Coding

(100) Wells-read

We were given two text files:

- "The Time Machine by H. G. Wells.txt”, containing the book specified in the file name, renamed to book.txt for convenience
- words.txt, which contained the list of the english words

Upon closer inspection, we can notice that some letters in the book.txt file were replaced with wrong ones. Therefore our task is to find the misspelled words (probably by using the words.txt file) and extract the wrong characters.

The book.txt contained many correct words, which weren't present in the words.txt. Instead, inspired by the last year's ctf, we went to the gutenberg.org and downloaded the utf-8 version of the same book, which turned out to have exactly the same formatting, with the only difference being, that the book.txt has replaced the non-ascii special characters with their ascii counterparts.

After solving this issue, we can run a simple byte by byte diff to get the following result:

```
eGEteGOZyIy:otMeUAqDYXVu}gHjNp54E6qqM{BFD:5LHXJoYpwvjthF_BpSp7Kd}PC}q5vxfSMC97yllwR74l
Xl{b5SYMLY5{VVt_Z4{q{RwdP:pNfv4ApnK43XSl2:FF23q1hHCwGKQ{sjppwhjiyBXy85tp0eYKM3DZvPUwkl
B1IwMFM5JDm7cyENCR2r{iDDNqits9rmMs1skCxHyDaBb68gkPK{V0x1BPieEU0H{gUV2_t9SHGJsIbxIvE{HI
oA9Fw0p0ZmvpPmZu5b}I5UEeH9dSeqp:Vdw_otv1Ouu_LK8{6G_V2o8ga}xj0T29VlC5aO0wmU8Ph{5WxJ0M4F
Cs5JsQq:jk38vzR3RFroDXvfBq{EYK{5dIEARz{6IV34iVIF98aqb4aa{Av:9mYAAE8gv}uOii2GQoeYYoxbWS
101G3dyIV:A63c}fO{BMH8WFU_b5j{je61W{HkGaoqmgkeyl{5Lf{7lRqNPvl9aXp4vpiw6x38}kOGjcoOMFtQ
LM847e}uGU_}s:lqrs:M{0K_baDC:Rku197CAVdq_r_TDOKDNqLNJ8b_{PLQyD:FM:kb{}zcK6vajYFVKlaebq
2zc{xYp_}wqLlVV6odWO7I}252f1J{H}BOQ7h5{s4OjoZAAhjY{FLG:1_kn0w_3v3ryth1ng_4b0ut_t1m3_tr
4v3ls}0:w8Wj_SJeGS_og{6k{6NshxTn:INYsHC0FHNmiz{:jpXQB28ZksKq{rsEOFzkF}TaYFX_6RZ:MA83{T
jL04zOn{Onxaldtlw6Aquq89K_1ofz1QzQcLje8vV{Xlz3xv6:BFU1TCwO9qVof2kg{s6v4jSXLJSHauo8y:06
CMlaS:h:UBx4BCKVLj}gRrYoPSAAH4tz6Y1cSAvL:Dgatm9IyGc2u1j2lJEJoT{uMvU2X5xomiEN9CkJB4o_zR
ajzXsalz}BD5CRFz2Dw6oVPikFrfSQazV2DzeHvGJ{9{aQJW}Fd101ugfn:EGp8{W_Tbrvacq{GHT:MheEVmxX
YQrxrDLVLmzut{aLIva4780ZBJ9YQkTMr}5T}I2aeJAxVyqHdeQegTUfEU0MUwOfZT{8:jD6h{MIb_8m:6qSG{
YrkLsi3FjwgPbhBTs_567v:QaYF:TpThGe5fTfYSXu2eKUHdEeR}1xE0Mscpfhap23zTkG{6_5q7UV:w2__NBp
dILP4nYDhq9X3akDC6LyLsHS0iDD9Lw5ZJF3Ic}K41jn3b6V5zykM134DzlyIpb:Y_xyfvlW30C0NMqDFmt1{O
O:vqoTvtaDtw5rjKI_oel_HW5G0qCpF2qbwyiKNCMT3KSmI:bat9:jC7FSN3}ef6TvCLJ0hKC5X6oBMjBFMl0a
Iqoeghx5_8yZvOi8zZi0ZT0IK7BVcl9q3dfQbXQs{Yl6vyH5pTkWGJ}}111qLxZLyx2dpw{4jm495rBJQQ5R2{
_hAG{ZI2tvp4:x4XuKeY2tb00{bKjaigNBl_2_tniQA6DA9KUb_{w5Y3uSihDQSgffi_UmqtS4h6:_AX{DrG6V
o4DTfNjkIUw516}QBOcuXqlb8kpxw1WsMsrc{GA}w}G}T7nmOxxa7QT5OGd6G_u_GGXjklQafi_kaAiZ_s:{jG
HaKjpk{VU5Kj_klqTzKMHPag8vgj05lNNB:yj{kPOkBhUxZRu:G8gRzfacsqVgLI101mg
```

We can find a flag hidden in this text: `{FLG:1_kn0w_3v3ryth1ng_4b0ut_t1m3_tr4v3ls}`

(200) Limbo Zone -?> Limbo Z0ne

We were given a huge archive `level_0.7z` containing the following files:

- password-protected archive `level_1.7z`
- script bruteforcing this password
- two images `level_0.png` and `lev3l_0.png`

The passwords tested by the script have the `{x}{y}{hexcolor1}{hexcolor2}` format. Performing a pixel-by-pixel diff on both images, we get that they differ at exactly one pixel. Taking this pixel's coordinate along with its colors in both images, we can unpack the `level_1.7z` archive, which contains another set of images and archive called `level_2.7z`.

It's clear that we have to automate this process. After several levels, the task gets slightly harder because one of the images is flipped verticaly. Moreover, the following levels contain different image transformations.

Therefore, we first generate a list consisting of the first image modified by different transforms (we chose all that were available in python's PIL library, which turned out to be enough), then perform pixel by pixel comparison with every generated image and select the one which differs at exactly 1 pixel.

After unpacking 1024 archives and filling up all of our available disk space, we end up with the text file containing the flag:
`{FLG:p1xel0ut0fBound3xcept1on_tr4p_1s_shutt1ng_d0wn}`

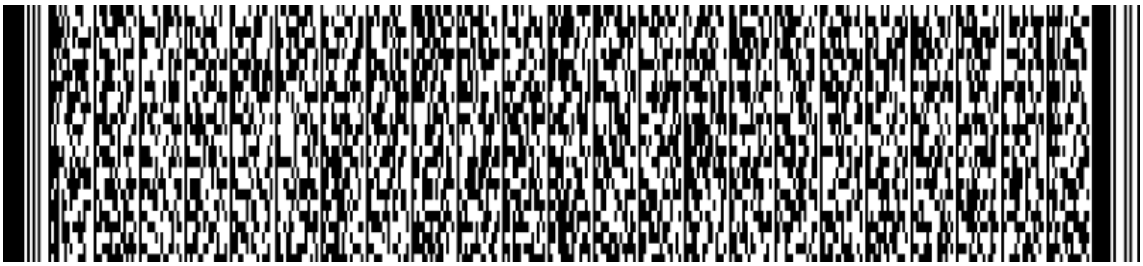
The final script:

```
from PIL import Image
import py7zr
from random import shuffle
import os

def tryOpen(x, y, c1, c2, lvl):
    password = format(x, y, c1, c2)
    print(password)
    try:
        with py7zr.SevenZipFile('level_' + str(lvl) + '/level_' + str(lvl + 1) +
'.7z', mode='r', password=password) as z:
            z.extractall(path='./level_' + str(lvl + 1))
    except:
        print("Nope")

def format(x, y, t1, t2):
    r1, g1, b1 = t1
    r2, g2, b2 = t2
    xy = str(x) + str(y)
    rgb1 = '{:0{}X}'.format(r1, 2) + '{:0{}X}'.format(g1, 2) + '{:0{}X}'.format(b1, 2)
    rgb2 = '{:0{}X}'.format(r2, 2) + '{:0{}X}'.format(g2, 2) + '{:0{}X}'.format(b2, 2)
    return xy + rgb1 + rgb2

def diff(im1, im2):
    if im1.size != im2.size:
        return []
```

```
    im1_l = im1.load()
    im2_l = im2.load()
    xs, ys = im1.size
    ret = []
    for x in range(xs):
        for y in range(ys):
            if im1_l[x, y] != im2_l[x, y]:
                ret.append((x, y))
    return ret


lvl = 0

while True:
    im1 = Image.open('level_' + str(lvl) + '/level_' + str(lvl) + '.png')
    im2 = Image.open('level_' + str(lvl) + '/lev3l_' + str(lvl) + '.png')
    os.system('cp level_' + str(lvl) + '/level_' + str(lvl) + '.png images/')

    im2_0 = im2
    im2_1 = im2.transpose(method=Image.FLIP_LEFT_RIGHT)
    im2_2 = im2.transpose(method=Image.FLIP_TOP_BOTTOM)
    im2_3 = im2.transpose(method=Image.ROTATE_90)
    im2_4 = im2.transpose(method=Image.ROTATE_180)
    im2_5 = im2.transpose(method=Image.ROTATE_270)
    im2_6 = im2.transpose(method=Image.TRANSPOSE)
    im2_7 = im2.transpose(method=Image.TRANSVERSE)

    im2_list = [im2_0, im2_1, im2_2, im2_3, im2_4, im2_5, im2_6, im2_7]
    shuffle(im2_list)
    #some images were symmetric, which ended up crashing this solver, hence the
shuffle

    for im2 in im2_list:
        d = diff(im1, im2)
        if len(d) == 1:
            x = d[0][0]
            y = d[0][1]
            c1 = im1.load()[x, y]
            c2 = im2.load()[x, y]
            tryOpen(x, y, c1, c2, lvl)
            break
    lvl += 1
```

(300) Hide & eXec

We are given an archive, which contains another password-protected archive and the following image:



The image contains a PDF417 barcode. After decoding the barcode using online tools, we end up with a php code snippet, which evaluates to the string `5ymCQoqt3wnOecebYkfvaUK7gSread73`, which turns out to be the password to the second zip archive.

After unpacking the archive, we end up with another archive and another image. This time, the image is a QR code and it decodes to the code snippet written in python. The same story repeats itself many times, with new formats of 1d/2d codes and code snippets written in new languages.

To decode the barcodes we used the zxing library with the python wrapper for it from https://github.com/dlenski/python-zxing. The code snippets were written in either php, js, bash, python or java and we used the appropriate tools to execute them.

After a lot of archives unpacked we encounter another problem. Instead of the languages used so far, the new code snippet is written in brainfuck. As the remaining archive size was rather small, we decided to perform the several dozen of final unpacks by hand, using the zxing online tool for decoding and https://copy.sh/brainfuck/ for running bf code.

Finally, we end up with the text file containing the flag: `{FLG:P33k-4-b0o!UF0undM3,Y0urT0olb0xIsGr8!!1}`

The unpacking script for the first phase of the task:

```
import subprocess
import zxing
import py7zr
import os
from pyzbar import pyzbar
from PIL import Image


reader = zxing.BarCodeReader('zxing-master')

# zxing sometimes mistakes ] for }, so we have to fix them manually
def fix(code):
    stack = []
    code2 = ''
    for i in range(len(code)):
```

```
        c = code[i]
        if c == '{' or c == '[':
            stack.append(c)
        if c == '}' or c == ']':
            top = stack[-1]
            stack = stack[:-1]
            if top == '{':
                c = '}'
            else:
                c = ']'
        code2 += c
    return code2


path = 'fwqmBkDB'
outpath = 'fwqmBkDB'
while True:
    print(path, outpath)
    code = reader.decode(path + '.png')

    if code is None: #zxing had trouble with decoding qr codes, hence pyzbar library
        code = pyzbar.decode(Image.open(path + '.png'))[0].data.decode()
    else:
        code = code.data

    code = fix(code)

    pwd = ""
    if code.startswith('<?php'):
        script = open('script.php', 'w+')
        script.write(code)
        script.close()
        proc = subprocess.Popen(['php', './script.php'], stdout=subprocess.PIPE)
        pwd = proc.stdout.read().decode()
    elif code.startswith('var'):
        script = open('script.js', 'w+')
        script.write(code)
        script.close()
        proc = subprocess.Popen(['node', './script.js'], stdout=subprocess.PIPE)
        pwd = proc.stdout.read().decode()[:-1]
    elif code.startswith('class Main'):
        script = open('script.java', 'w+')
        script.write(code)
        script.close()
        subprocess.call(['javac',  './script.java'])
        proc = subprocess.Popen(['java', '-cp', '.', 'Main'], stdout=subprocess.PIPE)
        pwd = proc.stdout.read().decode()[:-1]
    elif len(code.split('\n')) < 3:
        script = open('script.sh', 'w+')
        script.write(code)
        script.close()
        proc = subprocess.Popen(['/bin/sh', './script.sh'], stdout=subprocess.PIPE)
```

```
        pwd = proc.stdout.read().decode()[:-1]
    else:
        script = open('script.py', 'w+')
        script.write(code)
        script.close()
        proc = subprocess.Popen(['python', './script.py'], stdout=subprocess.PIPE)
        pwd = proc.stdout.read().decode()[:-1]
    print(pwd)

    subprocess.call(['7z', 'x', path + '.zip', '-p' + pwd, '-o' + outpath],
stdout=subprocess.DEVNULL)
    fn = os.listdir(outpath)[0]
    fn = fn[:-4]
    path = outpath + '/' + fn
    outpath = fn
```

(400) Sphinx's math

The task is to solve 500 systems of linear equations displayed on the web page.

Our solution uses `BeautifulSoup4` for extracting tasks from HTML page, `numpy` for solving linear matrix equations and `requests` for communication with the web server. As `numpy.linalg.solve` had problems with solving non-linearly independent systems, we used the additional methods from `sympy` to extract the linearly independent part, then proceeded to use `numpy`.

The following script solves the challenge:

```
#!/usr/bin/env python3
import re
import requests
from bs4 import BeautifulSoup
import numpy as np
import random
import sympy


s = requests.session()

def solveEq(ls, rs):
    for i in range(len(ls[0]) - 1, -1, -1):
        if all([row[i] == 0 for row in ls]):
            for j in range(len(ls)):
                ls[j] = ls[j][:i] + ls[j][i + 1:]

    last_ls = ls[-1]
    ls = ls[0:-1]
    rs = rs[0:-1]
    newRS = [[x] for x in rs]

    A = np.array(ls)
    B = np.array(newRS)

    _, inds = sympy.Matrix(A).T.rref()
    inds = list(inds)
    A = A[inds]
    B = B[inds]

    solution = np.linalg.solve(A, B)
    solution = [[round(s[0])] for s in solution]
    answer = sum([solution[i][0] * last_ls[i] for i in range(len(solution))])
    return round(answer, 6)

def getEq(text):
    soup = BeautifulSoup(text, "html5lib")

    ls = []
    rs = []
    var = {}
```

```
    enigma = soup.find("div", class_="enigma")
    children = enigma.findChildren("p")

    for child in children:
        eq = child.text
        [l, r] = eq.split("=")
        rs.append(int(eval(r)) if r != ' ?' else '?')

        # Replace non-digit and non-operator to A,B,C...
        for letter in l:
            if letter not in "0123456789?=+-*/ .()" and letter not in var:
                    var[letter] = chr(ord('A') + len(var))

    for child in children:
        eq = child.text
        [l, r] = eq.split("=")
        for letter in var:
            l = l.replace(letter, var[letter])

        parted = re.findall("([\(\d.+\-\)]+)(\w)", l)
        left = [0 for i in range(len(children) - 1)]
        for el in parted:
            left[ord(el[1]) - ord('A')] = int(eval(el[0]))
        ls.append(left)

    return ls, rs


r = s.get("http://codingbox4sm.reply.it:1338/sphinxsEquaji/")
results = []
last_idx = -1
for i in range(512):
    ls, rs = getEq(r.text)
    solved = solveEq(ls, rs)

    # the gift of writing code at 4am
    if int(solved) == solved:
        solved = int(solved)

    print("answer={}".format(solved))
    r = s.post("http://codingbox4sm.reply.it:1338/sphinxsEquaji/answer?answer=
{}".format(solved))
    idx = int(re.search('Step (\d+) of 512.', r.text).group(1))
    if not idx > last_idx:
        print("Error on step", last_idx)
        break
    last_idx = idx
    results.append(idx)

    if idx > 509:
        print(r.text)
```

(500) Space-time honeymoon

We are given a webserver with several endpoints:

- http://codingbox5sh.reply.it:2449/romeo/introduce
- http://codingbox5sh.reply.it:2449/juliet/introduce
- http://codingbox5sh.reply.it:2449/romeo/private-channel
- http://codingbox5sh.reply.it:2449/juliet/private-channel

The `introduce` endpoints allow us to get the first authorization token and start the exchange. The `private-channel` endpoints can be used to communicate with both Romeo and Juliet, by issuing a POST to send them a message and GET to fetch their question.

GETing the `private-channel` for the first time after the introduction will return a base64 encoded audio file. Additionally, the server's response contains new token and information to ask the other party a question.

Playing the audio file we hear a sequence of beeps which turn out to be the sounds of a DTMF telephone keypad. Decoding the beeps and using the keypad layout provided on the webserver, we get the following message: `x_of_the_city_0_is:_XXX`, where `XXX` is some number.

GETing the other `private-channel` endpoint with new authorization token, we get a question to provide a received coordinate. After sending this coordinate (encoded as a series of digits corresponding to the keypad) with a POST, we get another information about the coordinate and information to ask the first party next question.

This process repeats itself till we get the `those_are_all_the_cities_we_want_to_see` response instead of the usual one with the coordinates. After performing one last GET, we get the question `how_long_is_the_shortest_trip_to_visit_all_cities`.

Given the information provided on the webserver, we deduce the following information:

- distance between two cities is calculated as an euclidean distance in 4d space, with time being the last dimension (we're not entirely sure if that's correct, check notice at the end of the writeup)
- we can teleport instantly to the place we've already visited

The second information is very important, as it means that our trip don't have to be a continous path. Instead, any spanning tree of the cities can be used as a valid trip visiting all cities. Therefore, we solve for the MST of the graph and send back the output (encoded as a keypad presses) to the server.

The solving scripts are large, therefore we attatched them in the seperate files (`goertzelify.py` for decoding the DTMF and `code500.py` for the communication and other stuff).

Notice: we didn't manage to get the flag during the competition, due to what we believe to have been a bug with the server itself. As for the final, "trip planning" part, we tried many different formulas for distance (3d euclidean + absolute difference of time, taking floor of the whole output instead of every single distance, etc.) and problems to solve (TSP instead of MST), but to no avail.