TLS-SEC

Drone predator, *one drone tu rule them all*

Rapport de projet long TLS-SEC

Florent Fayollas et Antoine Vacher, promotion 2018 – 2019

INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE
INP TOULOUSE ENSEEIHT
ENAC ECOLE NATIONALE DE L'AVIATION CIVILE

Table des matières

1 Introduction

1.1 Introduction généraliste sur les drones

Les UAVs (*Unmanned Aerial Vehicles*), communément appelés drones, sont des aéronefs sans pilote humain à bord. Un drone est un composant d'un UAS (*Unmanned Aircraft System*), qui comprend un UAV, une station de contrôle au sol et un system de communication entre les deux.

Les UAVs étaient, à l'origine, développés par les militaires et utilisés pour des missions trop dangereuses pour les humains. Cependant, ces dernières années, leur utilisation s'est généralisée à beaucoup de secteurs, tels que l'industrie ou les loisirs. Des exemples concrets d'utilisations sont la surveillance de réseaux électriques EDF ou des voies ferrées ou la photographie aérienne.

Leur champ d'utilisation est en croissance continue. En effet, les drones pourraient nous aider lors de scénarios de secours, par exemple, en étant utilisés par les pompiers pour un suivi en temps réel par images thermiques d'un feu. En outre, certaines entreprises, telles qu'Amazon, envisagent d'effectuer des livraisons par drone.

1.2 Motivations de ce travail

Bien que les précédents cas d'usage civils soient bénéfiques, d'autres usages peuvent exister… En effet, un cas concret concernant tout le monde est celui d'une personne qui utiliserait un drone pour vous espionner, en survolant votre maison et passant proche de vos fenêtres. Une autre utilisation, plus grave, est l'utilisation, par Daesh, de drones civils pour larguer des bombes sur le front en Syrie. Enfin, plus récemment, l'aéroport de Londres Gatwick a été fermé pour cause de survols répétés par un drone non identifié.

Ainsi, il devient nécessaire de pouvoir se protéger des drones. Du côté militaire, des solutions existent déjà, telles que le brouillage de la liaison de commandes, par exemple grâce à des sortes de fusils, comme le DroneGun développé par DroneShield [1]. Cependant, très peu de solutions sont présentes côté civil.

On pourrait penser à implémenter les solutions militaires dans le civil. Or, celles-ci mettent souvent hors d'état de nuire le drone, peu importe l'état final (éventuellement un *crash*), ce qui n'est pas acceptable dans un cadre civil.

1.3 Définition des objectifs

L'objectif de notre projet est donc de développer un outil capable de prendre le contrôle de plusieurs drones. Cette prise de contrôle ne doit pas impliquer une chute du drone piraté. Cet outil pourrait ensuite être utilisé, par exemple, pour sécuriser la médiatisation d'un match sportif : il sera en mesure de "capturer" les drones détectés et de les récupérer. Après capture, il sera aussi possible d'analyser le contenu des drones et éventuellement de remonter au propriétaire.

Pour réaliser cet outil, nous nous sommes focalisés sur la prise de contrôle de drones civils : le Parrot AR.Drone 2.0 et le Syma X5C-1. C'est ce que nous détaillons dans les deux premières parties de ce rapport. Lorsque ces prises de contrôle furent terminées, nous avons cherché à embarquer l'outil sur un autre drone. Ceci est discuté dans la troisième partie de ce rapport.

1. `https://www.droneshield.com/`

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

Le drone Parrot AR.Drone 2.0 est paru sur le marché en janvier 2012. Il s'agit d'un drone hélicoptère quadrirotor développé par la société française Parrot SA. Il est principalement dédié au divertissement et peut se piloter avec un appareil sous iOS, Android ou Symbian (téléphones Nokia) via une liaison Wi-Fi. Dans cette liaison, le drone joue le rôle de point d'accès WiFi (*access point*, ou AP) et l'appareil de l'utilisateur est un client de ce réseau ouvert non sécurisé.

Figure 1 – Un drone Parrot AR.Drone 2.0

En décembre 2013, Samy Kamkar a publié une attaque sur ce drone. Cette attaque, nommée Skyjack [2], permet de prendre le contrôle de l'AR.Drone 2.0 et consiste simplement :

1. Déconnecter le pilote réel du drone du réseau WiFi ;
2. Se connecter au réseau WiFi ouvert du drone ;
3. Lancer un programme permettant de contrôler le drone grâce à la machine ayant perpetré l'attaque.

Nous nous sommes basés sur cette attaque pour prendre le contrôle du drone.

2.1 Présentation technique de l'attaque existante

L'attaque de Samy Kamkar consiste en un simple script Perl effectuant les actions suivantes :

1. Listing des AP WiFi alentours (et de leurs éventuels clients) grâce au programme `airodump-ng`.
2. Filtrage des *access points* ayant une adresse MAC de fabricant Parrot SA.
   Ce filtrage nous permet d'obtenir une liste de drone Parrot alentours.
3. Pour chaque drone ayant un client :
   1. Déconnexion du vrai pilote du drone avec le programme `aireplay-ng`
   2. Connexion au réseau WiFi du drone, ce qui implique la prise de contrôle du drone
   3. Lancement d'un programme de pilotage du drone

Cette attaque est perpétrée grâce à deux cartes WiFi de l'attaquant :

— La première carte, est en mode *monitor*, dans le but de pouvoir écouter l'ensemble du trafic WiFi aux alentours, ainsi qu'émettre les trames de désauthentification du vrai pilote. Cette carte est donc utilisée pour les étapes 1 et 3.1.

— La seconde carte, est en mode *managed*[3] et sert pour se connecter au réseau WiFi du drone lorsque le vrai pilote est déconnecté. Elle est donc utilisée lors des étapes 3.2 et 3.3.

2. `https://samy.pl/skyjack/`
3. Aussi appelé mode infrastructure

2.2 Implémentation de notre variante d'attaque

Dans l'optique d'embarquer notre outil sur un drone, nous avons cherché à réduire le poids du matériel à embarquer et les dépendances du code envers d'autres outils tels que `airodump-ng` et `aireplay-ng`. Ainsi, nous avons décidé d'implémenter une variante de cette attaque en Python 3, n'utilisant qu'une seule carte WiFi, contrairement à l'attaque de Samy Kamkar, dans laquelle deux cartes sont utilisées.

2.2.1 Premier script de déconnexion d'un client

Dans un premier temps, nous avons rédigé le script ci-dessous. Il s'agit d'un script Bash minimal et fonctionnel reproduisant l'attaque de Samy Kamkar avec deux cartes WiFi. Celui-ci suppose que la découverte des réseaux WiFi a déjà été faite :

— Le *channel* 6 est celui du réseau WiFi.
— `ardrone2_025774` est le SSID du réseau WiFi du drone piraté.
— L'adresse MAC `90:03:b7:c8:68:d0` est celle du drone piraté.
— L'adresse MAC `60:45:cb:22:11:9f` est celle du vrai pilote.

```
1  #! /bin/sh
2  #
3  # Disconnect real client
4  iw mon0 set channel 6
5  echo "Sending deauth"
6  aireplay-ng -0 3 -a 90:03:b7:c8:68:d0 -c 60:45:cb:22:11:9f mon0 >/tmp/aireplay-ng
7  #
8  # Connect to drone
9  echo "Waiting for deauth"
10 sleep 2
11 #
12 iw wlan0 connect ardrone2_025774
13 echo "Connecting"
14 sleep 0.5
15 while [ "$(iw wlan0 link)" = "Not connected." ]; do
16   sleep 0.5
17   echo "Not connected..."
18 done
19 #
20 echo "Connected"
21 dhclient -v wlan0
22 echo "Running drone controler software"
23 node app.js
```

L'écriture de ce script nous a permis de définir les informations nécessaires pour pouvoir perpétrer l'attaque de manière automatisée, sans utiliser `airodump-ng` :

— Le *channel* WiFi, c'est-à-dire la fréquence sur laquelle se trouve le réseau ;
— Le BSSID, c'est-à-dire l'adresse MAC de l'AP ;
— Le SSID, c'est-à-dire le nom du réseau ;
— L'adresse MAC du client à déconnecter.

2.2.2 Découverte des drones environnants et de leur pilote

Pour découvrir les drones environnants, du fait que les drones étaient associés à des points d'accès WiFi, deux méthodes s'offraient à nous :

1. **Effectuer un scan passif des réseaux WiFi.** Il s'agit d'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*, BSSID et SSID.
2. **Effectuer un scan actif des réseaux WiFi.** 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`, l'ensemble des AP à proximité répondent avec un paquet *Probe Response*.

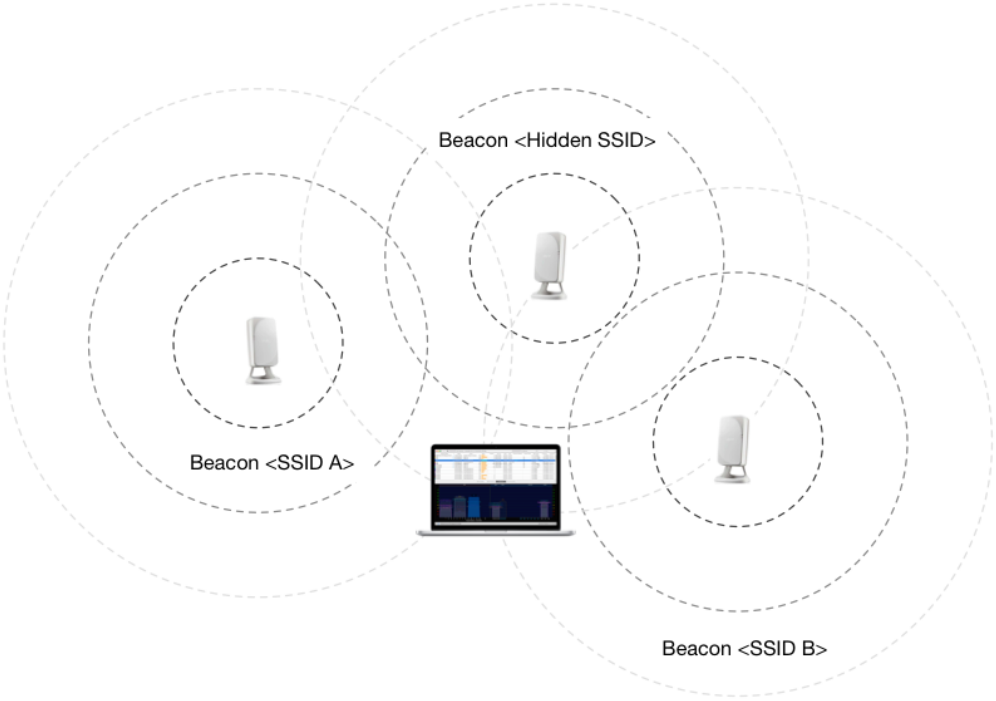

Figure 2 – Scan passif[4]

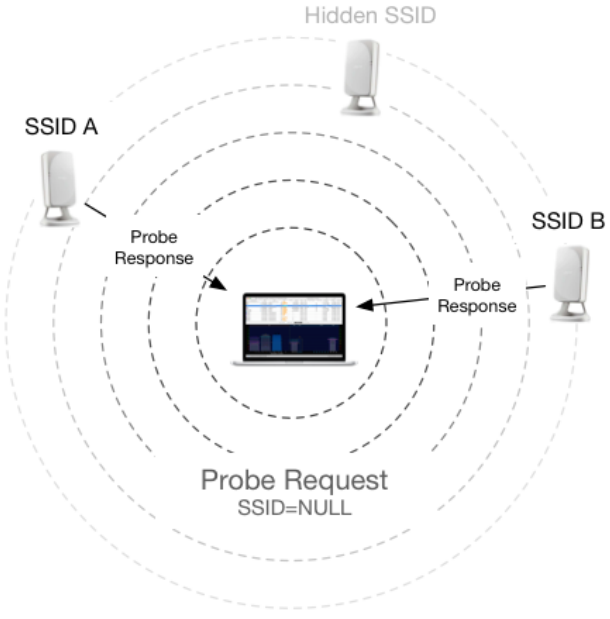

Figure 3 – Scan actif[5]

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

Nous avons choisi d'effectuer un scan passif pour que l'attaque reste inaperçue le plus longtemps possible. Ainsi, pour développer notre script listant les APs Parrot environnants, nous avons utilisé la fonction `sniff` de la bibliothèque Python Scapy. Cette fonction nous a permis d'écouter l'ensemble des paquets WiFi. Nous avons ensuite filtré les *Beacon frames* et en avons extrait des informations importantes concernant les réseaux WiFi environnants : *channel*, SSID et BSSID. Pour extraire ces informations, nous avons utilisé la fonction `network_stats` introduite en décembre 2018 [6]. À partir de la liste d'AP obtenue, le script effectue ensuite un filtrage des APs Parrot grâce à leur adresse MAC (BSSID).

Il ne nous manquait alors que l'adresse MAC du client pour pouvoir perpétrer l'attaque de manière automatisée. Nous avons donc utilisé à nouveau la fonction `sniff` de Scapy afin d'écouter les paquets IP transitant sur le *channel* obtenu précédemment. Il nous a suffit de filtrer les paquets WiFi (couche `Dot11FCS` dans Scapy) reçus en ne traitant que les paquets dont le `addr3` était notre BSSID et pour lesquels une couche IP existait.

En effet, les paquets WiFi comporte 3 champs intéressants lors de l'écoute :

— `addr1` qui contient l'adresse MAC destination ;
— `addr2` qui contient l'adresse MAC source ;
— `addr3` qui contient la BSSID du réseau sur lequel le paquet transite.

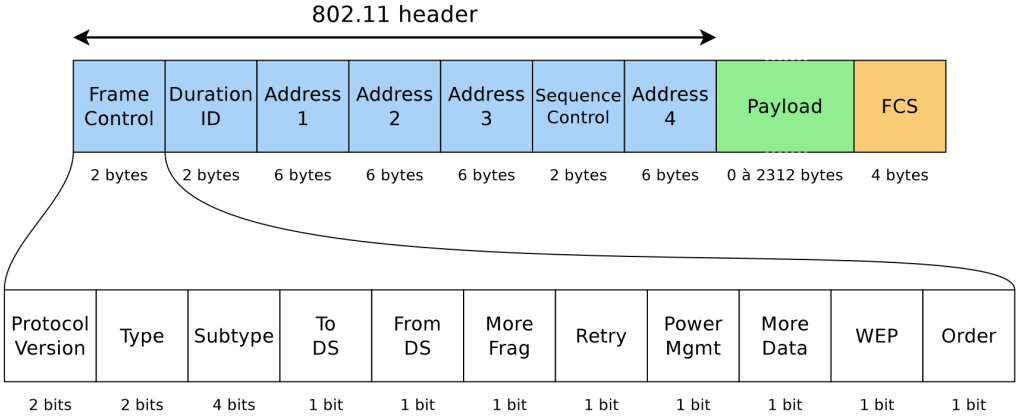

Figure 4 – Format d'un paquet WiFi 802.11

2.2.3 Intégration de la déconnexion avec `aireplay-ng` du pilote réel

Ainsi, notre script Python permettait de lister les réseaux WiFi des drones environnants, ainsi que leur · s client · s. Nous avons alors amélioré le script pour afficher un menu dans le cas où plusieurs drones seraient détectés (voir page suivante). Nous avons ensuite intégré la déconnexion du client détecté en appelant le programme `aireplay-ng`.

2.2.4 Choix d'un programme de pilotage du drone

Enfin, notre script devait permettre le pilotage du drone. Pour cela, nous avons cherché des programmes existants et avons trouvé les solutions suivantes :

— Le programme AutoFlight [7], permettant de contrôler des drones Parrot. Nous n'avons pas retenu cette solution car elle nécessitait une interface graphique sur l'ordinateur exécutant, ce qui nous limiterait sûrement lors de l'embarquement de notre outil sur un drone pirate.
— Les bibliothèques Python PS-Drone [8] et `python-ardrone` [9]
— L'interface `ardrone-webflight` [10], basée sur NodeJS et proposant un contrôle du drone par navigateur web, au clavier, ou avec une manette.

6. Cette fonction comportait un bug en Python 3, que nous avons fait remonté sur `https://github.com/secdev/scapy/issues/1872` et pour lequel nous avons proposé un correctif.
7. `http://electronics.kitchen/autoflight`
8. `https://www.playsheep.de/drone/`
9. `https://github.com/venthur/python-ardrone`
10. `http://eschnou.github.io/ardrone-webflight/`

Nous avons opté pour la solution d'`ardrone-webflight` car il s'agissait d'une solution clé en main, avec une interface attractive. En outre, ce programme pourrait fonctionner de manière embarquée sur le drone pirate, puisque le contrôle du drone se fait par navigateur web.

Figure 5 – Menu de l'outil `predator-drone`

2.2.5 Utilisation de Scapy pour déconnecter le client

Toujours dans l'optique d'embarquer, à terme, notre outil, nous avons cherché à remplacer notre utilisation de `aireplay-ng` par Scapy. Pour cela, nous avons tout d'abord étudier le fonctionnement de `aireplay-ng`.

Une attaque par désauthentification WiFi fonctionne en envoyant un certain nombre de trame de désauthentification à l'AP et au client. Ainsi, les deux entités vont enregistrer qu'elles ne sont plus associées entre elles sur le réseau WiFi. Tout le trafic qui suivra, provenant d'une entité vers l'autre sera alors jeté, comme le montre le schéma suivant :

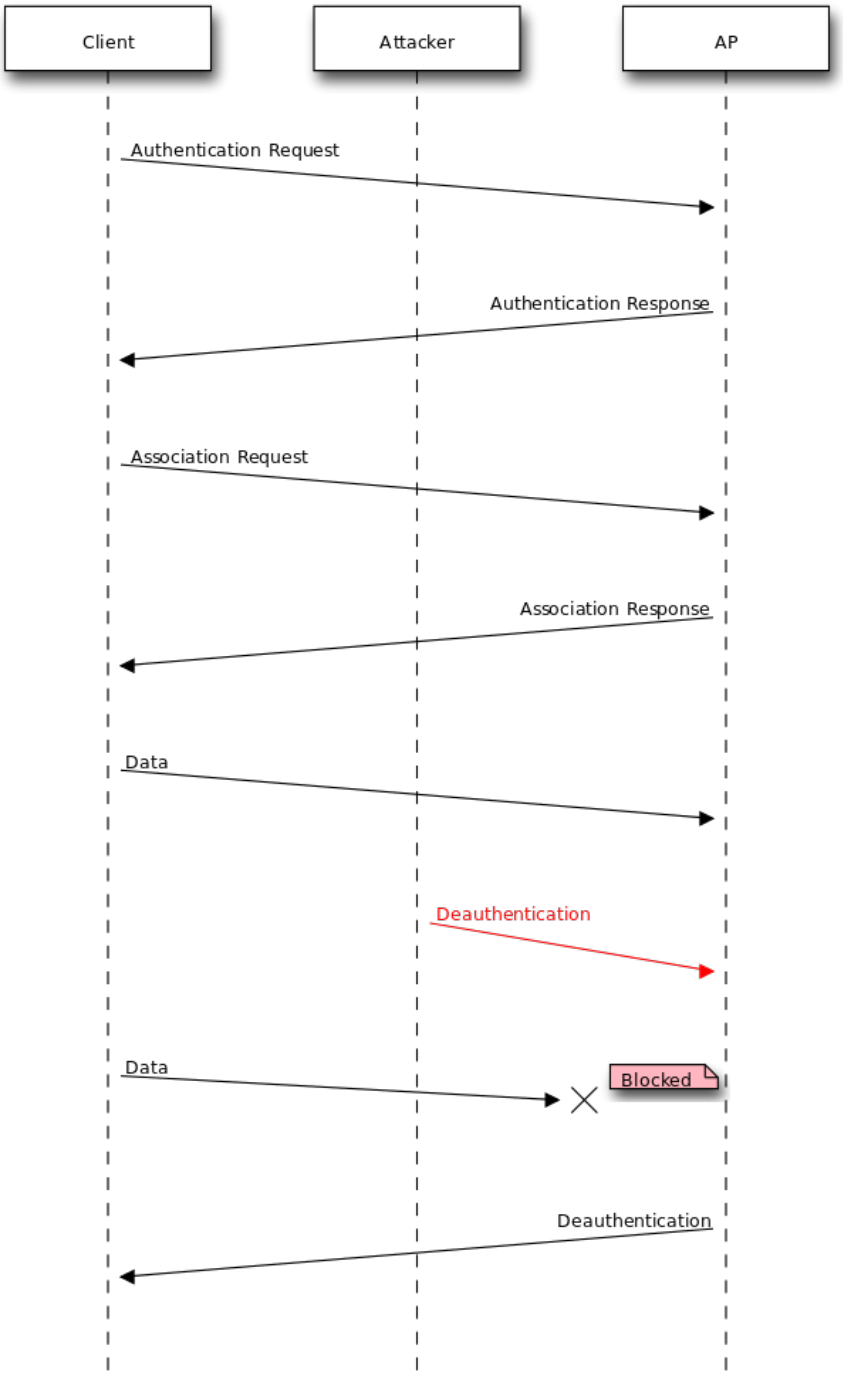

Figure 6 – Attaque par désauthentification WiFi [11]

`aireplay-ng` permet de perpétrer une telle attaque. Dans le but de "copier" au maximum le comportement de ce programme, nous avons perpétré l'attaque et effectué une capture réseau grâce à Wireshark. Nous avons pu remarquer que 64 trames de désauthentification étaient envoyées à l'AP, et 64 autres étaient envoyées au client. Le numéro de séquence des trames était incrémenté à chaque envoi. En outre, le `reason code` des trames était fixé à 7, ce qui signifie "*Class 3 frame received from nonassociated STA.*", autrement dit qu'un paquet de données a été reçu d'un émetteur qui ne devrait pas être sur le réseau.

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

Basé sur ces observations, nous avons développé le code suivant, supposé avoir le même comportement :

```
# Build deauth packets: addr1=dst, addr2=src, addr3=bssid
deauth_client = RadioTap() \
    / Dot11(addr1=ap.bssid, addr2=client.mac, addr3=ap.bssid) \
    / Dot11Deauth(reason=7)

deauth_ap = RadioTap() \
    / Dot11(addr2=ap.bssid, addr1=client.mac, addr3=ap.bssid) \
    / Dot11Deauth(reason=7)

# Send packets, updating sequence number
pkt_count = 3*64*2
for i in range(0, pkt_count, 2):
    deauth_client.SC = i << 4
    deauth_ap.SC     = (i + 1) << 4

    sendp(deauth_client, iface=self.card.dev, verbose=False)
    sendp(deauth_ap,     iface=self.card.dev, verbose=False)

print("Sent", str(pkt_count), "deauth packets")
```

> **Remarque :** le numéro de séquence du paquet (champ `SC`) est constitué de :
> — 4 bits de numéro de fragment
> — 12 bits de numéro de séquence
>
> C'est pour cela qu'un décalage à gauche de 4 bits est effectué.

Cependant, ce code n'a pas fonctionné. Nous avons pu observer sur une capture Wireshark que certaines des trames de désauthentification étaient envoyés plusieurs fois. Après quelques heures recherches, nous n'avions toujours pas trouvé d'explication valable. Nous avons donc décidé de conserver le code avec `aireplay-ng`.

> **Remarque :** lorsque nous avons souhaité embarquer notre outil sur une Raspberry Pi Zero W (voir plus loin), nous avons été confrontés à des problèmes de *drivers* de cartes WiFi. Ceci pourrait expliquer l'envoi répété des paquets. Cette découverte peu avant la fin du projet, nous n'avons pas eu le temps de la confirmer/infirmer.

2.2.6 Manipulation de la carte WiFi par notre programme

Jusqu'ici, nous considérions que l'utilisateur de notre outil avait déjà configuré une de ses cartes WiFi en mode *monitor*. Cependant, cette configuration nécessitant quelques connaissances, nous avons souhaité intégrer le passage en mode *monitor* directement le script. En outre, effectuer cela était une étape nécessaire vers le passage de notre outil à une utilisation d'une unique puce WiFi.

Pour effectuer cette configuration automatique, nous avons utilisé la bibliothèque Python `PyRIC` [12]. L'utilisation de celle-ci a un peu ralenti l'initialisation de notre script, mais pas son exécution.

2.2.7 Utilisation d'une unique puce WiFi

Enfin, nous avons cherché à n'utiliser qu'une puce WiFi pour perpétrer l'attaque. L'objectif était alors de réduire le matériel embarqué sur le drone prédateur.

Ainsi, l'algorithme haut niveau de l'attaque était :

1. Passage de la puce WiFi en mode *monitor*
2. Découverte d'un drone Parrot et de son pilote
3. Désauthentification du client (pilote réel)
4. Passage de la puce WiFi en mode *managed*
5. Connexion au réseau WiFi du drone piraté
6. Lancement du cockpit de pilotage de l'AR.Drone

12. `https://github.com/wraith-wireless/PyRIC`

Cependant, après de multiples tentatives, nous n'avons pas réussi à faire fonctionner l'attaque comme cela : la désauthentification fonctionnait moins bien et durait moins longtemps : le client (pilote réel) se reconnectait rapidement. De plus, l'étape 5 bloquait souvent : il nous était impossible de nous connecter au réseau WiFi rapidement après le passage en mode *managed*. Lorsque nous arrivions à nous connecter, le client (pilote réel) s'était déjà reconnecté.

Lors des captures Wireshark que nous avons effectués pour déboguer, nous avons pu observer que des trames de désauthentification étaient encore envoyées même après le passage en mode *managed* de la puce, ce qui est normalement impossible.

Nous avons pensé qu'il pouvait s'agir de *buffer* non vidés qui seraient encore envoyés après le passage en mode *monitor*, et avons donc tenté d'ajouter des attentes. Cela n'a pas réglé le problème.

Monsieur Morgan nous a proposé de décharger le *driver* noyau de la puce WiFi puis de le recharger. Ainsi, la puce serait remise à zéro. Nous n'avons pas adopté cette solution car celle-ci nécessitait des informations trop précises sur la puce WiFi utilisée pour l'attaque et que nous souhaitions développer un outil utilisable simplement.

De fait, nous avons abandonné l'idée d'utiliser une unique puce WiFi pour perpétrer l'attaque. Il est tout de même important de noter que nous avons fait ce choix après avoir vérifié qu'utiliser deux puces WiFi n'augmenterait pas la charge utile portée par le drone prédateur. Ceci est détaillé plus loin.

2.2.8 Récapitulatif de notre variante d'attaque

Finalement, notre variante d'attaque présente les caractéristiques suivantes :

— Utilisation de deux cartes WiFi ;
— Utilisation de la bibliothèque `PyRIC` pour passer une des deux cartes WiFi en mode *monitor* ;
— Découverte passive des drones environnants et de leur pilote ;
— Attaque par désauthentification WiFi grâce à `aireplay-ng` ;
— Pilotage du drone capturé grâce à `ardrone-webflight`.

2.3 Détection et prévention de l'attaque

Concernant la détection de l'attaque, un trait caractéristique de celle-ci sont les trames de désauthentification envoyées. Lors de nos essais à la volière de l'ENAC, M. Ruohao Zhang, doctorant à l'ENAC, a effectué une capture Wireshark de notre attaque. Son objectif est de tenter une caractérisation du trafic nominal et du trafic contenant l'attaque grâce aux outils sur lesquels il travaille.

Pour prévenir notre attaque, nous avons pensé à plusieurs solutions :

1. **Utiliser la solution d'association entre pilote réel et drone** proposée par Parrot. Cette solution consiste à associer une adresse MAC au drone à l'aide de l'application. Cette association permet de bloquer le trafic non autorisé, mais est facilement contournable à l'aide d'une usurpation d'adresse MAC.
2. **Configurer le drone Parrot afin qu'il n'envoie plus de *Beacon frames*** pour signaler la présence de son AP. Cependant, nous pourrions alors faire une recherche active des réseaux, en envoyant des *Probe requests*.
3. **Utiliser un réseau WiFi chiffré**. Cette solution a été adoptée par Parrot pour sécuriser ses nouveaux drones. La prise de contrôle ne fonctionnerait plus (car il faudrait connaître le mot de passe du réseau), mais la désauthentification du pilote réel fonctionnerait toujours.

3 Prise de contrôle d'un drone Syma X5C-1

TODO

4 Embarquement de l'outil de prise de contrôle sur un drone prédateur

4.1 Installation de l'outil sur une Raspberry Pi Zero W

présentation vulgarisée

4.1.1 Utilisation du cockpit `ardrone-webflight`

déport par iptables

4.2 Problème de pilote de puce WiFi

TODO

4.2.1 Compilation de la bibliothèque `pyRF24`

compilation pyrf24

4.2.2 Utilisation distante d'une manette

usbip

4.2.3 Contrôle distant de l'outil

Bluetooth PAN

4.3 Tests finaux de l'outil embarqué

volière ENAC avec paparazzi + utilisation moyens volière (vidéo, etc.) + portée bluetooth

5 Conclusion

TODO