

---

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

Tuto sécu, THCon 2019

Florent Fayollas et Antoine Vacher

---

## Introduction

L'objectif de ce tutoriel est de vous faire une introduction à la bibliothèque Python Scapy dans un cadre de prise de contrôle d'un drone Parrot AR.Drone 2.0, contrôlé par WiFi. L'attaque originelle est nommée SkyJack<sup>1</sup> 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` :

```
1 iw list
2 ## Wiphy phy0
3 ##
4 ##      Supported interface modes:
5 ##              * managed
6 ##              * AP
7 ##              * 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<sup>2</sup> :

```
1 # =====
2 #      Passage en mode monitor
3 # =====
4 iw dev
5 ## phy#0
6 ##      Interface wlan0
7 ##      ...
8 ##      type managed
9 iw wlan0 del
10 iw dev
11 iw phy0 interface add mon0 type monitor
12 ## phy#0
13 ##      Interface mon0
14 ##      ...
15 ##      type monitor
16 ip link set mon0 up
17
18 # =====
19 #      Passage en mode managed
20 # =====
21 iw mon0 del
22 iw phy0 interface add wlan0 type managed
23 ## phy#0
24 ##      Interface wlan0
25 ##      ...
26 ##      type managed
27 ip link set wlan0 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.

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)

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*.

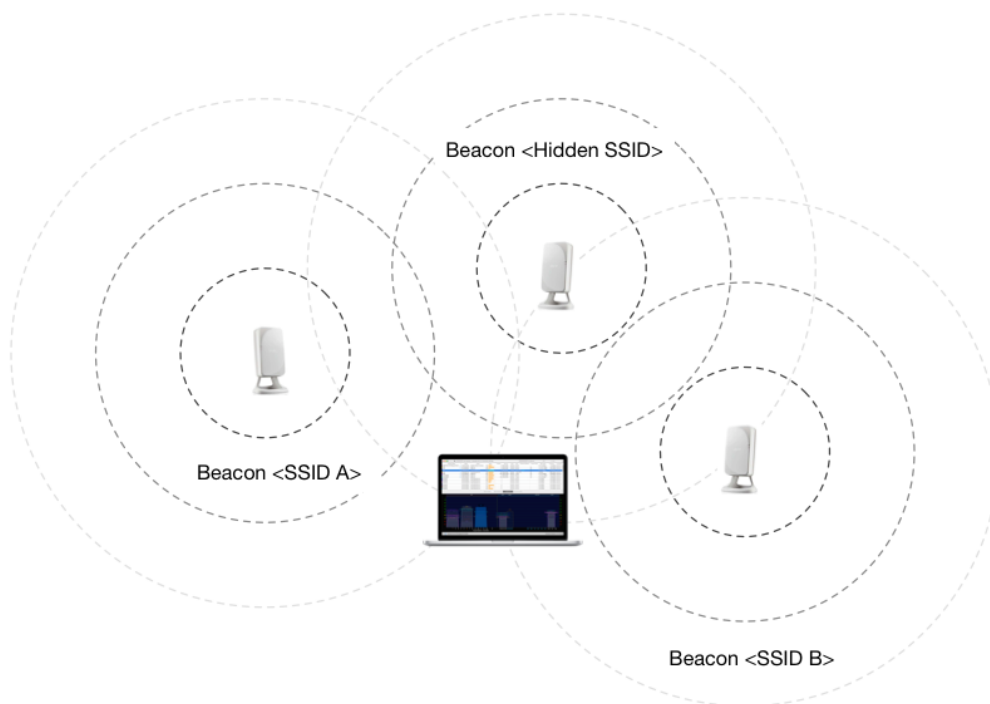
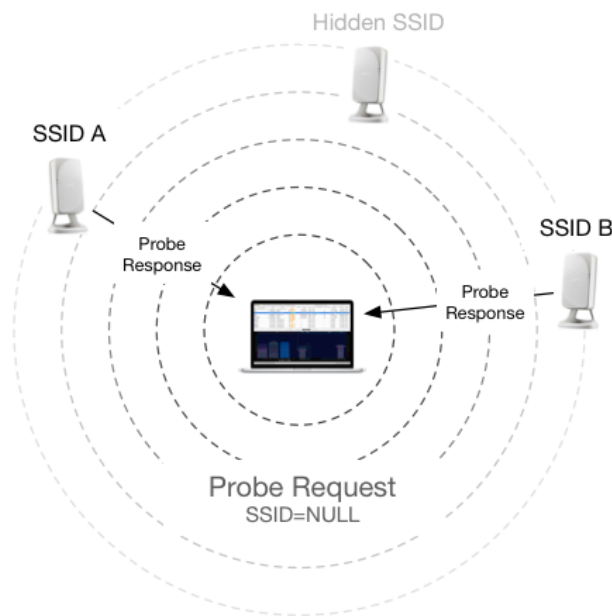


Figure 1 – Scan passif<sup>3</sup>

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

**Figure 2** – Scan actif<sup>4</sup>

## 1.2 Mise en pratique

Dans le but de faire que notre attaque reste inaperçu 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.

Avec Python 3, le code précédent 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>).

### 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 pour lister l'ensemble des réseaux à proximité.

Le script `ex1.py` donne la correction.

## 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 ? Implanter votre solution.

### Indices :

- Vous pouvez utiliser à nouveau la fonction `sniff` de Scapy.
- La méthode `packet.haslayer` permet de vérifier qu'un paquet réseau présente une couche précise (IP, TCP, Dot11, etc.). Pour récupérer une couche précise associée à un paquet, il faut procéder comme suit : `packet[couche]` (par exemple : `packet[TCP]`).

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

- Les champs suivants peuvent être utiles :
  - `dot11_packet.addr1` : adresse MAC destination
  - `dot11_packet.addr2` : adresse MAC source
  - `dot11_packet.addr3` : BSSID
  - `ip_packet.dst` : adresse IP destination
  - `ip_packet.src` : adresse IP source

Le script `ex2.py` donne la correction.

### 3 Désauthentification d'un client avec *aireplay-ng*

#### À vous de jouer :

Nous avons maintenant la liste des clients d'une AP précise. Désauthentifiez le client pilote avec *aireplay-ng* dans le script Python.

Le script `ex3.py` donne la correction.

Le pilote officiel 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 contrôlant le drone, tel que `ardrone-webflight`.<sup>5</sup>

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.

### 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`.

Le script `ex4.py` donne la correction.

5. <https://github.com/eschnou/ardrone-webflight>