) external onlyOwner {

\_liquidityFee = liquidityFee;

}

function setBuyBackSellLimit(uint256 buyBackSellSetLimit) external onlyOwner {

buyBackSellLimit = buyBackSellSetLimit;

}

function setMaxTxAmount(uint256 maxTxAmount) external onlyOwner {

\_maxTxAmount = maxTxAmount;

}

function setMarketingDivisor(uint256 divisor) external onlyOwner {

marketingDivisor = divisor;

}

function setNumTokensSellToAddToBuyBack(uint256 \_minimumTokensBeforeSwap) external onlyOwner {

minimumTokensBeforeSwap = \_minimumTokensBeforeSwap;

}

function setMarketingAddress(address \_marketingAddress) external onlyOwner {

marketingAddress = payable(\_marketingAddress);

}

function setSwapAndLiquifyEnabled(bool \_enabled) public onlyOwner {

swapAndLiquifyEnabled = \_enabled;

emit SwapAndLiquifyEnabledUpdated(\_enabled);

}

function setBuyBackEnabled(bool \_enabled) public onlyOwner {

buyBackEnabled = \_enabled;

emit BuyBackEnabledUpdated(\_enabled);

}

function setAutoBuyBackEnabled(bool \_enabled) public onlyOwner {

\_isAutoBuyBack = \_enabled;

emit AutoBuyBackEnabledUpdated(\_enabled);

}

function prepareForPreSale() external onlyOwner {

setSwapAndLiquifyEnabled(false);

\_taxFee = 0;

\_liquidityFee = 0;

\_maxTxAmount = 1000000000 \* 10\*\*6 \* 10\*\*9;

}

function afterPreSale() external onlyOwner {

setSwapAndLiquifyEnabled(true);

\_taxFee = 2;

\_liquidityFee = 10;

\_maxTxAmount = 3000000 \* 10\*\*6 \* 10\*\*9;

}

function transferToAddressETH(address payable recipient, uint256 amount) private {

recipient.transfer(amount);

}

function changeRouterVersion(address \_router) public onlyOwner returns(address \_pair) {

IUniswapV2Router02 \_uniswapV2Router = IUniswapV2Router02(\_router);

\_pair = IUniswapV2Factory(\_uniswapV2Router.factory()).getPair(address(this), \_uniswapV2Router.WETH());

if(\_pair == address(0)){

// Pair doesn't exist

\_pair = IUniswapV2Factory(\_uniswapV2Router.factory())

.createPair(address(this), \_uniswapV2Router.WETH());

}

uniswapV2Pair = \_pair;

// Set the router of the contract variables

uniswapV2Router = \_uniswapV2Router;

}

// To recieve ETH from uniswapV2Router when swapping

receive() external payable {}

function transferForeignToken(address \_token, address \_to) public onlyOwner returns(bool \_sent){

require(\_token != address(this), "Can't let you take all native token");

uint256 \_contractBalance = IERC20(\_token).balanceOf(address(this));

\_sent = IERC20(\_token).transfer(\_to, \_contractBalance);

}

function Sweep() external onlyOwner {

uint256 balance = address(this).balance;

payable(owner()).transfer(balance);

}

function setAddressFee(address \_address, bool \_enable, uint256 \_addressTaxFee, uint256 \_addressLiquidityFee) external onlyOwner {

\_addressFees[\_address].enable = \_enable;

\_addressFees[\_address].\_taxFee = \_addressTaxFee;

\_addressFees[\_address].\_liquidityFee = \_addressLiquidityFee;

}

function setBuyAddressFee(address \_address, bool \_enable, uint256 \_addressTaxFee, uint256 \_addressLiquidityFee) external onlyOwner {

\_addressFees[\_address].enable = \_enable;

\_addressFees[\_address].\_buyTaxFee = \_addressTaxFee;

\_addressFees[\_address].\_buyLiquidityFee = \_addressLiquidityFee;

}

function setSellAddressFee(address \_address, bool \_enable, uint256 \_addressTaxFee, uint256 \_addressLiquidityFee) external onlyOwner {

\_addressFees[\_address].enable = \_enable;

\_addressFees[\_address].\_sellTaxFee = \_addressTaxFee;

\_addressFees[\_address].\_sellLiquidityFee = \_addressLiquidityFee;

}

}