User Name: Sui

This is the write-up for PKU GGG0 (May. 16-23, 2021) written by Sui Li (sui@pku.edu.cn). Attached are some code files for a few problems, but they are only attached as proof-of-work, and are not needed to understand anything in this write-up.

签到

Given text:
`c3ludHtKM3lwYnpyIGdiIDBndSBDWEggVGhUaFRoLCByYXdibCBndXIgdG56ciF9`

- Step 1: base64 decode
  `synt{J3ypbzr gb 0gu CXH ThThTh, rawbl gur tnzr!}`
- Step 2: Caesar Cipher decode (-13 to all alphabetical characters)
  `flag{W3lcome to 0th PKU GuGuGu, enjoy the game!}`

主的替代品

Problem description: Write a C program whose output contains a substring "main", while the c code itself does not contain the substring "main".

- The only difficulty is the main fuction name, which can be solved easily by macros.

```c
#include<stdio.h>
#define fun m###a###i##n
int fun()
{
    printf("ma");
    printf("in\n");
    return 0;
}
```

- Solving this problem gives us the flag
  `flag{to_main_or_not_to_main_that_is_a_question_c274b32d}`

小北问答

- Q1: 理科一号楼共有 8 个计算中心机房，其中第 n 机房的门牌号是Xn（1000≤Xn≤9999），求 ∑(Xn)n 的最大质因数
  - Since my lab is at LiKeYiHaoLou 2nd floor, I just had a walk around the corridor and got the required Rm. Numbers.
  - Then calculate the factorization using python.

```python
  # pip install sympy
  from sympy.ntheory import factorint
  s = 1258^1 + 1263^2 + 1261^3 + 1204^4 + 1203^5 + 1249^6 + 1339^7 + 1338^8      # 10279576720584031841969783
  factorint(s)
        # {17: 1, 5574340319: 1, 108475792463321: 1}
```

- Q2: 北京大学的哪门课被称为“讲得好、作业少、考试水、给分高的课”（中文全称）？

    - Google "讲得好、作业少、考试水、给分高" (including quotes)
    - The only hit is courses.pinzhixiaoyuan.com . Its homepage has a subtitle with underlined text "讲得好、作业少、考试水、给分高". Hover the mouse over it and we have the answer "穆良柱老师的热学"
- Q3: 根据 HTCPCP-TEA 协议，当一个茶壶暂时无法煮咖啡时，应当返回什么状态码?
    - Google "HTCPCP-TEA"
    - One of the results is https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/418, which tells us the answer is 503
- Q4: 在 Conway's Game of Life 中，有多少种稳定的由 7 个活细胞构成的局面？稳定是指每个时刻的状态都与初始状态完全相同。旋转或对称后相同的视为同一种局面。
    - Google "Conway's Game of Life", and learn its definition from wikipedia
    - Google "Conway's Game of Life stable patterns"
        - The answer (4) lies in another wiki page https://en.wikipedia.org/wiki/Still_life_(cellular_automaton)
- Q5: FAStT Management Suite Java 是 IBM 推出的一款软件，它的默认密码是?
    - Google "FAStT Management Suite Java"
    - One of the results seems like its official diagnostic and configuration tool. Open the page https://www.ibm.com/support/pages/ibm-fastt-management-suite-java-msj-diagnostic-and-configuration-utility-microsoft-windows-2000-windows-nt-40-and-novell-netware-fibre-channel-solutions
    - Among the download links there's a text file, follow the link. ftp://ftp.software.ibm.com/systems/support/system_x/90p4957.txt
    - Skim through the page and we can find the answer 'config'
- Q6: 最小的汉信码图案由多少像素（被称为“模块”）构成?
    - (It's ridiculous that I could not find anything useful in Chinese webpages.)
    - Google "Han Xin Code"
    - Follow the first hit https://barcodeguide.seagullscientific.com/Content/Symbologies/Han_Xin.htm
    - On the last text field we can learn that the smallest code is of size 23 * 23, therefore the answer is 529.
- Q7: 哪个国密算法基于椭圆曲线密码?
    - Google "国密算法 椭圆曲线" and we can find the answer "SM2" among the titles.
- Q8: 在 2013 年 5 月 4 日，全世界共有多少可用的顶级域名（TLD）?
    - Google "top domain list"
    - Among the results there's a ICANN page https://www.icann.org/resources/pages/tlds-2012-02-25-en
    - Follow the link on the page and we could find IANA's official list file https://data.iana.org/TLD/tlds-alpha-by-domain.txt
    - Use https://archive.org/web/ to find the history version of this file
        - We could not find the file on the exact day, but the versions of the file few days before or afterwards the requested date are the same, both contain 317 entries, thus the answer.

All the above procedures only showed the correct way found after tons of trial and errors, so they may look a lot easier than they originally were.
The two flags gained from answering 5 and 8 of these questions correctly are `flag{you-are-master-of-searching_92bc938a}` and `flag{you-are-phd-of-searching_fc6ce5ab}` respectively.

与佛论禅网大会员

Problem description: find the hidden keys in the provided gif file "quiz.gif"

- Run shell command `xxd quiz.gif` and from the text and I realize there's a zip file with two txt files in side at the end of the gif file (Thanks to the quine.zip problem, I recognized it at the first sight)
- Count the bytes carefully, and run the following collamd to get the zip file `dd skip=236978 count=392 if=quiz.gif of=out.zip bs=1`
- Extract the zip file and get flag1.txt. `flag{K33p_going!PasswOrd-is-hidden-in-the-1mage}`
- The flag2.txt is encrypted so we still need to find the password from the image. Look carefully and we can realize there are some unusual pixels at the corners of frame 2, 4, 6 and 8.
- Open the image with photoshop, carefully extract the four parts, put them together and got a QR code, but it lacks some squares on each corners.
- Goto `https://en.wikipedia.org/wiki/QR_code#Design` , and we can find some standard templates of QR codes. Count our QR code at hand and found it has the same dimension as the "size-3" QR code template.
- Copy the template size-3 QR code from wikipedia into our photoshop file, align the dimensions manually, cut its three squares from the upper left, upper right and lower left corners and paste them on our QR code. The lower right doesn't matter because of the redundancy in the design of QR-codes.
- Scan our final QR code, and we got the link with the password of the zip file. `https://www.pku.edu.cn/#hint=zip_password_is_fm2jbn2z6tOgl5le`
- Extract the zip file find the flag `flag{you are master of steganO. Here is yOur flag}`. Gained the title "Master of steganography" XD

2038 年的银行

Problem description: With the starting fund of 500 CNY, try to buy the flag (999888777 CNY) from three banks after a series of saving / loaning operations.

- Randomly play with it for a while, and find out the maximum money I can loan is something like $min(2000000000, 10 \times maximum\_net\_savings\_ever\_made\_in\_that\_bank)$ and it's refreshed only in the next day if you made any loan in the current day.
- Play around a bit more, repeat the following process:
  - deposit all money into bank i
  - loan maximum amount of money from back i (the credit limit equals to 10 times the deposit)
- Change around all 3 banks and repeat the above process until all 3 banks used, then buy a bread and sleep and do another round in the following day.
- Then in one moment one of the numbers will overflow. After some experiments we can know all the numbers in the program are 32bit integers.
- Since we already have a lot of money at hand, we can evenly distribute them in the 3 banks, sleep with breads repeatedly and wait until they grow. In the meanwhile, the number of our loan will jump between two numbers around 2000000000 and -2000000000 respectively due to repeated overflowing, there's no worry about them at all.
- Before our savings in banks overflow, withdraw money evenly from the 3 banks and fully repay the loans for one bank. Repeat until we owe no money.
- Withdraw some money and buy a flag `flag{SucH_Na!V3_b4Nk_e2b37210}`

人类行为研究实验

Problem description: Play a game that shows us 20 numbers one by one and asks us to press the button when we think the current number is the maximum of the 20. Repeat the game and win 12 out of 20 consecutive games and we are eligible for the first flag. "Upload" score into a fake IAAA login system to earn the second flag.

- First spend some time playing the game. A number around 6400 is usually guaranteed to be the maximum. If all the numbers in the current round are small, we can click the button as soon as we see something around 5900.
- After it tells us we passed, we look at the source code of the webpage and found the link that we should follow to get the first flag (I can't provide the exact link here since the server seems to be down at the time of this writing.)
- Then we are redirected into a fake BaiJingDaXue IAAA login system and asked to log in to unplad our score. Played around with it for a while, and find it only tells us that we are not teacher and not allowed to upload our score.
- Notice there's a jwt string in the http get request when loggin in. Decode the jwt string in jwt.io and found out its payload says { "identity": "student" }. Change the student to teacher.
- The other problem is we need a password to generate the correct signature in the jwt string. After trying brute-forcing for no avail, I tried to set it to an empty string, and succeeded in the login, and got the second flag.

弗拉梅尔的宝石商店

Problem description: We have a starting fund of $500 and a commodity in the inventory that worth 250. We can only do business with one merchant, everything can only be sold for 90% of the price it's purchased. Try to buy a flag (worth $100000). The pyc file is provided.

- Decompile the pyc code using uncompyle6. It says part of the code failed the decompilation, but after reading the code carefully we can be sure that's nothing important.
- The code tells us that every transaction that we provided to the system is first written in a text file with a fixed filename that's generated according to our token. Then The transaction will need to pass a round of `check_transaction()` test after being executed with `perform_transaction()`. So our target is to deal with these two files respectively.
    - Task 1: find a way around `check_transaction()`
        - By reading the `check_transaction()` code we can learn about its behaviours:
            - All numbers and money amount are checked to make sure they are within the limits, so there's nothing we can cheet here.
            - Each line of transactions are checked independently, so we may be able to make a group of same transactions that are each within the limits when on its own but exceed the limits when combined.
            - Buying the flag will not pass the check, and the program will say it's not for sale.
        - The largest obstacle here is the flag being not for sale. But this also gives us the most important hint that we have to invent a way to bypass this verification.
        - Since all the transactions are written into a file and then read later in `check_transaction()` and `perform_transaction` each time independently, we realized we can run another instance of the program and overwrite the content of file after it passed the check.
    - Task 2: find a way to do unallowed operations in `perform_transaction()`

        - After reading its code, we learned that it does value checks and operations alternatively, and whole transcation does not rollback when unallowed operations are performed.
        - The thing that we can exploit is when we sell stuff, we first gain money, and then the commodity is deducted from our inventory. So we may sell anything that we don't even own.
- The whole solution is as follows (we need to open two consoles in order to overwrite the transaction file):
    - In console 1, enter the following transaction, and wait before confirming with 'y':
      trade
      citrine -1
      END
    - In console 2, enter the following transaction:
      trade
      flag -10
      END
    - In console 1, confirm the transaction with 'y'. We now have enough money to buy several flags.
    - In console 1, enter the following transaction, and wait before confirming with 'y':
      trade
      citrine -1
      END
    - In console 2, enter the following transaction:
      trade
      flag 1
      END
    - In console 1, confirm the transaction with 'y'. We now have a flag in our inventory.
    - In console 1, type 'inspect', and we can see the content of the flag `flag{a_good_merchant_knows_how_to_make_money_2fb3abc2}`

I loved this problem a lot. The process of solving it was really enjoyable.

未来的机器

Problem description: A code file written in an imagenary ASM format is provided alongside with an interpreter written in python. The ASM code takes a string (which contains a flag) as input, and verifies whether it is correct or not.

- First spend hours reading the ASM code. After reading we found out it consists of 7 loops
    - Loop 1: generate an array [0,1,2,...95]
    - Loop 2: permutate the array using some hash functions seeded with 114514 (this number stinks!)
    - Loop 3: permutate the input using the permutation described in the array generated in the last step
    - Loop 4: generate an array [0,1,2,...len(input)]
    - Loop 5: permutate the array using some hash functions
    - Loop 6: permutate the result of loop 3 using the permutation described in the array generated in the last step
    - Loop 7: compare the result of loop 6 with a predefined key and return the result of comparation.
- We may simplify this process in python: (assume input length is 41, no one enter anything else anyway)

```
key = '.q~03QKLNSp"s6AQtEW<=MNv9(ZMYntg2N9hSe5=k'

table1 = list(range(96))
h = 114514
for i in range(1, 96):
    h = (h * 1919 + 7) % 334363
    table1[h % i], table1[i] = table1[i], table1[h % i]

table2 = list(range(41))
for i in range(1, 41):
    h = (h * 1919 + 7) % 334363
    table2[h % i], table2[i] = table2[i], table2[h % i]

def encode1(instr):
    re = [0 for _ in instr]
    for i in range(41):
        re[i] = (table1[instr[i] - 32] + i) % 96 + 32
    return re

def encode2(instr):
    re = [0 for _ in instr]
    for i in range(41):
        re[table2[i]] = instr[i]
    return re

print(encode2(encode1([x for x in input()])) == [x for x in key.encode()])
```

- So we may write the decoder as follows:

```
def decode2(instr):
    re = [0 for _ in instr]
    for i in range(41):
        re[i] = instr[table2[i]]
    return re

rev1 = {j:i for i,j in enumerate(table1)}
def decode1(instr):
    re = [0 for _ in instr]
    for i in range(41):
        re[i] = rev1[(instr[i] - 32 - i) % 96] + 32
    return re

keystr = [x for x in key.encode()]
tmp2 = decode2(keystr)
tmp1 = decode1(tmp2)
result = ''.join([chr(i) for i in tmp1])
print(result)
```

- Run the decoder and we have our flag: `flag{W4SM_1S_sO_fun_but_1t5_subs3t_isNOT}`

无法预料的问答

Problem description: Repeatedly answering multiple-choice questions until achieving 20 correct answers in a row. All questions are of the same format (Which emoji among the following does You Chan love the most?). There are totally no less than 66 emojis, each question only entails the relative order between 2-4 emojis.

- At first I tried to do this by hand (I mean seriously, with written notes), but had to abandon after half an hour when discovering I can't give each emoji a distinguishable name. Then I tried to do this by hand again by copy-pasting all the info I know in a file, with a helper program that helps me generate all relationships that can be deduced from the current input data, but had to give up after like more than an hour (I was able to answer like 8 questions right in a row by that time) because I made a minor mistake in my records and the error was impossible to locate.
- After some bad tries I'm finally convinced this problem can only be solved by programming. I wrote a python script, sending automatic requests and maintain all the relative orders learned. Following are the details (There's nothing intersting about this script at all, so no need to continue reading)
  - I maintained all the relative orders using two dicts, with format `{emoji: set_of_emojis_better_than_it}` and `{emoji: set_of_emojis_worse_than_it}`
  - for each new learned pair of information `x > y`, record the partial order between every pair bewteen these two sets: {all the emojis better than x including x} and {all the emojis worse than y including y}
  - in each round, if we can deduce one of the emoji is better than every other choices, we can choose that emoji, otherwise, we randomly choose one.
    - if the answer is correct, the answer is better than all other choices, update our records with all these new relations.
    - if the asswer is wrong and there was only 2 choices, we learned our choice is worse than the other choice, update our records with this pair.
  - all partial relations stored in dicts (as described above) are saved in a file automatically, and when we rerun the program it will load from this file, so that we can sae some time.
  - After running the program for a while we can get the key. `flag{yOuAr3_sOO_K1raK1ra_f2f8209f}`

安全的密钥交换

Problem description: Alice and bob invented a way to send AES keys. Given their algorithm script, perform a Man-In-The-Middle attack on Alice and Bob's communication and intercept the two flags alice sent to bob.

- After reading the code, we summarize Alice and Bob's actions as follows:
  - Step 1: AES Key Exchange
    - Alice: generate a random 1024 bit even integer da, send $pka = G^{da}(modP)$ to Bob
    - Bob: generate a random 1024 bit even integer db, send $pkb = G^{db}(modP)$ to Alice
    - Alice: receive bob's $pkb = G^{db}(modP)$, and calculate Alice and Bob's shared key $key = pkb^d a = G^{da*db}(modP)$, take its first 128 bytes as AES key.
    - Bob: receive Alice's $pka = G^{db}(modP)$, and calculate Alice and Bob's shared key $key = pka^d b = G^{da*db}(modP)$, take its first 128 bytes as AES key.
  - Step 2: Certificate Verification
    - Alice: Send AES-encrypted cert_A to Bob
    - Bob: Send AES-encrypted cert_B to Alice
    - Alice: Receive and decrypt cert_B, Validate Bob's certificate

        - Bob: Receive and decrypt cert_A, Validate Alice's certificate
    - Step 3: Flag Transmission
        - Alice: if if key is less than 512 bits long, send flag1 to Bob, otherwize send flag2 to Bob. All under AES encryption
        - Bob: Receive and decrypt the message to obtain the flag.
- After playing around the provided web console, we made sure that Alice and Bob cannot hear each other, So it's a perfect setting for us to make an Man-In-The-Middle attack.
- I originally tried to only talk to alice, but found that her certificate verification seems different from the code. I could not pass her certificate verification with her own certificate.
    - So I have to talk to Bob as well just to get Bob's certificate, and with Bob's certificate, I was able to pass Alice's certification verification.
- Another problem is how to get Alice's flag No. 1. Since random numbers nearly always result in a key with more than 512 bits long (since the prime is a lot longer than this), we need to find a better d instead of random number, in order to get a smaller key.
    - This reminds me of Fermat's Little Theorem (I feel so nostalgic, as if back in high school). Since for any prime number P, and number G (as long as G is not P's multiple) $G^{P-1} = 1(modP)$. We may just choose d = P - 1, then we can guarantee that the final key $key = G^{da*d}(modP) = (G^d)^d a(modP) = 1$
- We summarize the whole MITM process as follows:
    - Step 1: AES Key Exchange
        - Generate a random 1024 bit even integer d (or P-1 if we want flag 1), and send it to Alice and Bob.
        - Receive $pka = G^{da}(modP)$ from Alice, and $pkb = G^{db}(modP)$ from Bob.
        - Calulate $keya = pka^d = G^{da*d}$, and $keyb = pkb^d = G^{db*d}$, and take the first 128 bytes as the AES keys to Alice and Bob, respectively.
    - Step 2: Certification Verification
        - Receive Bob's AES-encrypted certificate, encrypt it using Alice's AES key and send it to Alice. Receive and verify Alice's certificate.
    - Step 3: Receive the flag
        - Receive encrypted flag from Alice, decrypt it and get the flag.

The two flags obtained by Fermat's Little Theorem and random bits are `flag{th3_Tr1V14l_1s_NOt_trIviAL_dO527677}` and `flag{Wh4t_A_w3Ak_PrOtoCo1_3b464a35}` respectively.

计算概论B

(First Blood Award)
Problem description: An unknown algorithm encoded an input string into some binary bits. Given the letter frequency in the original input string and the output binary, try to recover the input string.

- There's an assertion in the code sample suggesting the letters are translated to bits with different prefices. This suggested a Trie-Tree-like structure. Furthermore, the letter frequency statistics strongly hints a Huffman tree.
- We build a Huffman tree from the letter frequency statistics, decode the bits and got a hex string
- Reverse the order of the hex string, as the sample code has done before the huffman, and decode it into bytes. The flag is right inside the decoded text: `flag{wOw_congrats_th1s_1s_rea11y_huffman}`

The code is pretty simple.

```
# pip install bigstring dahuffman
from dahuffman import HuffmanCodec
from bitstring import BitArray

# assume the encoded bit string is stored in `data`, and the frequency stat is
stored as a dict `freq`
codec = HuffmanCodec.from_frequencies(freq)
decoded = codec.decode(BitArray(bin=data).tobytes())
result = binascii.unhexlify(decoded[::-1].encode())
```

巴别压缩包

Problem description: Given a zip file that extract to a zip exactly like itself, fill in its missing CRC32 values.

- I took lots of time reading zip specifications and the Deflate algorithm that it used for compressing, in order to understand how it works (it's fun!). However the real problem is probably a lot easier than that.
- All we have to do is filling in 16-bytes, which are all CRC32 values. According to zip specifications and how quine works, the 4 DWORDS should all be the same. So all we have to do is to fill in one DWORD (4 bytes), that's only 2^32 combinations.
- I initially tried to brute-force using shell command `unzip -v quine.zip` however it was so slow (it may take 1000 hours) that I nearly lost confidence in brute-forcing. I then tried to read CRC32 algorithm and learnt its properties.
- Then I came up a more efficient brute-forcing method, using CRC32's properties that we can concatenate two CRC outputs efficiently. But when I was trying to write it in C, it occurred to me why don't I try simple brute-forcing using C first?
- I wrote a C program, simply calling zlib's crc32 method (without my previously intended CRC32 optimization), and, wow, it ran 20000 times faster than my shell script. It only took less than 3 minutes to brute-force the CRC value which is 4a0955f5. Writing it in little endian and we got our flag `flag{QUINE_F555094A_F555094A_F555094A_F555094A}`

Afterwords

Before this tournament, I had zero experience or knowledge about any CTF-related activities. This week has been amazing for me, I could feel my internal geek's soul burning every minute. It's like all my passion for computer science that I had lost over time was back again. Huge THANKS to the authors to all these great problems, they are really intriging, and I learnt a ton from them.