TLS-SEC

Prise de contrôle d'un drone Parrot AR.Drone 2.0

Tuto sécu, THCon 2019

Florent Fayollas et Antoine Vacher

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

INP TOULOUSE ENSEEIHT

ENAC ECOLE NATIONALE DE L'AVIATION CIVILE

Introduction

L'objectif de ce tutoriel est de vous faire une introduction à la bibliothèque Python Scapy dans un cadre de prise de contôle d'un drone Parrot AR.Drone 2.0, contrôlé par WiFi. L'attaque originelle est nommée SkyJack [1] et a été développée par Samy Kamkar.

Le drone Parrot AR.Drone 2.0 est contrôlé par WiFi. Celui-ci crée un réseau WiFi ouvert nommé par défaut sur le modèle `ardrone2_xxxxxx`. Le pilote n'a alors simplement qu'à se connecter au réseau avec son smartphone pour contrôler le drone.

L'attaque de Samy Kamkar est perpétrée en désassociant le vrai pilote du réseau WiFi du drone puis en s'y connectant à sa place. C'est ce que nous allons voir ici.

Pré-requis sur la carte WiFi

Une carte WiFi peut fonctionner dans 4 modes différents :

— maître (*master*), pour se comporter comme un point d'accès
— infrastructure (*managed*), pour être un client communiquant à travers un point d'accès
— *ad-hoc*, pour un réseau point-à-point
— moniteur (*monitor*), permettant une écoute passive

Le mode moniteur est celui que nous allons utiliser ici. Votre adaptateur WiFi doit donc le supporter. Pour vérifier cela, utiliser la commande `iw list` :

```
iw list
## Wiphy phy1
##         ...
##         Supported interface modes:
##                  * managed
##           * AP
##                  * monitor
```

Souvent, les cartes WiFi sont utilisées en mode *managed*, pour se connecter à un point d'accès existant. Nous devons donc passer la carte en mode *monitor* sur le principe suivant [2] :

```
# ============================
#    Passage en mode monitor
# ============================
iw dev
## phy#1
##         Interface wlan1
##                 ...
##                 type managed
iw wlan1 del
iw dev
iw phy1 interface add mon1 type monitor
## phy#1
##         Interface mon1
##                 ...
##                 type monitor
ip link set mon1 up

# ============================
#    Passage en mode managed
# ============================
iw mon1 del
iw phy1 interface add wlan1 type managed
## phy#1
##         Interface wlan1
##                 ...
##                 type managed
ip link set wlan1 up
```

1. `https://github.com/samyk/skyjack`
2. À exécuter en `root`

Il est nécessaire de désactiver tous les processus gérant la carte WiFi avant d'effectuer cette manipulation, notamment `NetworkManager` (`systemctl stop network-manager`).

Plutôt que d'utiliser les commandes précédentes, il est possible d'utiliser `airmon-ng` :
- `airmon-ng check kill` permet de tuer les processus utilisant la carte WiFi
- `airmon-ng start wlan0` permet de passer la carte en mode *monitor*
- `airmon-ng stop wlan0mon` permet d'arrêter le mode *monitor*

1 Découverte des *Access Points* WiFi environnants

1.1 Quelques éléments de théorie

Pour découvrir les réseaux WiFi disponibles, deux méthodes s'offrent à nous :

1. Effectuer un scan passif
2. Effectuer un scan actif des réseaux

Le scan passif est une simple écoute de paquets WiFi du type *Beacon frames*. Ceux-ci sont envoyés par les *Access Points* pour signaler leur présence. Ils contiennent notamment les informations suivantes :

- *channel* : fréquence sur laquelle le paquet est reçu
- *BSSID* : adresse MAC de l'*Access Point* (AP)
- *ESSID* : nom du réseau WiFi (abrégé SSID)

Remarque : pour changer le *channel* de la carte WiFi, il est nécessaire d'exécuter :
`iw mon1 set channel <CHANNEL>`

Le scan actif consiste en l'envoi de paquets *Probe Request* en indiquant un SSID. Si l'AP correspondant à cet SSID reçoit le paquet, il répondra avec un *Probe Response*. Le client saura alors que l'AP est à proximité. Si le SSID du *Probe Request* est `null`, alors l'ensemble des AP à proximité répondent avec un paquet *Probe Response*.

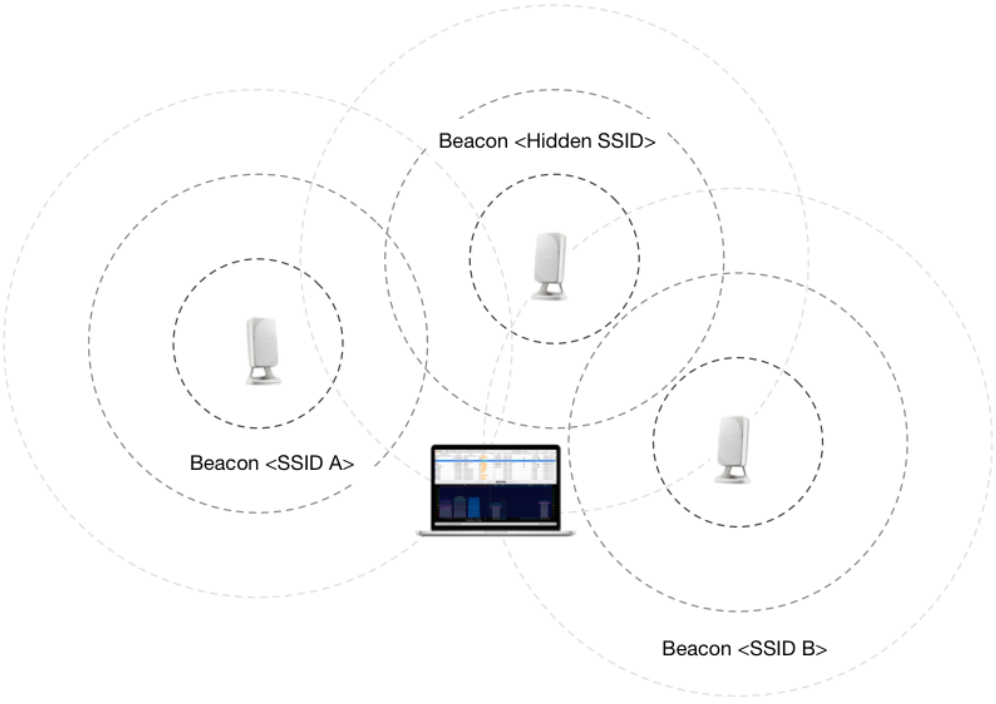

Figure 1 – Scan passif[3]

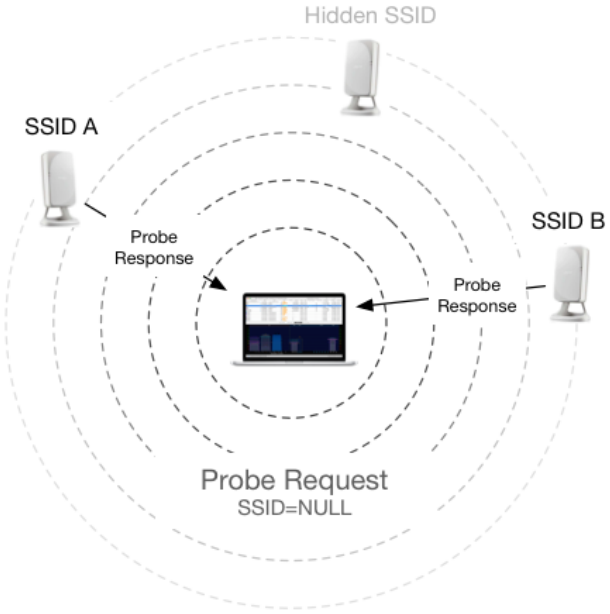

Figure 2 – Scan actif[4]

1.2 Mise en pratique

Dans le but de faire que notre attaque reste inaperçue un maximum de temps, nous avons choisi d'effectuer un scan passif. Le script Python `main.py` permet de lister passivement les AP WiFi aux alentours. La bibliothèque Scapy (`https://scapy.net/`) est utilisée, notamment les fonctions `sniff` pour l'écoute des paquets et `Dot11Beacon.network_stats` pour obtenir les informations sur le réseau.

À vous de jouer :

- Si vous comparez la liste des réseaux WiFi produite par votre téléphone avec celle produite par le script, vous pourrez observer des différences. Quelles sont elles ?
- Corrigez le script `main.py` pour lister l'ensemble des réseaux à proximité. Pour lancer le script, vous pouvez utiliser `./main.py mon1 -l`

Avec Python 3, le code fourni risque de ne pas fonctionner. En effet, il faut que votre version installée de Scapy intègre le correctif du bug 1872 (`https://github.com/secdev/scapy/issues/1872`). Le correctif est déjà installé sur le PC de démo.

Quelques précisions sur la fonction `sniff(iface=, timeout=, lfilter=, prn=)` :

- Cette fonction permet d'écouter des paquets réseaux sur une interface précise (`iface`).
- L'argument `timeout` spécifie un temps d'expiration en seconde.
- L'argument `prn` est une fonction (*callback*) appelée lorsqu'un paquet est reçu.
- L'argument `lfilter` est un *callback* renvoyant un booléen, permettant de filtrer les paquets transmis à `prn` (`True` pour transmettre le paquet, `False` pour le jeter).

Le script `ex1.py` donne la correction.

3. Schéma pris de `https://www.adriangranados.com/blog/understanding-scan-modes-wifiexplorerpro`
4. Schéma pris de `https://www.adriangranados.com/blog/understanding-scan-modes-wifiexplorerpro`

2 Détection des clients d'un réseau

À vous de jouer :

— Nous souhaitons maintenant lister les clients de l'AP (adresses IP et MAC). Quelle technique proposeriez-vous, toujours dans l'optique que notre attaque ne soit détectée qu'au dernier moment ? Implémenter votre solution.
— Pour lancer le script, vous pouvez utiliser `./main.py mon1 -c <BSSID> <CHANNEL>`

Indices :

— Vous pouvez utiliser à nouveau la fonction `sniff` de Scapy.
— Vous pouvez utiliser les fonctions *lambda* (fonctions anonymes) de Python (voir plus bas).
— La méthode `packet.haslayer` permet de vérifier qu'un paquet réseau présente une couche précise (`IP`, `TCP`, `Dot11FCS`, etc.). Pour récupérer une couche précise associée à un paquet, il faut procéder comme suit : `packet[couche]` (par exemple : `packet[TCP]`).
— Les champs suivants peuvent être utiles sur un paquet WiFi :
  — `packet.addr1` : adresse MAC destination
  — `packet.addr2` : adresse MAC source
  — `packet.addr3` : BSSID
  — `packet[IP].dst` : adresse IP destination
  — `packet[IP].src` : adresse IP source

  Remarque : pour les deux derniers, il faut que la couche `IP` soit présente.

Topo sur les fonctions anonymes en Python :

Pour définir une fonction anonyme en Python, il faut utiliser le mot clé `lambda` [a]. On peut ensuite appeler la fonction comme une autre. Par exemple :

```
1  x = lambda a : a + 10
2  print(x(5))
```

Ceci est particulièrement utile lorsqu'une fonction attend en argument un *callback* :

```
1  def my_filter(packet, layer):
2      """ Filter packet on layer. """
3      return packet.haslayer(layer)
4
5  # Print all packets with IP layer
6  sniff(
7          lfilter = lambda p: my_filter(p, IP),
8          prn     = lambda p: p.show()
9        )
```

a. `lambda arg1, arg2, ...: value(arg1, arg2)`

Le script `ex2.py` donne la correction.

3 Désauthentification d'un client avec `aireplay-ng`

Nous avons maintenant la liste des clients d'une AP précise. Sur l'AP du drone Parrot, on peut supposer qu'il n'y a qu'un seul client : le pilote du drone. Nous devons le déconnecter du réseau pour prendre le contrôle du drone à sa place.

Pour cela, il nous suffit d'envoyer des trames de désauthentification à l'AP. Dans le cas d'un WiFi ouvert, c'est-à-dire dans notre cas, on parle de trames de désassociation. L'AP croira alors que le client n'est pas associé au réseau et refusera le futur trafic de celui-ci. Le schéma de la page suivante présente le principe.

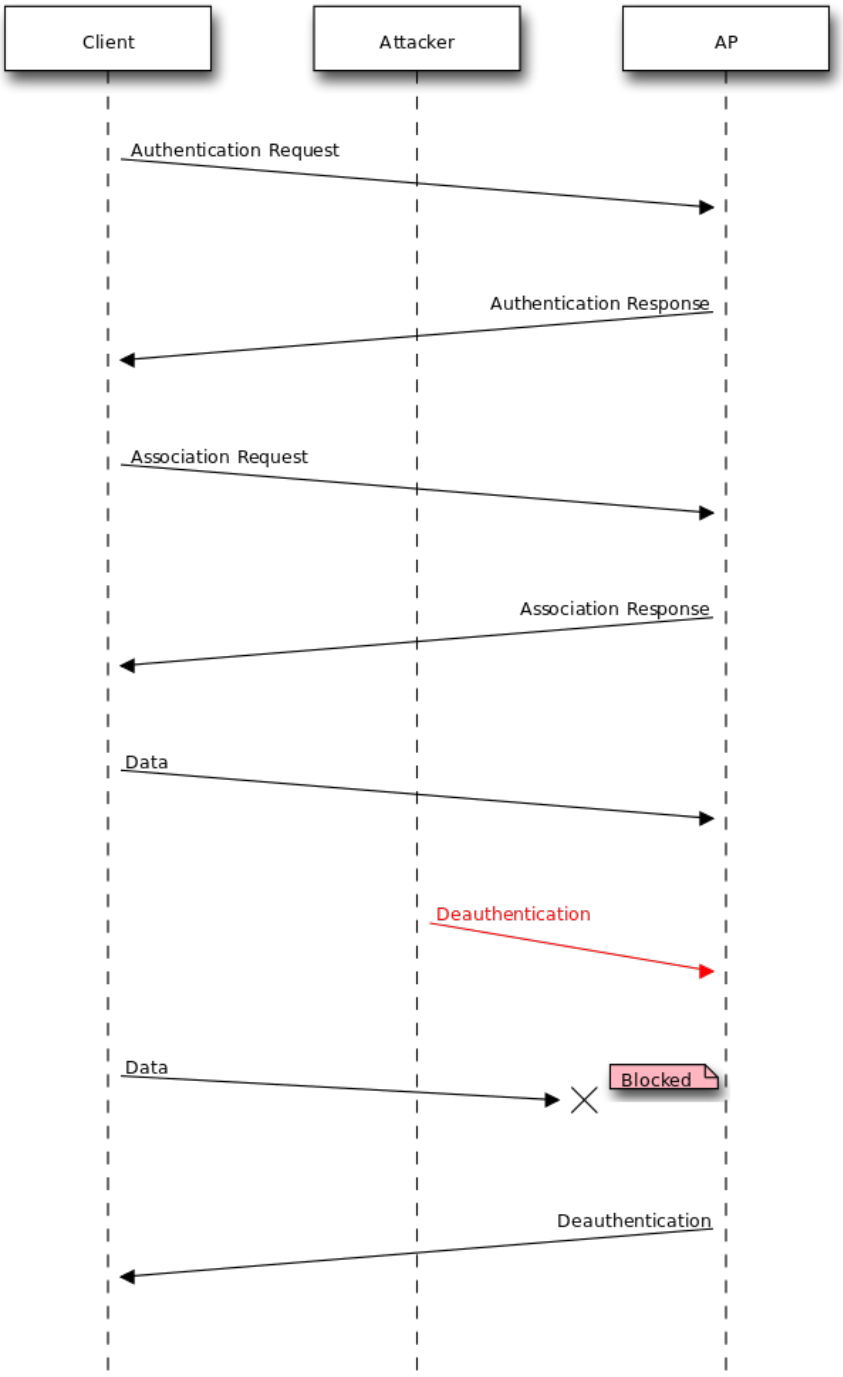

Figure 3 – Attaque par désauthentification WiFi[5]

À vous de jouer :

— Pour perpétrer l'attaque de désauthentification WiFi, on peut utiliser `aireplay-ng`. Désauthentifiez le client pilotant le drone avec `aireplay-ng` dans le script Python.

— Pour lancer le script, vous pouvez utiliser `./main.py mon1 -d <CLIENT_MAC> <BSSID> <CHANNEL>`

5. Schéma pris sur `https://en.wikipedia.org/wiki/Wi-Fi_deauthentication_attack`

Indices :

— Étudiez le manuel de `aireplay-ng` (`man aireplay-ng`).
  — Les options `-c`, `-0`, `-a` peuvent être utiles.
  — L'option `-0` permet d'indiquer le nombre de paquets de désassociation à envoyer. 3 est une bonne valuer.
— Commencez par construire la ligne de commande en dehors du script Python.

Le script `ex3.py` donne la correction.

Le pilote officiel du drone est maintenant déconnecté de l'AP. Nous pouvons "prendre sa place" en se connectant à celle-ci. Ceci peut être fait en ligne de commande par `iw wlan0 connect ardrone2_xxxxxx` où `ardrone2_xxxxxx` est le SSID de l'AP. Il suffit alors de demander une adresse IP au drone (`dhclient wlan0`) et de lancer un programme controlant le drone, tel que `ardrone-webflight` [6].

Pour se connecter à l'AP, il est nécessaire que la carte WiFi soit en mode *managed*. Ainsi, nous vous conseillons d'avoir une seconde carte WiFi pour perpétrer réellement l'attaque :

— La première, en mode *monitor*, permet de lister les APs, leurs clients, et de désassocier le client pilote.
— La seconde, en mode *managed*, permet de se connecter au drone et d'en prendre le contrôle.

Ce n'est pas fait dans le cadre de cet exercice.

4 Bonus : filtrage des *Access Points* Parrot

Avec le script précédent, l'ensemble des réseaux WiFi est listé. Cependant, nous souhaitons prendre le contrôle d'un drone Parrot et cherchons donc seulement les APs Parrot.

Quelle solution proposeriez-vous pour cela ?

Une solution est d'effectuer un filtrage des adresses MAC des APs (BSSID). En effet, les adresses MAC sont sous la forme `3A:34:52:C4:69:B8`, où :

— la première partie (`3A:34:52`) est l'*Organizationally Unique Identifier* (OUI)
— la seconde partie (`C4:69:B8`) est le *Network Interface Controller* (NIC)

Ainsi, en filtrant les BSSID sur leur OUI, il est possible de ne lister que les APs Parrot.

À vous de jouer :

— Améliorer le script de l'étape 1 pour ne lister que les APs Parrot. Les blocs d'adresses MAC alloués pour Parrot sont : `90:03:B7`, `00:12:1C`, `90:3A:E6`, `A0:14:3D`, `00:12:1C`, `00:26:7E`.

Indice :
Utilisez la notation `string[:<int>]` où `<int>` est le nombre de caractères à extraire depuis le début de la chaîne de caractères.

Le script `ex4.py` donne la correction.

6. `https://github.com/eschnou/ardrone-webflight`