Sécurité des réseaux sans fil

Laboratoire 802.11 sécurité MAC

Auteurs: Godi Matthieu, Issolah Maude

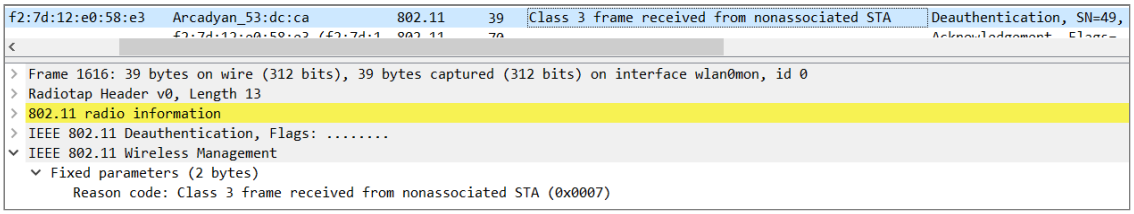
Date: 31.03.2022

Partie 1 - beacons, authenfication

Deauthentication attack

Questions:

- Quel code est utilisé par aircrack pour déauthentifier un client 802.11. Quelle est son interprétation ?
  - Le code 7: Class 3 frame received from nonassociated STA.
  - La station a essayé d'envoyer des données avant d'être associée à l'AP.



- A l'aide d'un filtre d'affichage, essayer de trouver d'autres trames de déauthentification dans votre capture. Avez-vous en trouvé d'autres ? Si oui, quel code contient-elle et quelle est son interprétation ?

  Toutes les trames de désautentification étaient du même type.

- Quels codes/raisons justifient l'envoie de la trame à la STA cible et pourquoi ?

  Les codes qui impliquent un problème du côté de la STA, ou du réseau.

  La station reçoit une information de pourquoi elle a été déconnectée.

  - 1: Unspecified reason
  - 2: Previous authentication no longer valid
  - 4: Disassociated due to inactivity
  - 6: Class 2 frame received from nonauthenticated station
  - 7: Class 3 frame received from nonassociated station
  - 9: Station requesting (re)association is not authenticated with responding station

- Quels codes/raisons justifient l'envoie de la trame à l'AP et pourquoi ?

  Les codes qui impliquent un problème du côté de l'AP, ce qui coupe la connexion et désauthentifie la station:

  - 3: station is leaving (or has left) IBSS or ESS
  - 5: Disassociated because AP is unable to handle all currently associated stations

    - 8: Disassociated because sending station is leaving (or has left) BSS

- Comment essayer de déauthentifier toutes les STA ?

  En envoyant un message de déasuthentification en broadcast.

- Quelle est la différence entre le code 3 et le code 8 de la liste ?

  Le code 3 est dû à la disparition de l'access point, tandis que le 8, est dû à un changement d'access point fait par l'os de la station.

- Expliquer l'effet de cette attaque sur la cible

  La connexion entre l'AP et la STA est coupée, et la STA doit se réauthentifier.

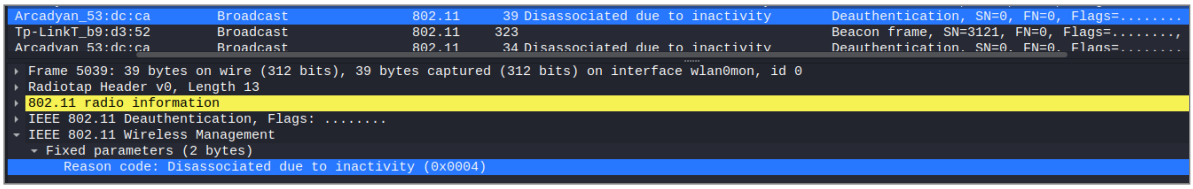
Script: deauthentication.py

```
└─$ sudo python3 deauthentication.py
Choose a reason:
1 - Unspecified
4 - Disassociated due to inactivity
5 - Disassociated because AP is unable to handle all currently associated stations
8 - Deauthenticated because sending STA is leaving BSS
4
.....................................................................................
.........
Sent 100 packets.
```

Script en console

Validation:

Dans Wireshark nous pouvons vérifier que nous envoyons bien un message de désauthentification de l'AP en broadcast, et que ce message a bien le code choisi par l'utilisateur (ici le 4).



Validation via Wireshark

Fake channel evil tween attack

Ce script liste les AP disponibles, sauf ceux qui n'ont pas de nom, car impossible à copier.

Script: fake-channel.py

```
└─sudo python3 fake-channel.py                                    130 ×
1 AP MAC: a0:b5:49:2d:61:c8 with SSID: lcg-33033 - channel: 11 - RSSI: -81
2 AP MAC: 48:8d:36:53:dc:ca with SSID: this-wifi-is-a-lie - channel: 11 - RSSI: -49
3 AP MAC: 10:5a:f7:66:04:a8 with SSID: xfd-28862 - channel: 1 - RSSI: -79
4 AP MAC: 56:67:11:23:01:84 with SSID: UPC Wi-Free - channel: 1 - RSSI: -83
5 AP MAC: 8c:59:c3:ca:18:70 with SSID: axa-79627 - channel: 1 - RSSI: -79
6 AP MAC: fc:ec:da:b1:0b:17 with SSID: knecht - channel: 1 - RSSI: -73
7 AP MAC: 54:67:51:23:01:84 with SSID: UPC9D2BC7E - channel: 1 - RSSI: -81
8 AP MAC: 08:3e:5d:26:6b:24 with SSID: Sunrise_2.4GHz_266B20 - channel: 1 - RSSI: -69
9 AP MAC: 0a:ec:da:b1:0b:17 with SSID: Intern - channel: 1 - RSSI: -75
10 AP MAC: 06:ec:da:b1:0b:17 with SSID: Public - channel: 1 - RSSI: -77
11 AP MAC: 96:53:30:b4:6b:33 with SSID: DIRECT-33-HP M277 LaserJet - channel: 1 - RSSI: -83
^CSelectionner le SSID a attaquer (1 - 11):2
Fake SSID on channel: 5
.....................................................^C
Sent 59 packets.
```

Script en console

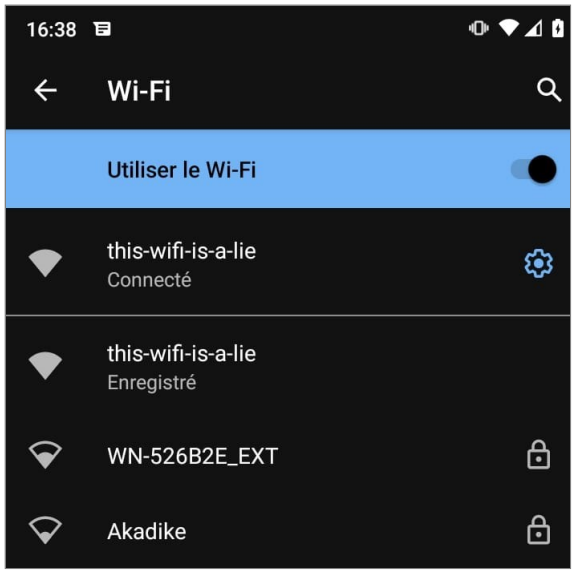
Validation:

Dans Wireshark nous pouvons vérifier que nous envoyons bien des beacons avec le même SSID que celui choisi par l'utilisateur, et sur le canal calculé par le script à une distance de 6 du canal d'origine. Ici le canal d'origine est le 11, et les beacons forgée sont envoyées sur le canal 5.

```
                    Beacon frame, SN=0, FN=0, Flags=........, BI=100, SSID=this-wifi-is-a-lie
                    Acknowledgement, Flags=........
▸ Frame 77286: 67 bytes on wire (536 bits), 67 bytes captured (536 bits) on interface wlan0mon, id 0
▸ Radiotap Header v0, Length 8
  802.11 radio information
▸ IEEE 802.11 Beacon frame, Flags: ........
▾ IEEE 802.11 Wireless Management
  ▸ Fixed parameters (12 bytes)
  ▾ Tagged parameters (23 bytes)
    ▾ Tag: SSID parameter set: this-wifi-is-a-lie
        Tag Number: SSID parameter set (0)
        Tag length: 18
        SSID: this-wifi-is-a-lie
    ▾ Tag: DS Parameter set: Current Channel: 5
        Tag Number: DS Parameter set (3)
        Tag length: 1
        Current Channel: 5
```

Validation via Wireshark

Ci-dessous, nous pouvons observer les deux AP avec le même nom.



Validation via un mobile

Question : Expliquer l'effet de cette attaque sur la cible

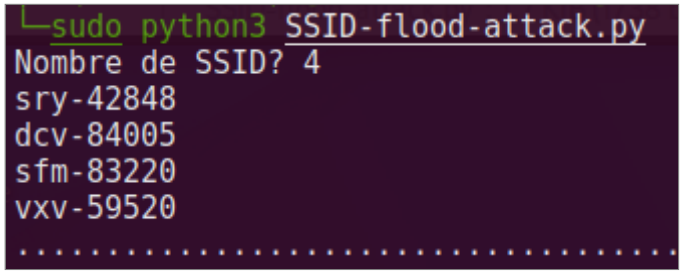
Cette attaque trompe la cible, qui peut se connecter automatiquement au faux réseau, s'il n'y a pas de mot de passe demandé.

SSID flood attack

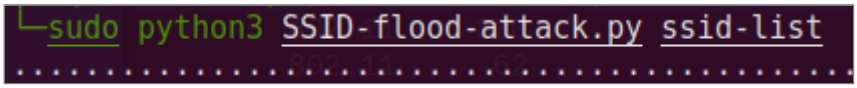
Script: ssid-flood-attack.py

Si le script est lancé sans argument, il va créer le nombre de SSID demandé par l'utilisateur.

Le pattern de nommage de ces SSID est: *abc-12345*. une fois créés, le script affiche la liste des SSIDs.



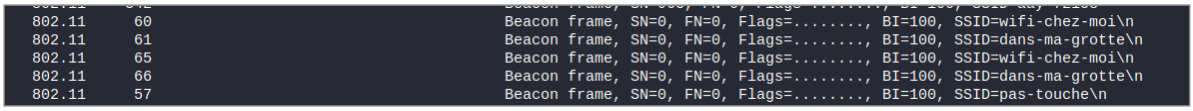
Script en console, sans argument



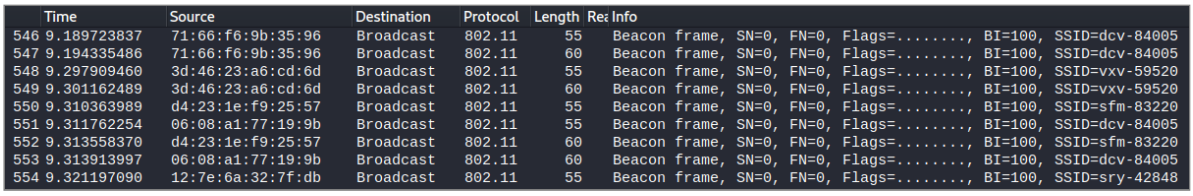
Script en console, avec une liste de SSID en argument

Vérification:

Nous voyons que nous envoyons bien des beacons avec les noms de la liste comme SSID, et ci-après avec les noms crées aléatoirement.



Paquets créés depuis la liste de noms



Paquets crées avec des nom aléatoires

Partie 2 - Probes

Probe Request Evil Twin Attack

Question : comment ça se fait que ces trames puissent être lues par tout le monde ? Ne serait-il pas plus judicieux de les chiffrer ?

La station cherche un réseau, mais n'est pas authentifiée. Elle ne peut donc pas chiffrer les probe request qu'elle envoi, car elle ne va pas créer une connexion sécurisée avec tous les AP qu'elle rencontre. Il est normal que ces messages ne soient pas chiffrées.

Question : pourquoi les dispositifs iOS et Android récents ne peuvent-ils plus être tracés avec cette méthode ?

Car ils créent des adresses MAC aléatoires pour cacher la vrai.

Script: twin-attack.py

Ce script prend un SSID en argument et créer un faux AP s'il trouve des stations cherchant ce SSID.

```
└─$ sudo python3 twin-attack.py Sunrise_5GHz_266B20
SSID Sunrise_5GHz_266B20 found, sending packets...
....................................................
Sent 61 packets.
```

Script en console

Vérification:

Nous pouvons constater que nous envoyons bien des beacons se faisant passer pour l'AP.

```
Beacon frame, SN=0, FN=0, Flags=........, BI=100, SSID=Sunrise_5GHz_266B20
Beacon frame, SN=0, FN=0, Flags=........, BI=100, SSID=Sunrise_5GHz_266B20
Beacon frame, SN=0, FN=0, Flags=........, BI=100, SSID=Sunrise_5GHz_266B20
Beacon frame, SN=0, FN=0, Flags=........, BI=100, SSID=Sunrise_5GHz_266B20

Frame 26842: 70 bytes on wire (560 bits), 70 bytes captured (560 bits) on interface wlan0mon, id 0
Radiotap Header v0, Length 13
802.11 radio information
IEEE 802.11 Beacon frame, Flags: ........
  Type/Subtype: Beacon frame (0x0008)
▸ Frame Control Field: 0x8000
  .000 0000 0000 0000 = Duration: 0 microseconds
  Receiver address: Broadcast (ff:ff:ff:ff:ff:ff)
  Destination address: Broadcast (ff:ff:ff:ff:ff:ff)
  Transmitter address: 16:fb:73:26:58:2d (16:fb:73:26:58:2d)
  Source address: 16:fb:73:26:58:2d (16:fb:73:26:58:2d)
  BSS Id: 7a:10:fc:9b:50:92 (7a:10:fc:9b:50:92)
  .... .... .... 0000 = Fragment number: 0
  0000 0000 0000 .... = Sequence number: 0
IEEE 802.11 Wireless Management
```

Validation via Wireshark

Détection de clients et réseaux

Script: sta-searching-ap.py

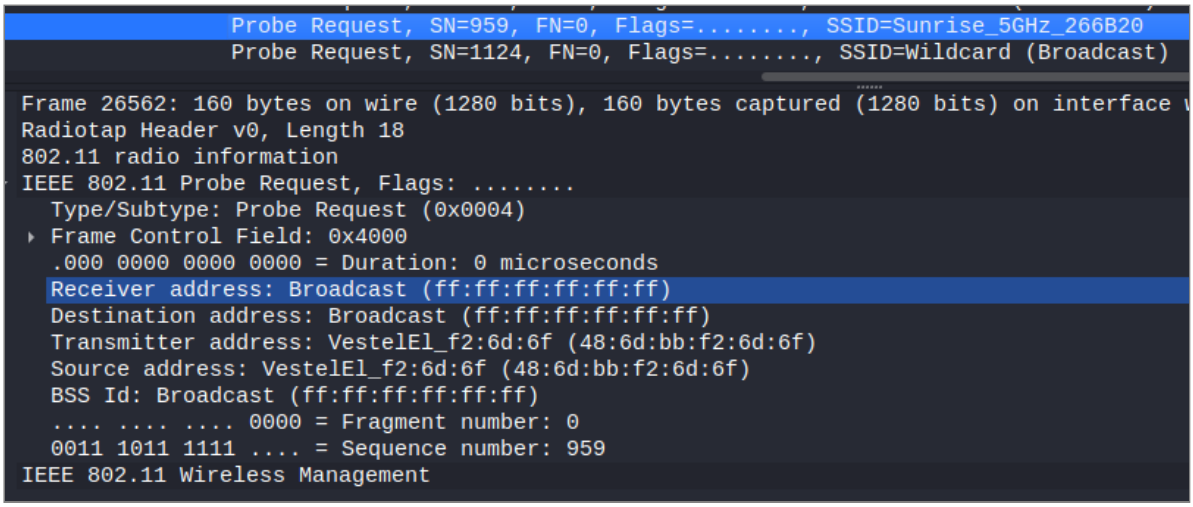
Ce script va lister les stations cherchant un AP spécifique passé en argument.

```
└─sudo python3 sta-searching-ap.py Sunrise_5GHz_266B20
STA looking for Sunrise_5GHz_266B20 AP:
48:6d:bb:f2:6d:6f
```

Script en console

Vérification:

Nous voyons dans Wireshark que le packet *Probe Request* a bien été pris en compte, et que la bonne adresse mac est affichée par le script.

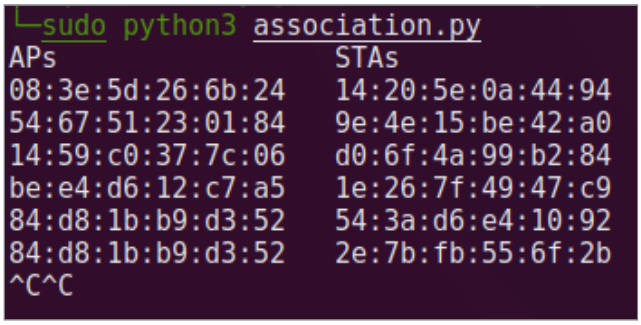


Validation via Wireshark

Script: association.py

Ce script liste les stations associée à un AP.

Pour ce faire il utilise des paquets qui sont échangés par des pairs associés: *block ack*.

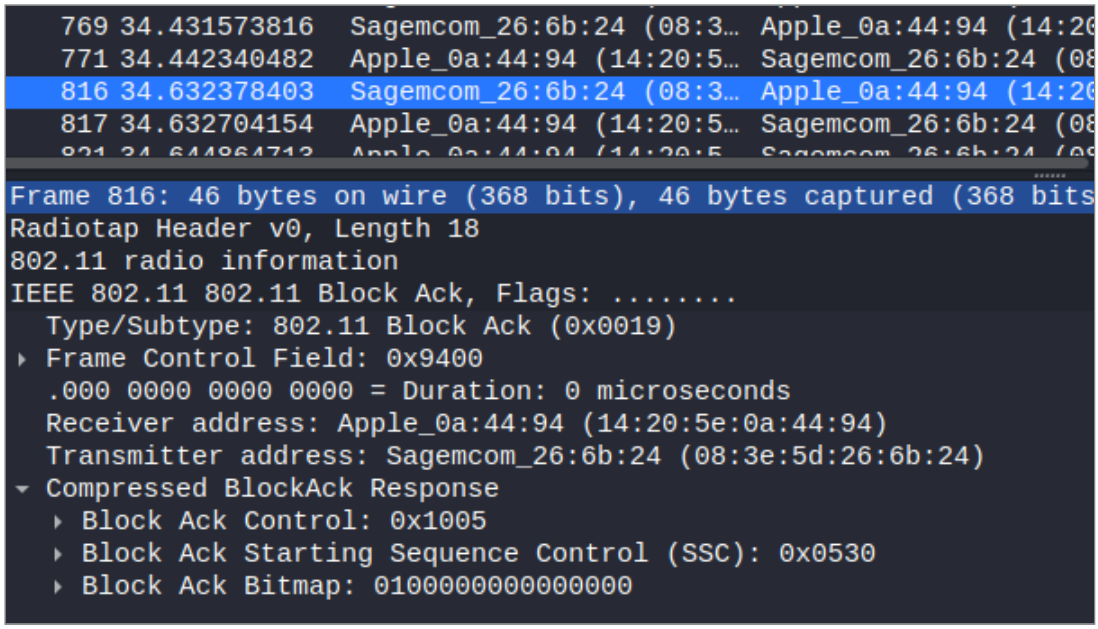


Script en console

Vérification:

Vérification de la première ligne affichée dans le terminal.

Nous pouvons observer que les deux adresses sont bien listée dans la bonne colonne (AP/STA).



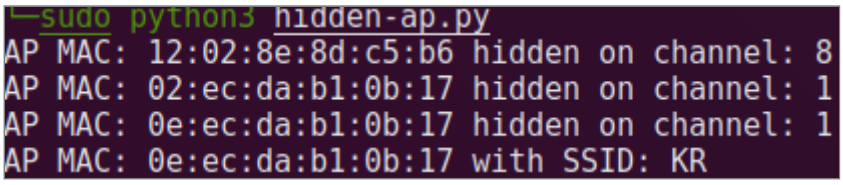
Validation via Wireshark

Hidden SSID reveal

Ce script trouve les SSID cachés en commençant par lister les beacons qui n'ont pas de SSID (chaîne vide).

Il stock les adresses mac de ces AP, puis écoute les *probe request/response* pour trouver une de ces adresses. Le SSID caché se trouve dans les paquets *probe*.

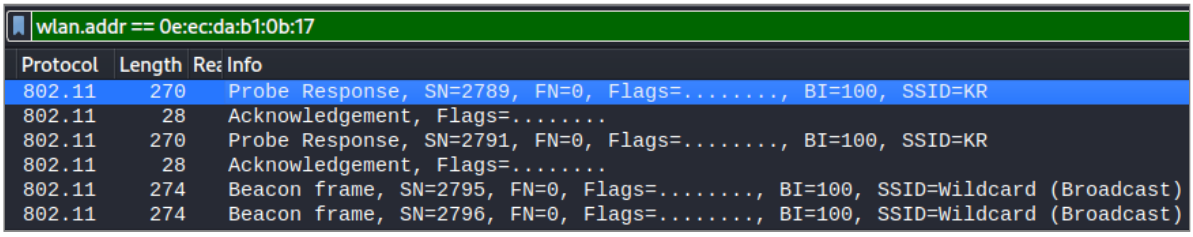
Script: hidden-ap.py



Script en console

Vérification:

En filtrant sur l'adresse mac donnée en console, on peut vérifier que cet AP envoi bien des *beacon*s vide, mais une *probe response* avec un SSID, et que c'est celui affiché en console.



Validation via Wireshark