



TLS-SEC

**Drone predator,
*one drone to rule them all***

Rapport de projet long TLS-SEC

Florent Fayollas et Antoine Vacher, promotion 2018 – 2019

DRONE PREDATOR



ONE DRONE TO RULE THEM ALL

INSA

INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
TOULOUSE

INP
TOULOUSE

ENSEEIHT

ENAC

ÉCOLE NATIONALE DE L'AVIATION CIVILE

Table des matières

1 Introduction	3
1.1 Introduction généraliste sur les drones	3
1.2 Motivations de ce travail	3
1.3 Définition des objectifs	3
2 Prise de contrôle d'un drone Parrot AR.Drone 2.0	4
2.1 Présentation technique de l'attaque existante	4
2.2 Implémentation de notre variante d'attaque	5
2.2.1 Premier script de déconnexion d'un client	5
2.2.2 Découverte des drones environnants et de leur pilote	5
2.2.3 Intégration de la déconnexion avec <code>aireplay-ng</code> du pilote réel	7
2.2.4 Choix d'un programme de pilotage du drone	7
2.2.5 Utilisation de Scapy pour déconnecter le client	9
2.2.6 Manipulation de la carte WiFi par notre programme	10
2.2.7 Utilisation d'une unique puce WiFi	10
2.2.8 Récapitulatif de notre variante d'attaque	11
2.3 Détection et prévention de l'attaque	11
3 Prise de contrôle d'un drone Syma X5C-1	12
3.1 Présentation de l'attaque	12
3.2 Compréhension de la couche physique	12
3.2.1 Découverte des canaux utilisés	12
3.2.2 Découverte de la modulation	15
3.3 Compréhension de la couche liaison de données	17
3.3.1 Première impression	17
3.3.2 Module nRF24L01+	17
3.3.3 Détermination des paramètres du protocole nRF24L01+	18
3.4 Compréhension du protocole Syma	20
3.5 Implémentation du protocole	22
3.6 Caractérisation de l'attaque	22
3.7 Détection et prévention de l'attaque	23
4 Embarquement de notre outil sur un drone prédateur	24
4.1 Installation sur un Raspberry Pi Zero W	24
4.1.1 Contrôle distant de l'outil	24
4.1.2 Compilation de la bibliothèque <code>pyRF24</code>	24
4.1.3 Problème de pilote de puce WiFi	25
4.1.4 Utilisation du cockpit <code>ardrone-webflight</code>	25
4.1.5 Prise de contrôle du Syma X5C-1 et utilisation distante d'une manette	26
4.2 Tests finaux de l'outil embarqué	26
5 Étude du drone PNJ Discovery	28
5.1 Démarche	28
5.2 Identification des contrôleurs	28
5.2.1 Sur le drone	28
5.2.2 Sur la télécommande	29
5.3 Analyse du bus SPI	29
5.3.1 Connexion au bus	29
5.3.2 Analyse grâce à un analyseur logique	29
6 Conclusion	31
6.1 Résultats et limitations	31
6.2 Perspectives et améliorations	32
6.3 Compétences acquises	33

Remerciements

33

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 système 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, de 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 de loisirs 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 DroneShield¹. 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.

En effet, le crash du drone n'est pas forcément la meilleure solution à un survol non autorisé. On peut imaginer un drone non identifié survolant une foule : Il est dans ce cas préférable de l'éloigner plutôt que de le faire tomber.

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 de loisir : 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 réussies, nous avons cherché à embarquer l'outil sur un autre drone. Ceci est discuté dans la troisième partie de ce rapport. Enfin, nous avons commencé à nous intéresser à un troisième drone de loisir, le PNJ Discovery, pour lequel nous n'avons pas terminé de faire la rétro-conception dans le temps du projet. La quatrième partie de ce rapport décrit l'avancement partiel de cette tâche.

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 Skyjack², permet de prendre le contrôle de l'AR.Drone 2.0 et consiste simplement à :

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 sur 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 des 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://sam.y.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 ce 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 répondront avec un paquet *Probe Response*.

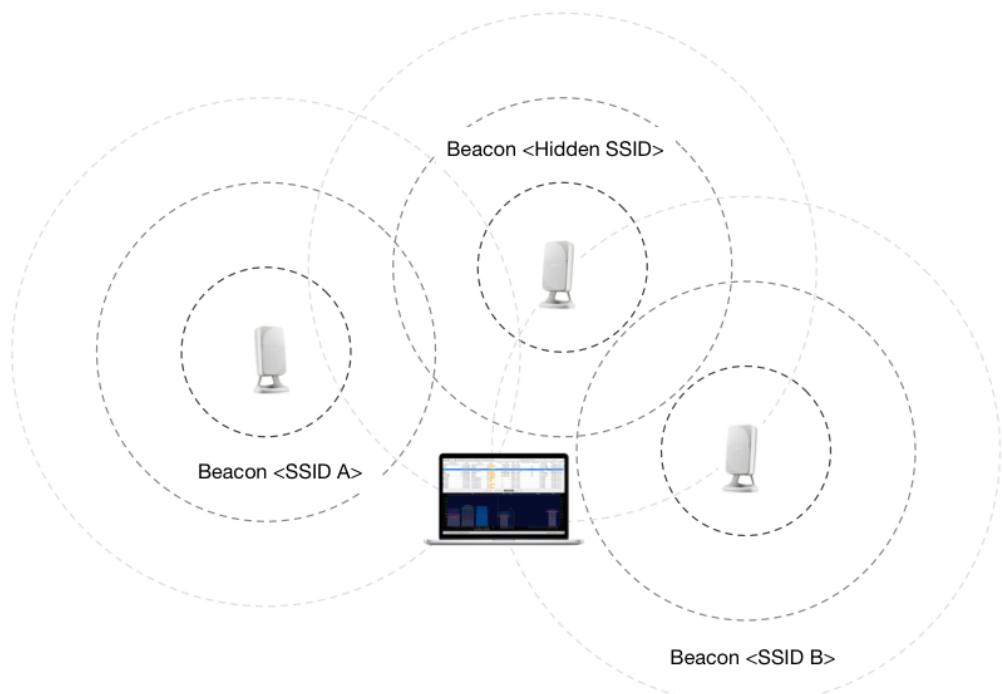


Figure 2 – Scan passif⁴

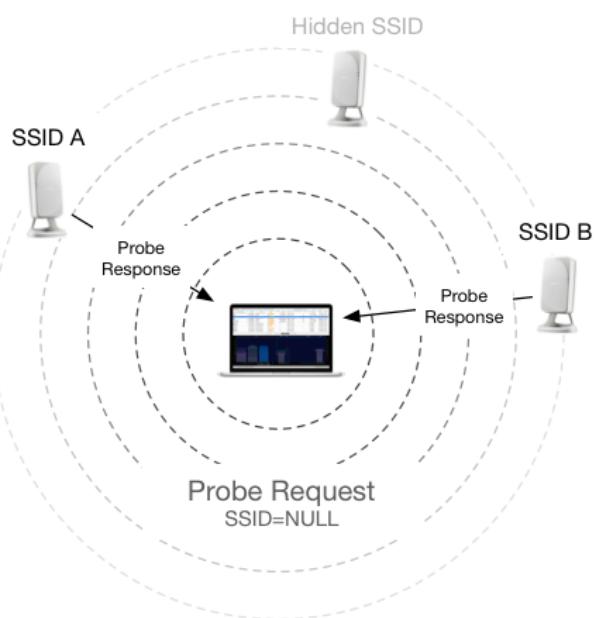


Figure 3 – Scan actif⁵

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 2018⁶. 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 suffi 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 comportent 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.

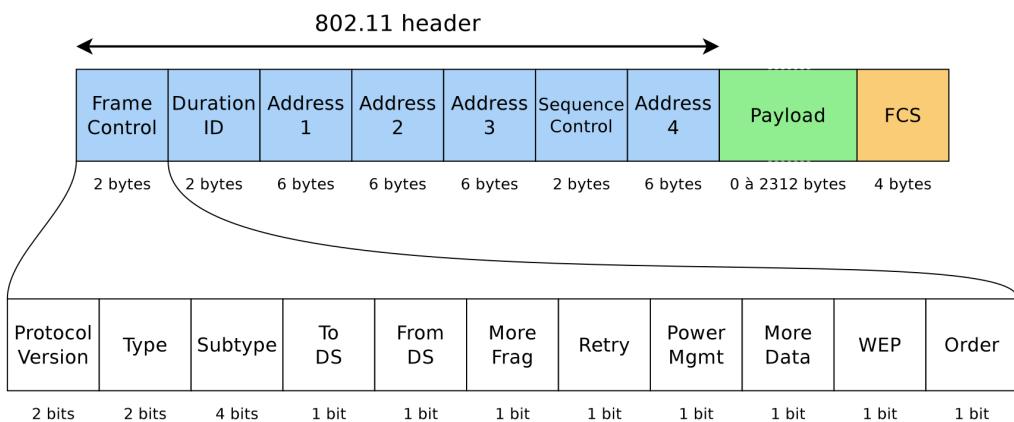


Figure 4 – Format d'un paquet WiFi 802.11

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 AutoFlight⁷, 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-Drone⁸ et python-ardrone⁹
- L'interface ardrone-webflight¹⁰, 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.

Il faut cependant noter que ce programme suppose, à son démarrage, que l'AR.Drone soit posé. Ainsi, si celui-ci est en vol lors de la prise de contrôle, `ardrone-webflight` lui envoie l'ordre de se poser. Ce comportement pourrait être corrigé en modifiant le programme, mais nous n'avons pas eu le temps de le faire.

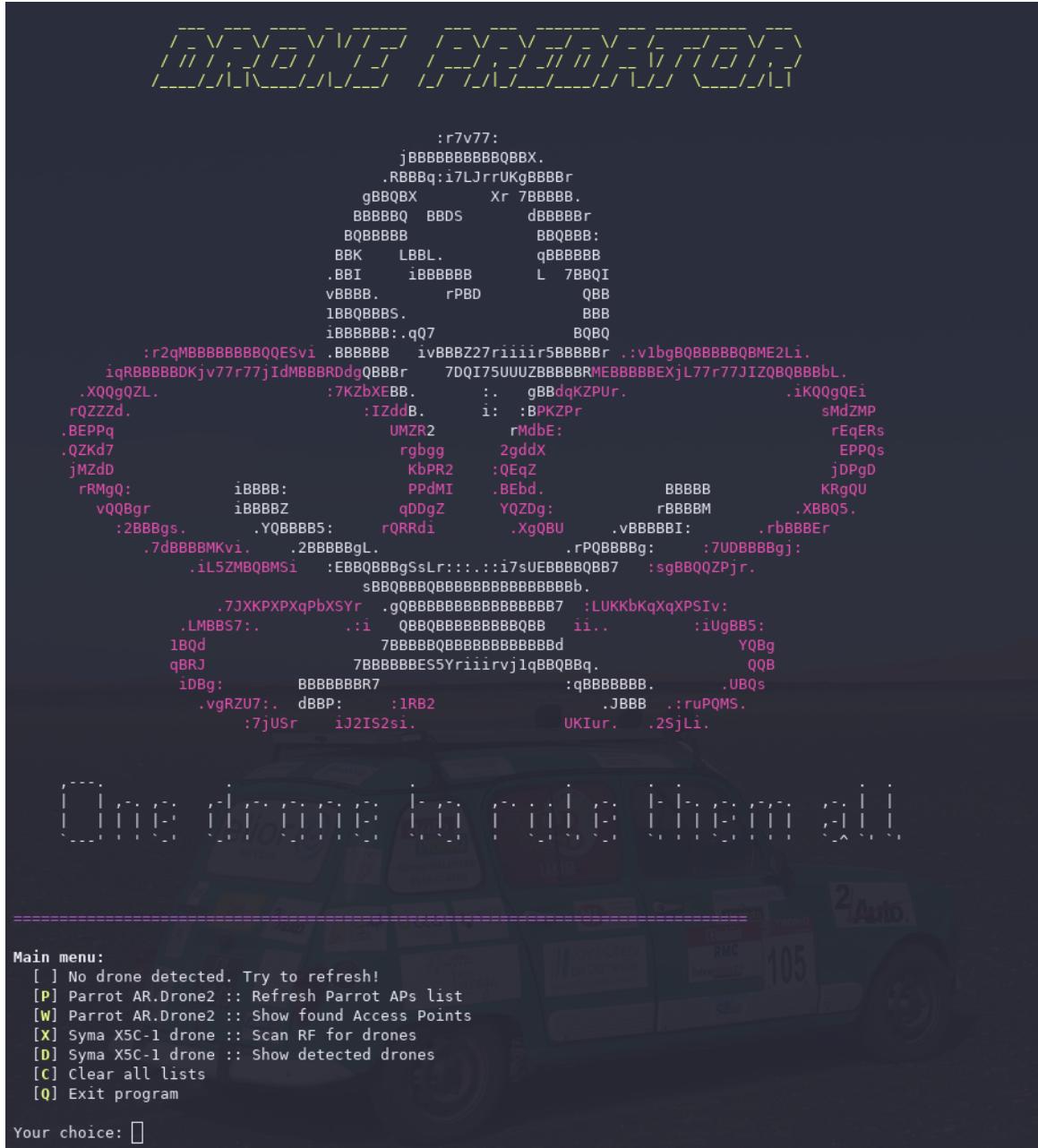


Figure 5 – Menu de l'outil predator-drone

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 étudié 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 rejeté, comme le montre le schéma suivant :

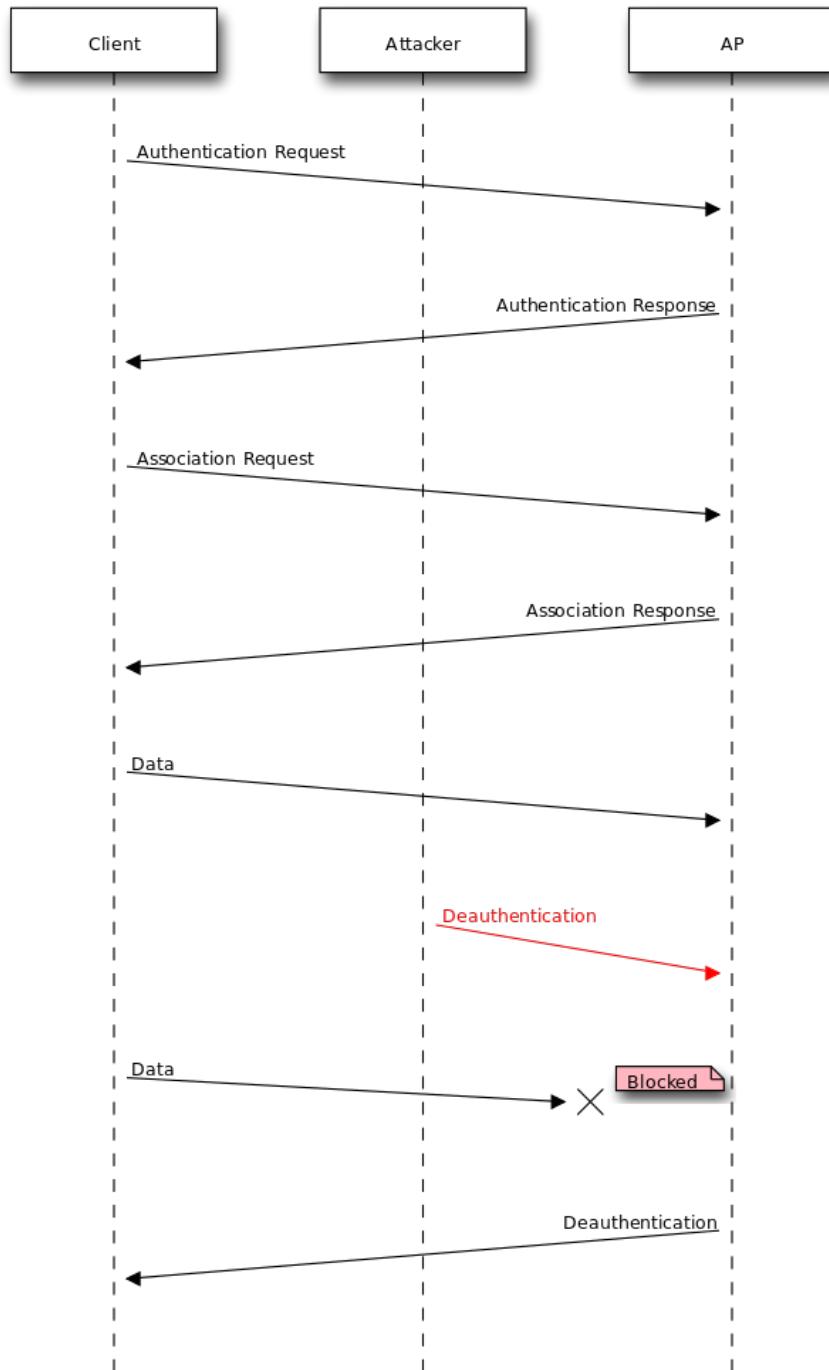


Figure 6 – Attaque par désauthentification WiFi¹¹

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.

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

```

1 # Build deauth packets: addr1=dst, addr2=src, addr3=bssid
2 deauth_client = RadioTap() \
3     / Dot11(addr1=ap.bssid, addr2=client.mac, addr3=ap.bssid) \
4     / Dot11Deauth(reason=7)
5
6 deauth_ap = RadioTap() \
7     / Dot11(addr2=ap.bssid, addr1=client.mac, addr3=ap.bssid) \
8     / Dot11Deauth(reason=7)
9
10 # Send packets, updating sequence number
11 pkt_count = 3*64*2
12 for i in range(0, pkt_count, 2):
13     deauth_client.SC = i << 4
14     deauth_ap.SC      = (i + 1) << 4
15
16     sendp(deauth_client, iface=self.card.dev, verbose=False)
17     sendp(deauth_ap,    iface=self.card.dev, verbose=False)
18
19 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 envoyées 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 un 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 ayant eu lieu peu avant la fin du projet, nous n’avons pas eu le temps de confirmer/infirmier cette hypothèse.

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 dans 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 PyRIC¹². 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.

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

12. <https://github.com/wraith-wireless/PyRIC>

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

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

3.1 Présentation de l'attaque

Le Syma X5C-1 est un petit drone RF opérant sur 3 canaux dans la bande de fréquence 2.4 GHz. Il se pilote via une télécommande.



Figure 7 – Syma X5C-1

Une attaque, dont les détails ont été publiés sur le blog *ptsecurity*¹³, a déjà été menée sur ce drone. Nous avons cherché à rejouer la même attaque pour la comprendre et l'embarquer sur le drone prédateur.

Afin de pouvoir détourner le drone en vol, il est nécessaire de :

- Découvrir les canaux utilisés
- Découvrir la modulation utilisée et décoder les trames du protocole
- Interpréter la trame du protocole afin de faire le lien avec les positions des joystick de la télécommande
- Implémenter le protocole avec une puce radio émettrice

Pour simplifier le contrôle du drone après l'attaque, nous avons choisi d'utiliser une manette de jeu USB comme contrôleur.

Une fois les étapes ci-dessus réalisées, nous avons pu constater que la télécommande légitime n'a quasiment plus d'effet sur le drone. Cela s'explique par la fréquence d'émission des trames. En effet, notre implémentation du protocole émet beaucoup plus de trames que la télécommande légitime.

Le drone reçoit donc à la fois les ordres légitimes et les ordres de l'attaquant. Il essaye de tous les exécuter. Comme l'attaquant envoie beaucoup plus de trames, c'est lui qui possède le contrôle dans les faits.

3.2 Compréhension de la couche physique

Ce chapitre détaille les étapes qui ont été nécessaires afin de comprendre la couche physique utilisée par le drone.

3.2.1 Découverte des canaux utilisés

Nous avons utilisé une radio logicielle (SDR USRP B200) avec le logiciel GnuRadio.

Un SDR (*Software-Defined Radio*) est un équipement qui permet de recevoir et émettre des signaux radios. Contrairement à une puce radio classique, le SDR ne fait aucune modulation/démodulation ni aucun traitement. Ceux-ci sont réalisés par un ordinateur relié au SDR.

13. <https://blog.ptsecurity.com/2016/06/phd-vi-how-they-stole-our-drone.html>

GnuRadio est un logiciel qui permet de construire des chaînes de traitement radio. Il permet des traitements très complexes. La chaîne peut être construite graphiquement avec l'application GnuRadio Companion.

Nous avons donc parcouru la bande de fréquence entre 2.400 GHz et 2.525 Ghz avec Gnuradio. Cela correspond à la plage de fréquences gérables par un nRF24L01+ (la puce que l'on souhaitait utiliser pour prendre le contrôle du drone).

Pour cela nous avons créé une *waterfall* et avons rapidement constaté de l'activité autour de **2.410 GHz**. Pour préciser notre observation, nous avons rajouté une FFT¹⁴ avec Gnuradio autour de cette fréquence. Après avoir bien centré notre FFT, nous avons pu relever une **bande passante autour de 800 kHz**.

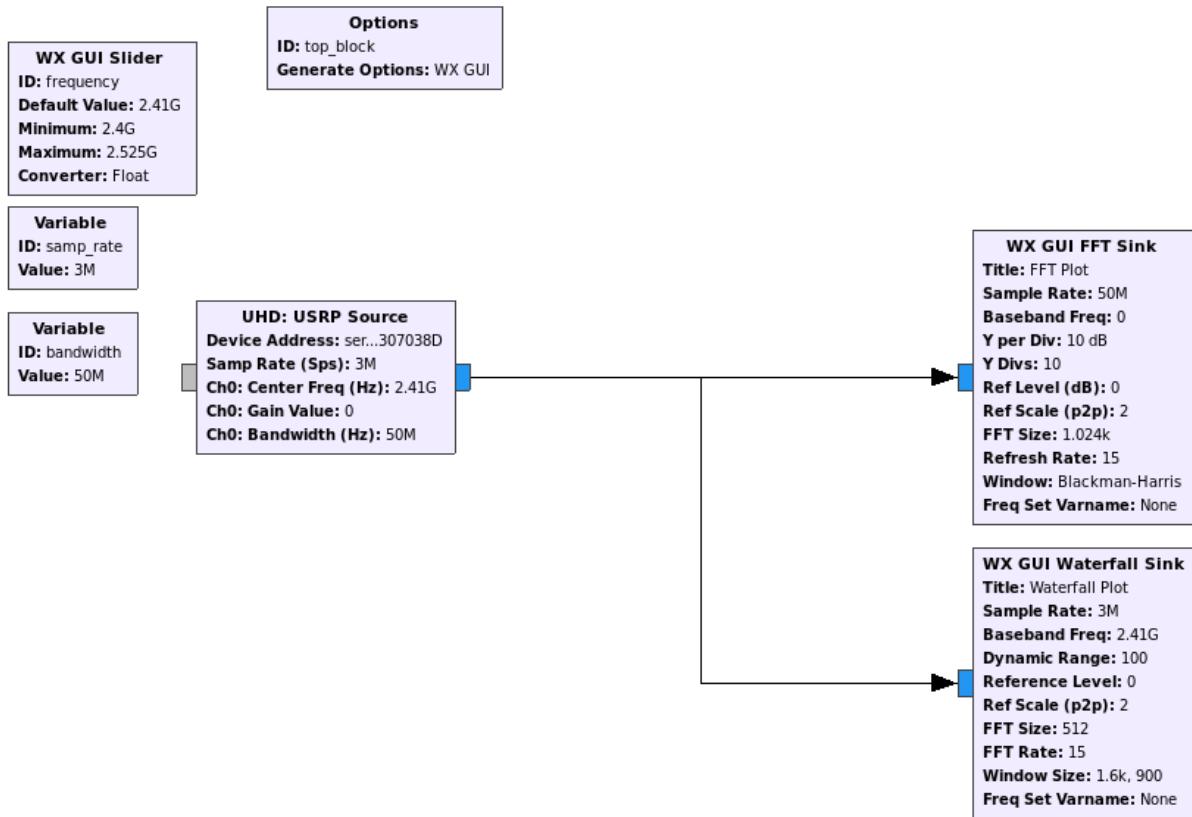


Figure 8 – Chaîne de traitement GnuRadio pour le scan des fréquences

14. *Fast Fourier Transform*, ou transformation de Fourier Rapide

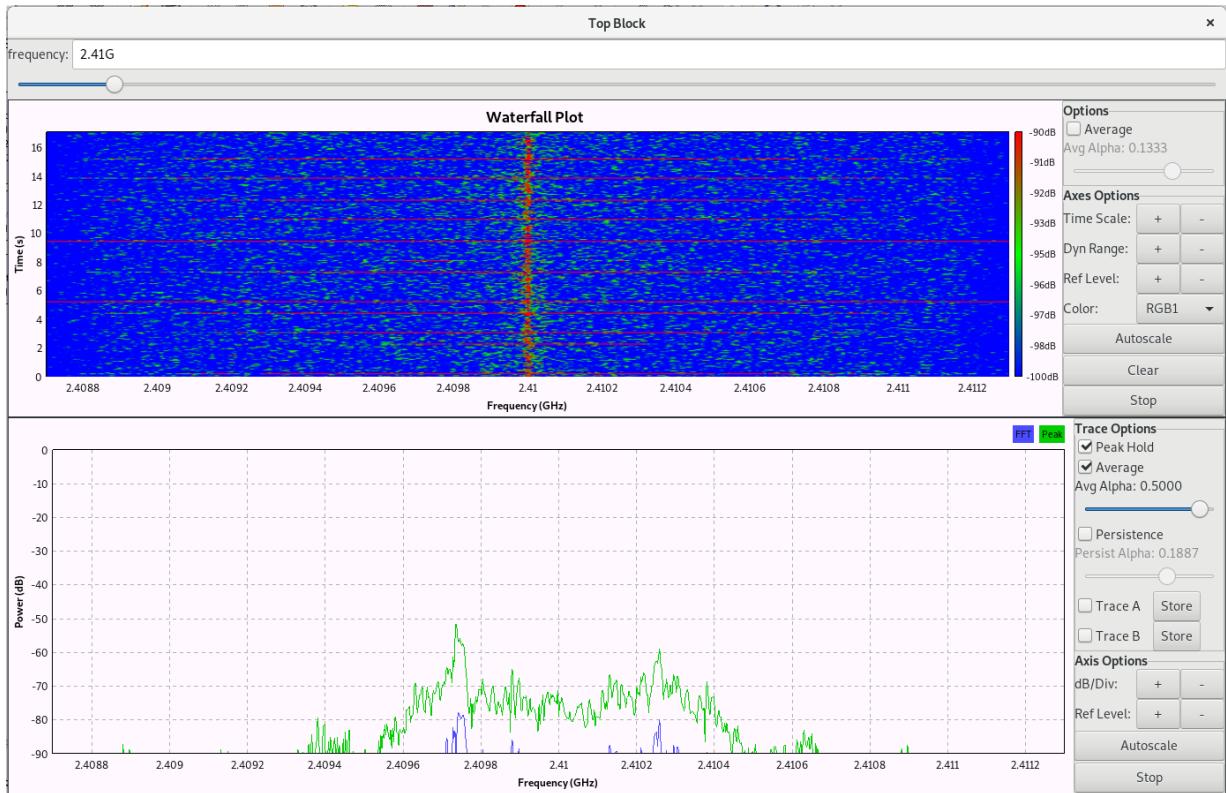


Figure 9 – Waterfall et FFT obtenues

En regardant un extrait de la *datasheet* d'un module radio nRF24L01+¹⁵, on peut lire :

```

1 6.3 RF channel frequency
2 -----
3
4 The RF channel frequency determines the center of the channel used by the nRF24L01+. The
5 channel occupies a bandwidth of less than 1MHz at 250kbps and 1Mbps and a bandwidth of
6 less than 2MHz at 2Mbps. nRF24L01+ can operate on frequencies from 2.400GHz to 2.525GHz.
7 The programming resolution of the RF channel frequency setting is 1MHz.
8
9 At 2Mbps the channel occupies a bandwidth wider than the resolution of the RF channel
10 frequency setting. To ensure non-overlapping channels in 2Mbps mode, the channel spacing
11 must be 2MHz or more. At 1Mbps and 250kbps the channel bandwidth is the same or lower than
12 the resolution of the RF frequency.
13
14 The RF channel frequency is set by the RF_CH register according to the following formula:
15
16 F0 = 2400 + RF_CH [MHz]
17
18 You must program a transmitter and a receiver with the same RF channel frequency to
19 communicate with each other.

```

En supposant que le chipset du drone est similaire à un nRF24L01, on voit qu'avec 800 kHz de bande passante, nous avons un **débit de 250 Kbps ou 1 Mbps** mais certainement pas de 2 Mbps.

Nous avons ensuite utilisé la formule suivante pour trouver le canal $F_0 = 2400 + RF_CH$. Le résultat est $2410 - 2400 = 10$ donc le canal utilisé est le canal 10.

En appliquant cette méthode sur les autres fréquences, nous avons constaté de l'activité sur les canaux : **10, 31, 42 et 66**.

15. https://framagit.org/tigre-bleu/predator-drone/blob/master/doc/nRF24L01+/nRF24L01Plus_Preliminary_Product_Specification_v1_0.pdf

3.2.2 Découverte de la modulation

Nous avons pris l'hypothèse que les quatre canaux sont utilisés de manière identique et nous nous sommes alors concentrés sur le canal 10. Nous avons donc créé un autre traitement GnuRadio pour enregistrer le signal en sortie de l'USRP.

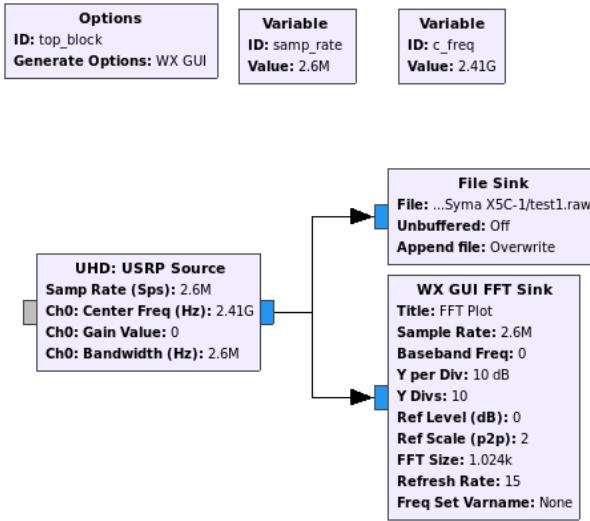


Figure 10 – Chaîne d'enregistrement GnuRadio

Puis, nous avons ouvert cet enregistrement avec le logiciel baudline, afin de le visualiser.

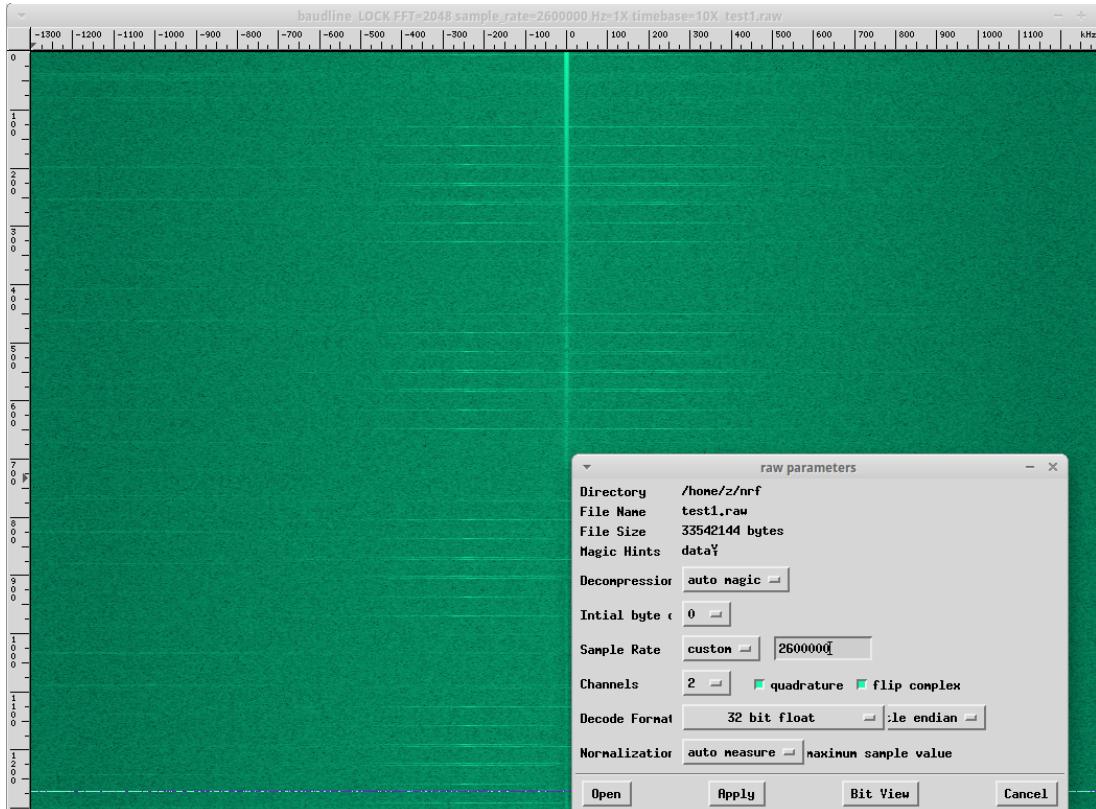


Figure 11 – Waterfall Baudline

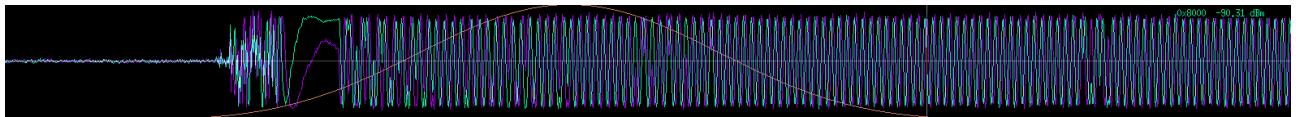


Figure 12 – Signal brut

Nous avons constaté que l'amplitude du signal est constante, ce n'était donc pas de l'ASK. Il était légitime de penser que la modulation était du FSK/GFSK mais cela aurait aussi pu être du PSK. La modulation GFSK étant souvent utilisée dans des équipements électroniques de loisirs, nous sommes partis dans cette direction.

Nous avons rajouté un filtre passe-bas afin d'avoir un signal plus propre, puis un bloc Quadrature Demod permettant de séparer les fréquences.

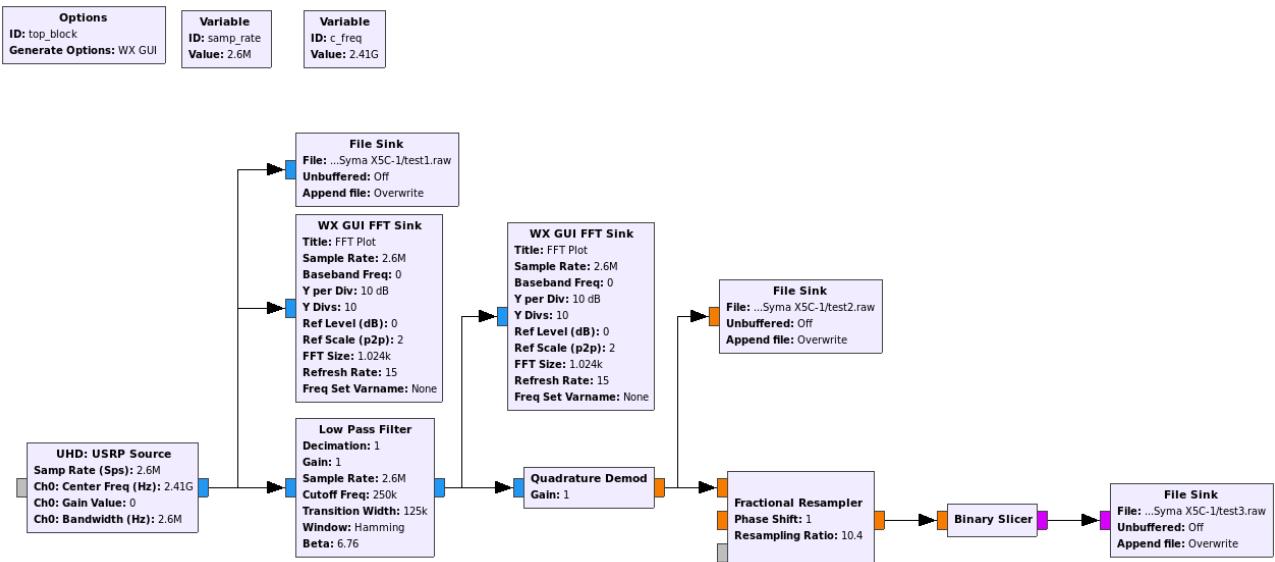


Figure 13 – Chaîne de démodulation GnuRadio

Nous avons alors pu voir les bits de notre signal :

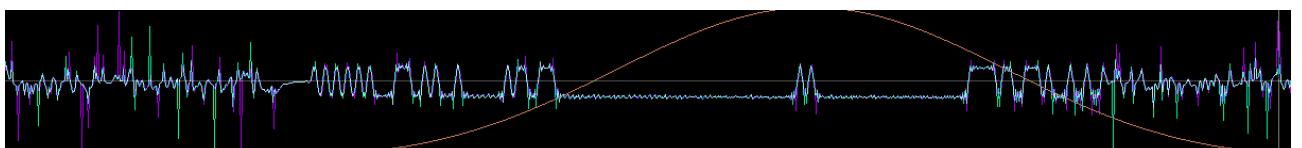


Figure 14 – Signal démodulé

La fin du traitement GnuRadio enregistre dans un fichier des bits pris à une période constante correspondante à 250 kbits/s. Nous y avons donc retrouvé nos trames mais aussi de “faux bits” issus de la mauvaise interprétation du bruit.

0000007f0	00 00 00 01 01 01 00 00	01 00 01 00 01 00 01 00
000000800	01 01 01 01 00 00 00 00	01 01 00 00 00 00 00 01 00
000000810	00 00 00 01 00 00 01 01	01 00 00 01 01 00 00 01
000000820	01 00 00 01 00 01 00 00	00 00 00 01 00 01 01 00
000000830	01 01 00 00 01 01 01 01	00 00 00 00 01 00 01 00
000000840	00 00 00 00 01 01 00 00	00 01 01 01 00 01 01 01
000000850	01 01 00 00 00 01 00 01	01 00 01 00 00 01 01 00
000000860	00 00 01 01 01 01 00 00	00 01 00 00 00 00 01 00
000000870	01 01 01 01 01 01 00 00	01 00 01 01 00 01 01 01
000000880	01 01 01 00 01 00 00 01	00 00 01 01 01 01 01 00
000000890	01 01 01 01 01 01 00 01	00 01 00 00 01 00 00 00

Figure 15 – Dump hexadécimal du bit stream

Remarque : pour être plus propre, nous aurions pu développer une chaîne de démodulation complète permettant de synchroniser l'horloge et de filtrer le bruit entre plusieurs messages. Cependant ce n'était pas strictement nécessaire pour réaliser notre attaque.

3.3 Compréhension de la couche liaison de données

Une fois la modulation déterminée, il nous a fallu comprendre le fonctionnement de la couche liaison de données.

3.3.1 Première impression

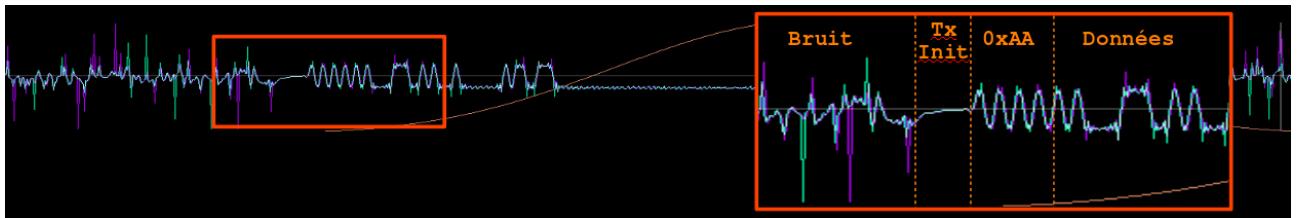


Figure 16 – Début de trame

En regardant de plus près notre signal démodulé, nous avons observé les éléments suivants :

- Du bruit
- Une phase où le signal se stabilise, probablement l'initialisation de l'émetteur
- Une suite de 1 et de 0 alternés, probablement 1 octet de préambule 0xAA
- Une série de bits, probablement les bit de données
- Du bruit

Le nombre d'octets total (incluant le préambule) semblait donc être de 18.

Il est bon de noter que la présence d'un octet de préambule n'est pas surprenante. Il est courant pour un émetteur de transmettre un préambule de 1 et de 0 alternés afin que le récepteur puisse synchroniser son horloge interne sur celle de l'émetteur.

On peut voir dans la spécification du module radio nRF24L01+ que celui-ci émet un préambule de 0xAA ou 0x55. Il est donc légitime de penser que le drone utilise un module de ce type. Il s'agit en effet d'un module bon marché très utilisé pour des petits équipements électroniques.

3.3.2 Module nRF24L01+

À ce stade, nous nous sommes intéressés plus en détail au fonctionnement du module nRF24L01+. C'est en effet un module radio très populaire faisant du GFSK sur la bande 2.4 GHz. Il est utilisé dans beaucoup de projets électroniques : drones mais aussi souris sans-fil, projets DIY, etc. En voici les caractéristiques les plus importantes pour notre projet :

- Il utilise un préambule en émission de type 0x55 ou 0xAA
- Une adresse Tx ou Rx est configurable entre 3 et 5 octets
- Deux couches de liaison de données en mode paquet sont possibles :
 - Le mode basique ;
 - Le mode *Enhanced Shockburst*, qui propose des fonctionnalités supplémentaires tels que des acquitements et de la reprise sur erreur.
- La taille de la *payload* est configurable entre 0 et 32 octets
- Un CRC est optionnel (sur 1 ou 2 octets)

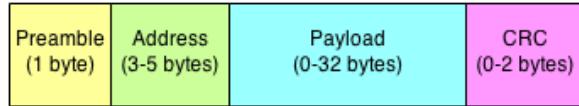


Figure 17 – Protocole nRF24L01 basique

Ainsi, nous avons supposé que le drone Syma utilisait un module de type **nRF24L01+** ou compatible, en mode basique. Il nous fallait alors trouver la taille des champs configurables. Pour cela, il nous fallait donc écouter sur un des canaux utilisés par la télécommande.

Nous savions déjà qu'il y avait 18 octets de bits utiles dont 1 octet de préambule. Nous en concluons donc que :

- L'adresse ferait entre 3 et 5 octets
- Le CRC ferait entre 0 et 3 octets
- La *payload* ferait entre 10 et 14 octets

3.3.3 Détermination des paramètres du protocole nRF24L01+

Méthode 1 : analyse en temps différé avec un SDR

Nous sommes partis du fichier de bits enregistré par GnuRadio en utilisant un outil déjà disponible sur GitHub¹⁶ qui assiste dans cette tâche.

Ce script recherche le préambule 0xAA puis affiche les octets qui suivent en fonction de taille de champs configurable. Il est ainsi possible de faire des hypothèses sur la taille des champs, par exemple : l'adresse est sur 3 octets, la *payload* sur 13 et le CRC sur 1. L'outil recalcule le CRC ce qui permet de vérifier en partie si nos hypothèses sont correctes ou non.

L'outil permet également de trouver de façon partiellement automatique la taille de la *payload* et la taille du CRC.

L'image ci-dessous donne la sortie du script avec les options suivantes :

```

1 addr_len = 3
2
3 find_addr = True
4 show_badCRC = False
5 show_crc16 = True
6 show_crc8 = True
7 show_shockburst = False
8 show_simple = True
9 brute_len = True

```

Avec ces options, on fait l'hypothèse que l'adresse est sur 3 octets puis le script essaye toutes les combinaisons de taille de *payload* et de CRC possibles. Si un CRC calculé correspond à un CRC reçu alors il affiche la trame.

16. https://github.com/chopengauer/nrf_analyze/blob/master/nrf24_analyzer.py

0Position 663 offset 663	Address alca20 Data 16700000000000007c000f01c7 CRC c2c3 Len 12 preamb = aa
2Position 2652 offset 1989	Address 04f2cc Data 6cb15cb5b08c042a1608a82e81022cb3ec60e8d35e4e6b1cdab58159ddbc93 CRC 91 Len 31
2Position 3753 offset 1101	Address 3e6b2e Data d5 CRC c8 Len 1
2Position 5432 offset 1679	Address 3ac82f Data ddddbbe4d7cd338bf4 CRC 09 Len 9
2Position 7123 offset 1691	Address 63d05e Data 5944b079467cbfbdaeae60731cd4d5a7cae37 CRC 57 Len 18
2Position 7832 offset 709	Address 8e3ef1 Data 8b46369f49f732e830e8f1be44be7fa499c0418062658e19e4 CRC ac Len 25
2Position 7834 offset 2	Address 38fb6 Data 2d CRC 18 Len 1
2Position 8098 offset 264	Address 6ed93a Data f87698429a6ea830e0f2d795a932 CRC db Len 14
0Position 8320 offset 222	Address alca20 Data 16700000000000007c000f01c7 CRC c2c3 Len 12 preamb = aa
2Position 8741 offset 421	Address 22e7df Data a9d3da8918cb75b4122e98b980dedc CRC 59 Len 15
2Position 9180 offset 439	Address 961058 Data 1df6489e47c48f6407d19759fc CRC 22 Len 13
0Position 9413 offset 233	Address alca20 Data 16700000000000007c000f0003 CRC 683a Len 12 preamb = aa
2Position 11269 offset 1856	Address d95eaf Data 05dd4d0f0f2de749df17fa393295ee62dccc CRC fb Len 18
2Position 15781 offset 4512	Address 06d472 Data a7f770e60137921d8bf0c3ee6e75c5c554577b75e5c5e3 CRC e7 Len 23
0Position 17070 offset 1289	Address alca20 Data 16700000000000007c000f01c7 CRC c2c3 Len 12 preamb = aa
0Position 18164 offset 1094	Address alca20 Data 16700000000000007c000f01c7 CRC c2c3 Len 12 preamb = aa
2Position 18509 offset 345	Address 34f629 Data 7a8c39f344b4ad42bac5c9839t CRC 29 Len 13
2Position 18897 offset 388	Address 25c5db Data 54bfc821 CRC 21 Len 4
2Position 19282 offset 385	Address 639c8b Data b8f11c3dffdd7a CRC 9f Len 7
2Position 20870 offset 1588	Address 930367 Data 44fe985069a98b0a CRC 25 Len 8
2Position 21577 offset 707	Address 916051 Data 3b CRC ec Len 1
2Position 22473 offset 896	Address 7c39f6 Data ee1247f0d26242ce0f47cc93c2cdcf1b7dc068f27e9a2d403276097cd28 CRC 62 Len 30
2Position 22775 offset 302	Address b8f71c Data 68 CRC 37 Len 1
2Position 23789 offset 1014	Address 71b899 Data f33874c146685d1fa5757a6dd5e36fb07df5c CRC 62 Len 19
2Position 23790 offset 1	Address e37133 Data e670e9828cd0a3f4aaef4 CRC db Len 11
2Position 24635 offset 845	Address 24f27e Data feff1163e6bb0bd3d2465f CRC 94 Len 10
2Position 24978 offset 343	Address 8fbeae Data eccded6f139caf6315dfeffd255e37 CRC 96 Len 15
2Position 25110 offset 132	Address e3796f Data 8f3e23ff19724a93f8ff5847 CRC 51 Len 12
0Position 25820 offset 710	Address alca20 Data 16700000000000007c000f01c7 CRC c2c3 Len 12 preamb = aa

Figure 18 – Exécution de l’outil `nrf24_analyser.py` avec une hypothèse d’une adresse sur 3 octets

On constate qu’il y a beaucoup de “fausses trames” provenant du bruit et pour lesquelles le CRC est valide, par hasard. Hors, nous savons que la *payload* fait entre 10 et 14 octets ce qui nous permet de filtrer une grande partie des fausses trames. Parmis celles qui restent, on remarque rapidement que la même trame se répète. Nous avons donc identifié la vraie trame parmi le bruit.

La combinaison adresse de 3 octets (0xa1ca20) et *payload* de 12 octets est donc une combinaison possible, mais elle n’est pas la seule. En effet, l’adresse pourrait être sur 4 octets (0xa1ca2016) avec une *payload* sur 11 octets ou encore être sur 5 octets (0xa1ca201770) avec une *payload* sur 10 octets.

On observe rapidement, en utilisant la télécommande, que les 5 premiers octets restent fixes quelle que soit la position des commandes. Nous pouvons donc en déduire les caractéristiques suivantes : **adresse de 5 octets (0xa1ca201670), payload de 10 octets et CRC de 2 octets**.

[root@tigrou Syma X5C-1]# ../../nrf_analyze/nrf24_analyzer.py	Start
0Position 483 offset 483	Address alca201670 Data 00000000002000000176 CRC c711 Len 10 preamb = aa
0Position 9226 offset 8743	Address alca201670 Data 00000000002000000003 CRC da12 Len 10 preamb = aa
0Position 16875 offset 7649	Address alca201670 Data 00000000002000000176 CRC c711 Len 10 preamb = aa
0Position 17968 offset 1093	Address alca201670 Data 00000000002000000176 CRC c711 Len 10 preamb = aa
0Position 25617 offset 7649	Address alca201670 Data 00000000002000000176 CRC c711 Len 10 preamb = aa
0Position 26710 offset 1093	Address alca201670 Data 00000000002000000003 CRC da12 Len 10 preamb = aa
0Position 34359 offset 7649	Address alca201670 Data 00000000002000000176 CRC c711 Len 10 preamb = aa
0Position 35452 offset 1093	Address alca201670 Data 00000000002000000176 CRC c711 Len 10 preamb = aa
0Position 44195 offset 8743	Address alca201670 Data 00000000002000000003 CRC da12 Len 10 preamb = aa
0Position 51845 offset 7650	Address alca201670 Data 00000000002000000176 CRC c711 Len 10 preamb = aa
0Position 52937 offset 1092	Address alca201670 Data 00000000002000000176 CRC c711 Len 10 preamb = aa
0Position 60588 offset 7651	Address alca201670 Data 00000000002000000176 CRC c711 Len 10 preamb = aa
0Position 61681 offset 1093	Address alca201670 Data 00000000002000000003 CRC da12 Len 10 preamb = aa
0Position 69331 offset 7650	Address alca201670 Data 00000000002000000176 CRC c711 Len 10 preamb = aa
0Position 79166 offset 9835	Address alca201670 Data 00000000002000000003 CRC da12 Len 10 preamb = aa
0Position 87909 offset 8743	Address alca201670 Data 00000000002000000176 CRC c711 Len 10 preamb = aa

Figure 19 – Exécution de l’outil `nrf24_analyser.py` avec des tailles de champs correctes

Méthode 2 : analyse en temps réel avec un nRF24L01+

Un attaquant n’ayant pas de SDR et supposant, à raison, que le drone utilise un protocole de type nRF24L01+ pourrait tout de même retrouver les tailles des champs et la valeur de l’adresse en utilisant un simple module nRF24L01+. Il faudrait pour cela qu’il écoute tout le traffic radio sur chaque canal et qu’il essaye d’y reconnaître des trames valides.

Il serait alors nécessaire que le module soit capable de passer en mode *promiscuous*, ce qui n’est pas officiellement supporté. Cependant, il est possible d’arriver à ce résultat en combinant certaines propriétés du module.

Pseudo-mode *promiscuous* avec un nRF24L01+

Le module possède les caractéristiques suivantes :

- Le préambule n'est pas utilisé à la réception d'un message. Le module se contente de rechercher son adresse sur le canal pour trouver le début d'une trame qui lui est destinée.
- La partie juste avant le préambule (Tx init) est, la plupart du temps, interprétée comme 0x00.
- Le registre de configuration de la taille de l'adresse autorise les valeurs 0b01, 0b10 et 0b11 pour des tailles de 3, 4 et 5 octets. La valeur 0b00 est interdite dans la *datasheet* du module. Pourtant, si on met 0b00 dans ce registre, on constate que le module considère une adresse de 2 octets.

03	SETUP_AW				Setup of Address Widths (common for all data pipes)
	Reserved	7:2	000000	R/W	Only '000000' allowed
	AW	1:0	11	R/W	RX/TX Address field width '00' - Illegal '01' - 3 bytes '10' - 4 bytes '11' - 5 bytes <small>LSByte is used if address width is below 5 bytes</small>

Figure 20 – Extrait de la spécification du nRF24L01+

En combinant ces trois propriétés, on peut configurer une adresse de 0x00AA (2 octets) qui va correspondre à tous les débuts de trames possibles.

Il faut tout de même noter une limitation de cette méthode : si la trame fait la taille maximale supportée^a, il sera impossible de récupérer les 2 derniers octets de CRC car le module retournera 39 octets en partant de 0x00AA.

- a. 39 octets : adresse de 5 octets, *payload* de 32 octets et CRC de 2 octets

Nous n'avons pas mis en pratique cette méthode pour la rétro-ingénierie du protocole vu que nous connaissons déjà toutes les informations nécessaires via la méthode 1, qui est plus simple. Nous avons néanmoins implémenté ce pseudo-mode promiscuous dans *predator-drone* lors de la phase de scan afin de détecter les adresses des drones inconnus environnants.

3.4 Compréhension du protocole Syma

Nous avons ainsi déterminé que l'adresse fait 5 octets et que la *payload* fait 10 octets. Il nous faut donc à présent comprendre le contenu de ces 10 octets de *payload*.

Pour cela, nous avons monté un nRF24L01+ sur un Raspberry Pi 3 et adapté un script existant permettant d'afficher dans une console en temps réel le contenu des *payload*¹⁷.

Pour faciliter l'interaction avec le public lors du tutoriel de la THCon, nous avons rajouté un petit écran OLED sur bus I2C afin d'y afficher le contenu des trames.

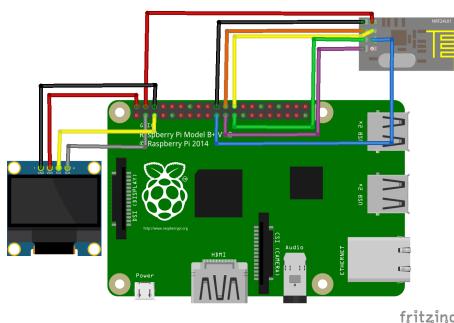


Figure 21 – Montage de l'analyseur

17. https://framagit.org/tigre-bleu/predator-drone/blob/master/tools/syma_protocol_analyser/syma_protocol_analyser.py

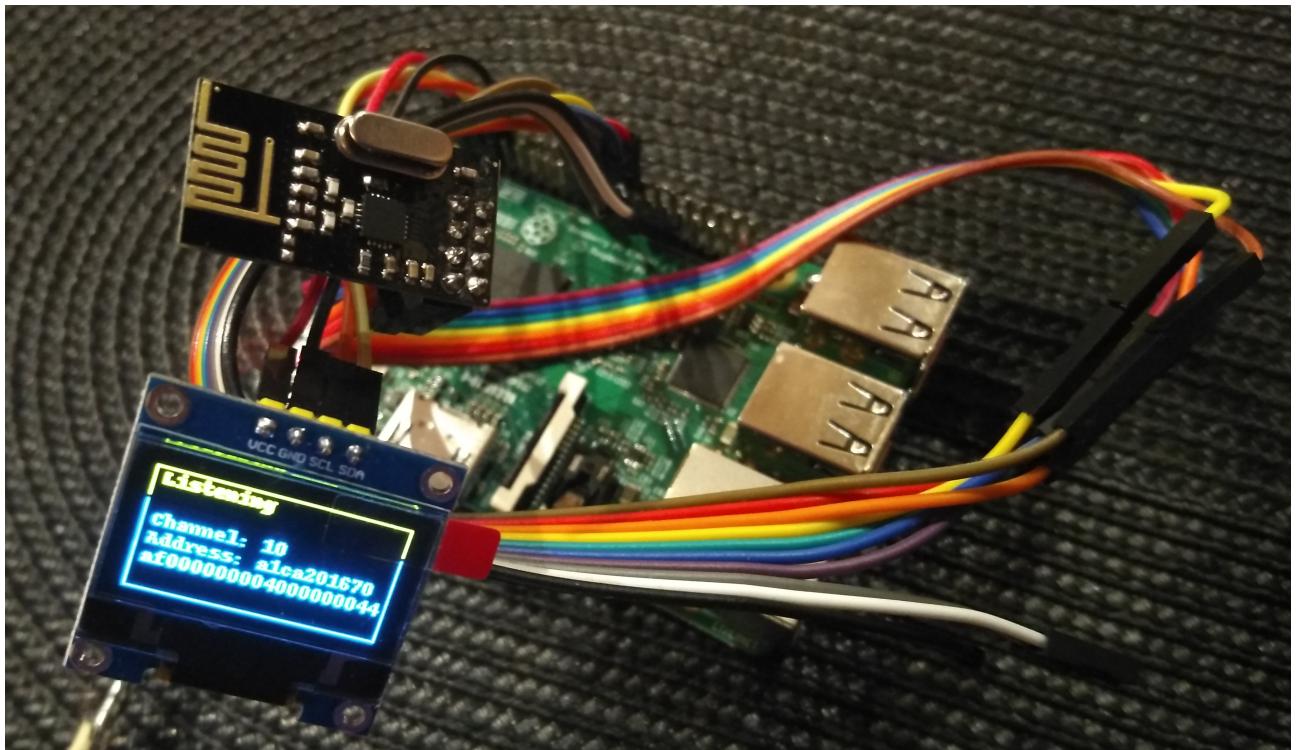


Figure 22 – Capture et affichage des paquets

Remarque : la performance de l'ensemble avec l'écran n'est pas suffisante pour faire les observations confortablement donc celui-ci n'a été utilisé que pour la THCon.

En observant les paquets émis par la télécommande alors qu'on bouge les joysticks ou actionne les boutons, nous avons vite observé que :

- Les octets 1 à 4 bougent suivant les 2 axes de chaque joystick ;
- Les octets 5 à 9 bougent en fonctions des appuis sur les boutons (trims, modes de la télécommande et actions sur la caméra) ;
- L'octet 10 bougent dès que l'un des bits des 9 premiers octets changent. Ce comportement nous a fait penser à celui d'une somme de contrôle.

En affinant un peu plus notre compréhension, nous avons conclu que :

- Octet 1 = altitude (0x00 – 0xFF)
- Octet 2 = tangage (avant : 0x00 – 0x7F / arrière : 0x80 – 0xFF)
- Octet 3 = roulis (gauche : 0x00 – 0x7F / droite : 0x80 – 0xFF)
- Octet 4 = lacet (gauche : 0x00 – 0x7F / droite : 0x80 – 0xFF)
- Octet 5 à 9 = trims, mode, etc. donc peu utiles pour notre attaque
- Octet 10 = pseudo CRC construit comme le XOR des 9 premiers octets auquel on ajoute 0x55

Preamble (1 byte) 0xaa	Address (5 bytes) 0xa1ca201670	Altitude (1 byte)	Pitch (1 byte)	Roll (1 byte)	Yaw (1 byte)	Other Stuff (5 bytes)	Syma « CRC » (1 byte)	DataLink CRC (2 bytes)
------------------------------	--------------------------------------	----------------------	-------------------	------------------	-----------------	--------------------------	--------------------------	---------------------------

Figure 23 – Trame complète

Nous avons ainsi compris entièrement le protocole et sommes donc en mesure de le réimplémenter en émission pour usurper une télécommande.

3.5 Implémentation du protocole

Il est relativement simple d'implémenter le protocole en Python sur un RPi avec un module nRF24L01+. Afin de simplifier le contrôle du drone, nous choisissons d'utiliser une manette USB. Nous sommes parti d'un code existant¹⁸ que nous avons simplifié et adapté pour fonctionner avec notre manette.

Après implémentation de notre script d'attaque, nous avons observé que si le drone était allumé mais que la télécommande ne l'était pas, le drone clignotait et ne prenait pas en compte nos commandes. Nous en avons conclu qu'il y avait un appairage du drone et de la télécommande. Il faudrait creuser cet aspect mais pour réaliser l'attaque souhaitée, nous n'en avions pas besoin. En effet nous souhaitons intercepter le drone en vol, donc déjà appairé à une télécommande.

Lorsque le drone était appairé avec sa télécommande originale, nous avons observé que les ordres envoyés par le RPi l'emportent sur ceux de la télécommande : **l'attaque fonctionne**.

3.6 Caractérisation de l'attaque

L'attaque fonctionne car le Raspberry émet plus de trames que la télécommande légitime. Le drone reçoit des trames contradictoires mais valides des deux émetteurs et cherche à obéir aux deux ordres. L'attaquant envoyant plus d'ordres que le pilote légitime, c'est lui qui l'emporte.

Pour visualiser ce phénomène, nous avons développé un simple outil de log¹⁹ pour un Raspberry Pi avec un nRF24L01+ qui enregistre les trames reçues sur un canal. On peut ensuite tracer des courbes avec Gnuplot ou LibreOffice.

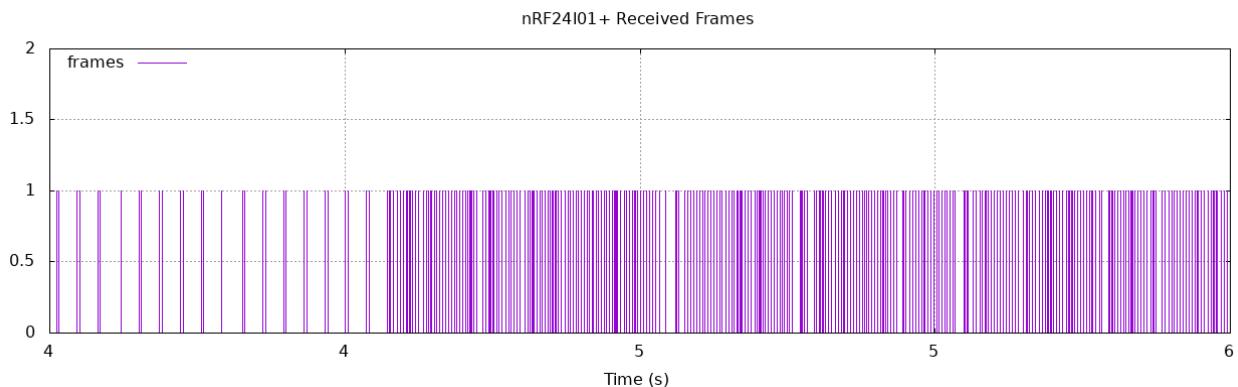


Figure 24 – Répartition des trames sur un canal au début d'une attaque

On voit très nettement l'augmentation du nombre de trames pendant l'attaque.

Nous avons également remarqué un phénomène qui nous avait échappé jusqu'alors : la télécommande émet 2 trames sur un canal avant de passer au canal suivant. Notre script d'attaque n'utilise pas le même algorithme et émet une seule fois sur chaque canal. En pratique, cette différence n'a pas d'effet. Cela est probablement dû au fort taux d'émission de trame par notre script attaquant.

A partir du log enregistré, nous pouvons aussi recalculer des taux de trames par seconde reçus par le drone avant et pendant l'attaque :

- Environ 50 trames/s par canal (200 trames/s au total) envoyées hors attaque
- Environ 250 trames/s par canal (1000 trames/s au total) envoyées pendant l'attaque

18. https://github.com/chopengauer/nrf_analyze/blob/master/syma_joy.py

19. https://framagit.org/tigre-bleu/predator-drone/tree/master/tools/nrf24_logger

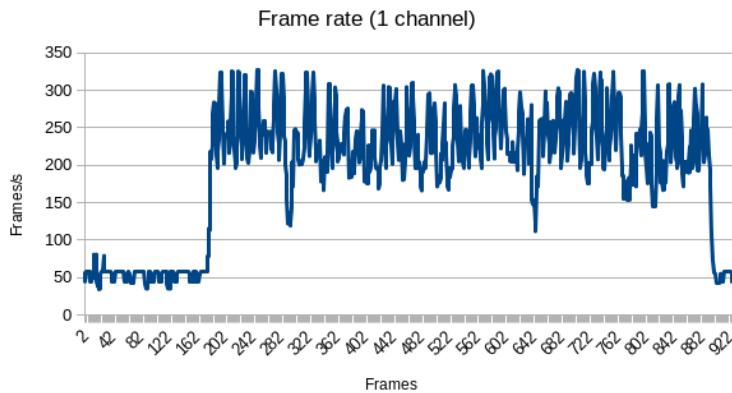


Figure 25 – *Frame-rate* sur un canal pendant une attaque

Ces taux de trames sont indicatifs, car on pourrait avoir atteint les limites de performance du nRF24L01+ en réception. La grande variation du taux pendant l'attaque nous semble confirmer cette hypothèse. Néanmoins nous pensons que l'ordre de grandeur est correct.

Nous avons également visualisé la différence de nombre de trames dans GnuRadio. Pour voir ces résultats, nous vous invitons à voir la fin de la vidéo²⁰ que nous avons faite dans la volière de l'ENAC.

Une caractérisation plus rigoureuse de l'attaque nécessiterait de passer par GnuRadio et de créer un traitement capable de compter les trames Syma.

3.7 Détection et prévention de l'attaque

L'attaque peut être difficile à comprendre du point de vue du pilote qui perd le contrôle d'un seul coup sans aucune indication d'un quelconque problème. En effet, si l'attaquant est suffisamment fin pour mettre sa manette dans une position proche de la position courante du drone, on ne note aucune interférence sur le drone (il ne tombe pas).

Un observateur disposant d'un SDR ou d'un moniteur spécialement développé (sur le principe de notre outil de log radio) pourrait facilement constater qu'une attaque est en cours en observant un nombre de trames beaucoup plus élevé que la normale.

Il n'y a pas de moyen de protection simple et efficace contre cette attaque.

On pourrait imaginer brouiller les canaux du drone mais le contrôle serait perdu pour tout le monde. La seule solution serait d'avoir un brouillage extrêmement précis qui s'interromprait uniquement le temps d'émettre une trame légitime. Nous n'avons pas de connaissances suffisamment précises dans ce domaine pour juger de la possibilité d'un tel brouillage. Ce serait une direction à creuser par la suite.

On pourrait également imaginer une “contre-attaque” : si une attaque est détectée, le pilote légitime augmente aussi le nombre de trames envoyées via un outil similaire au nôtre. Le résultat d'une telle bataille est difficile à prévoir car il pourra y avoir des collisions sur les trames. Le plus probable est que le contrôle du drone soit perdu pour tout le monde.

20. https://pe.ertu.be/video-channels/tls_sec/videos

4 Embarquement de notre outil sur un drone prédateur

L'outil que nous avons développé permettait donc de prendre le contrôle de deux drones de loisir : le Parrot AR.Drone 2.0 et le Syma X5C-1. Nous avons ensuite cherché à l'embarquer sur un drone prédateur.

4.1 Installation sur un Raspberry Pi Zero W

Pour cela, nous avons retenu l'utilisation d'un Raspberry Pi Zero W. Notre choix s'est porté sur cette carte car celle-ci fonctionne sous Linux (Debian Stretch) et qu'elle dispose d'une connectivité WiFi et Bluetooth intégrée. Elle embarque aussi une connectivité SPI, qui sera utilisée par le module radio nRF24L01+ nécessaire au piratage du drone Syma. Enfin, cette carte est très petite, donc légère, et peut facilement être embarquée. Concernant l'alimentation électrique, nous avons utilisé un cable microUSB pour le débogage et une batterie pour les tests en vol.



Figure 26 – Raspberry Pi Zero W

Il est bon de noter qu'un connecteur de caméra CSI est intégré au Raspberry Pi Zero W, ce qui permettra éventuellement à l'avenir d'intégrer une caméra à notre outil. Un bus I2C est aussi disponible, ce qui permettra éventuellement d'ajouter un écran intégré pour indiquer l'état de la batterie en vol.

4.1.1 Contrôle distant de l'outil

Comme nous le verrons après, la puce WiFi intégrée au Raspberry Pi était utilisée pour l'attaque de l'AR.Drone. Il nous fallait donc envisager une autre solution pour contrôler la carte de manière distante.

La seule autre option que nous avons entrevu était de mettre en place un réseau Bluetooth PAN (*Personal Area Network*). Ainsi, nous avons configuré le Raspberry Pi pour qu'il se comporte comme un *Network Access Point* (NAP) avec pour adresse IP 172.20.1.1 sur le réseau 172.20.1.0/24. L'ordinateur de contrôle de celle-ci devait simplement être appairé avec le Raspberry Pi, s'y connecter en Bluetooth et paramétrier son IP sur ce réseau.

Grâce à ce réseau Bluetooth NAP, nous pouvions accéder au Raspberry Pi en SSH. La seule critique que nous voyions était celle de la portée. En effet, le Bluetooth a souvent une portée aux alentours de 10 mètres. Cependant, selon l'adaptateur, la portée peut atteindre 100 mètres.

4.1.2 Compilation de la bibliothèque pyRF24

Afin d'installer notre outil sur le Raspberry Pi, nous avons dû installer la bibliothèque pyRF24, utilisée pour contrôler la puce nRF24L01+. Pour cela, il fallait compiler la bibliothèque. Cependant, en raison des faibles capacités calcul et mémoire de la carte, nous avons d'abord tenté une cross-compilation, sans succès. Après réflexion, nous avons simplement créé un fichier d'échange *swap* sur le Raspberry et avons compilé localement la bibliothèque. Cette seconde solution a bien fonctionné.

4.1.3 Problème de pilote de puce WiFi

Le Raspberry Pi Zero W intègre une puce WiFi. Celle-ci ne supporte pas le mode *monitor*. Ainsi, pour pouvoir perpétrer l'attaque sur les AR.Drone, il nous a fallu installer un dongle USB WiFi. Nous avons testé 3 dongles WiFi :

- Un Dlink DWA-131, qui n'était pas compatible avec PyRIC. En effet, son *driver* est trop ancien et n'est pas compatible avec nl80211²¹, qui est le nouvel en-tête de gestion d'interface sans fil sous Linux.
- Un Ralink MT7601U²² pour lequel l'attaque par désauthentification ne fonctionnait pas : le délai de reconnexion du vrai pilote du drone était trop court.
- Un TP-Link TL-WN823N²³ avec lequel aireplay-ng ne fonctionnait pas au premier abord, pour la même raison que précédemment (Ralink MT7601U).

Cependant, après quelques recherches, nous avons pu observer que le *driver* chargé par le Raspberry Pi n'était pas le *driver* officiel. En effet, celle-ci chargeait le *driver* 8192cu au lieu du rt18192cu. Après une tentative de mise à jour de Debian et du noyau Linux pour installer la dernière version du driver, nous avons découvert, grâce à un forum²⁴, que le pilote officiel était sur liste noire dans le fichier /etc/modprobe.d/blacklist-rt18192cu.conf. Après désactivation de ceci, le pilote était bien chargé et l'attaque fonctionnait.

Finalement, les cartes WiFi furent utilisées de la manière suivante :

- Le dongle TP-Link permettait de perpétrair l'attaque par désauthentification WiFi;
- La puce WiFi intégrée permettait de se connecter au drone piraté.

4.1.4 Utilisation du cockpit ardrone-webflight

Lors des premiers tests de prise de contrôle d'un AR.Drone, nous avons été confrontés aux faibles capacités du processeur de le Raspberry Pi Zero W. En effet, le cockpit que nous utilisions pour contrôler le drone piraté (ardrone-webflight) était trop lourd pour fonctionner convenablement. La vidéo envoyée par le drone était très sacadée et les commandes de contrôle prenaient effet avec quelques secondes de retard, ce qui n'était pas acceptable.

Nous avons donc décidé de déporter le cockpit sur l'ordinateur controlant le Raspberry Pi en instaurant un routage des paquets réseaux destinés au drone sur le Raspberry Pi. Ainsi, la chaîne de contrôle était celle ci-dessous :

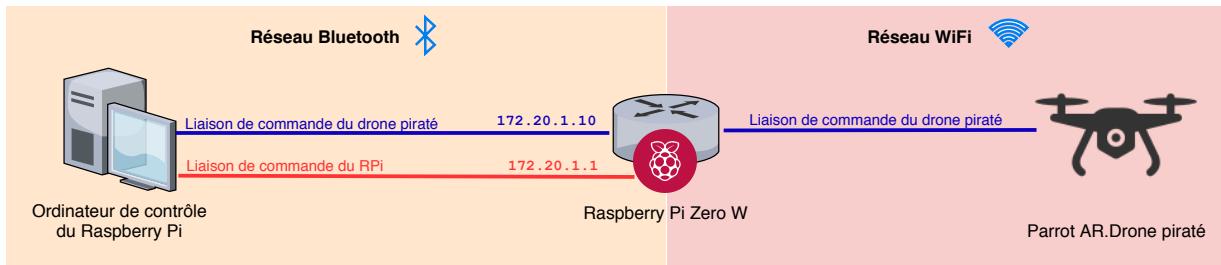


Figure 27 – Schéma fonctionnel du cockpit de piratage de l'AR.Drone

Pour mettre en place ce routage, nous avons d'abord pensé à instaurer des règles iptables transférant le trafic seulement destiné au drone. Cependant, après une étude Wireshark du trafic transmis pour contrôler le drone, nous avons pu observer divers flux transiter :

- Les commandes de l'AR.Drone sont envoyées en UDP sur le port 5556 du drone
- La vidéo est échangée en TCP, envoyée depuis le port 5555 du drone

21. <https://wireless.wiki.kernel.org/en/developers/documentation/nl80211>

22. Puce Realtek RTL8192SU

23. Puce Realtek RTL8192CU

24. <https://www.raspberrypi.org/forums/viewtopic.php?t=224931>

Ces choix de conception nous ont quelque peu étonnés, mais nous ne nous apesentirons pas sur cela ici. Après quelques tests non fructueux et une réflexion plus approfondie, nous avons conclu qu'une autre solution, plus simple, était envisageable.

Celle-ci consistait à ajouter une adresse IP à l'interface pan0 (réseau Bluetooth avec le pilote du Raspberry) et à transférer tous les paquets reçus sur cette adresse IP au drone. Cette solution permettait de donner l'illusion d'un accès transparent à l'AR.Drone, en laissant passer les accès FTP et telnet, eux aussi permis par défaut sur ce drone.

Cette solution fonctionne très bien. Les règles iptables mises en place sont énoncées ci-dessous, où 172.20.1.10 est l'adresse IP supplémentaire de l'interface pan0 et wlan0 l'interface WiFi avec laquelle nous nous connectons au réseau WiFi du drone. Les variables \$IP_PARROT et \$IP_RÉCUPÉRÉE_AVEC_DHCLIENT sont calculées par notre outil.

```
1 iptables -t nat -A PREROUTING -d 172.20.1.10 -j DNAT --to $IP_PARROT
2 iptables -t nat -A POSTROUTING -o wlan0 -j SNAT --to $IP_RÉCUPÉRÉE_AVEC_DHCLIENT
```

4.1.5 Prise de contrôle du Syma X5C-1 et utilisation distante d'une manette

Lors du passage de notre outil sur le Raspberry Pi Zero W, nous avons été confronté au problème de pilotage du drone Syma piraté. En effet, notre outil utilisait un joystick connecté en USB pour ce pilotage.

Nous avons donc modifié notre script pour utiliser le programme `usbip`. Celui-ci permet de partager un périphérique USB à travers le réseau :

- L'ordinateur qui partage son périphérique est le serveur ;
- L'ordinateur client se connecte au serveur. Un périphérique est alors créé dans `/dev`, et est présenté de manière identique à celui présent sur le serveur.

Cette solution fonctionne très bien.

4.2 Tests finaux de l'outil embarqué

Lorsque notre script était stable sur le Raspberry Pi Zero W, nous sommes allés effectuer des tests en vol dans la volière de l'ENAC. Pour cela, nous avons monté le Raspberry Pi Zero W ainsi qu'une batterie et le module nRF24L01+ sur une carène d'AR.Drone, comme le montre la photo ci-contre.

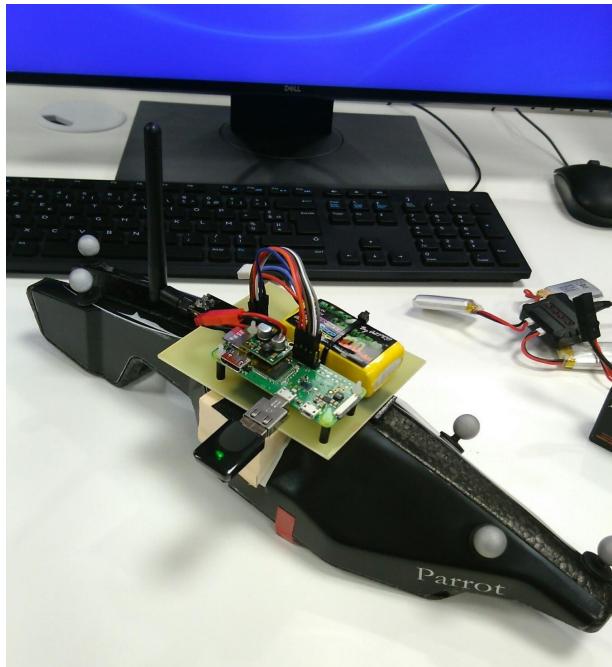


Figure 28 – Montage du drone prédateur

Nous avons ensuite, avec l'aide de Messieurs Xavier Paris et Yannick Jestin, enseignants-chercheurs ENAC, configuré un AR.Drone pour être le drone prédateur. Ce dernier utilisait Paparazzi UAV²⁵ pour effectuer un vol stationnaire. La localisation du drone dans la volière était effectuée grâce à la technologie OptiTrack et à des réflecteurs placés sur la carène du drone (petite boules grises).



Figure 29 – Drone prédateur



Figure 30 – Drone piraté

Lors de ces tests, nous avons pu utiliser le système de captation vidéo de la volière, ce qui nous a permis de monter deux vidéos présentant les attaques perpétrées :

- Attaque sur le Parrot AR.Drone 2.0 :
<https://pe.ertu.be/videos/watch/54cb4bff-c321-4030-ad70-543e044f7b74>
- Attaque sur le Syma X5C-1 :
<https://pe.ertu.be/videos/watch/14ae8a25-1c56-4ab7-91fe-47d0ec886a59>

25. Paparazzi UAV est un projet *open-source* développé par l'ENAC, dans le but de fournir des systèmes autopilotes. Le programme peut s'installer sur un AR.Drone. Pour plus d'informations, voir https://wiki.paparazziuav.org/wiki/Main_Page.

5 Étude du drone PNJ Discovery

Nos attaques étant fonctionnelles sur les deux drones que nous voulions étudier, nous avons commencé à faire de la rétro-ingénierie d'un troisième modèle en notre possession, un PNJ Discovery. Il s'agit d'un autre drone de loisir RF opérant sur 4 canaux de la bande 2.4 GHz.



Figure 31 – Drone PNJ Discovery

5.1 Démarche

Ne souhaitant pas refaire la même chose que pour le Syma X5C-1, nous avons décidé d'utiliser une approche alternative n'utilisant pas de moyen radio pour faire l'analyse du protocole : une approche semi-invasive, en se connectant directement aux circuits de la télécommande.

Nous n'avons pu mener à terme cette partie dans le temps du projet, mais nous avons tout de même pu avancer. Le chapitre suivant montre l'avancement de notre analyse.

5.2 Identification des contrôleurs

5.2.1 Sur le drone

Une fois le carénage du drone retiré, on peut voir une inscription UD U817B et une référence à 2013 sur la carte mère du drone. Un peu de recherche sur Internet²⁶ nous indique :

```

1 // Known UDI 2.4GHz protocol variants, all using BK2421
2 // * UDI U819 coaxial 3ch helicoper
3 // * UDI U816/817/818 quadcopters
4 //   - "V1" with orange LED on TX, U816 RX labeled '' , U817/U818 RX labeled 'UD-U817B'
5 //   - "V2" with red LEDs on TX, U816 RX labeled '' , U817/U818 RX labeled 'UD-U8170G'
6 //   - "V3" with green LEDs on TX. Did not get my hands on yet.
7 // * U830 mini quadcopter with tilt steering ("Protocol 2014")
8 // * U839 nano quadcopter ("Protocol 2014")

```

On remarque aussi une puce ARM, probablement le CPU, ainsi qu'une autre puce avec une inscription difficile à lire.

26. <https://www.deviationtx.com/forum/protocol-development/2551-udi-r-c-u816-u818-protocol?start=60#32387>

5.2.2 Sur la télécommande

La télécommande est séparée en 2 parties, une carte mère et une carte fille. Une grande puce sans inscription est présente sur la carte mère sur laquelle arrivent des pistes reliées aux boutons/joysticks et partent des pistes vers la carte fille et l'écran. On peut supposer que cette grande puce est le micro-contrôleur principal de la télécommande et la carte fille est le *transceiver* radio.

La carte fille est soudée à la carte mère à 90°. Il y a 8 points de soudure assez facilement accessibles. Nous supposons que l'on peut y trouver un bus SPI. Nous avons donc soudé des fils pour pouvoir mettre un analyseur logique.

Nous remarquons que ces puces filles sont les mêmes sur la télécommande et sur le drone. Une inscription est présente dessus mais illisible à cause d'un gros point de colle. L'inscription est à côté de ce qui semble être le cristal d'horloge.

En recherchant sur internet des personnes ayant déjà fait du *reverse-engineering* de drone, nous découvrons un article de blog²⁷ avec un circuit qui ressemble très fortement au nôtre.

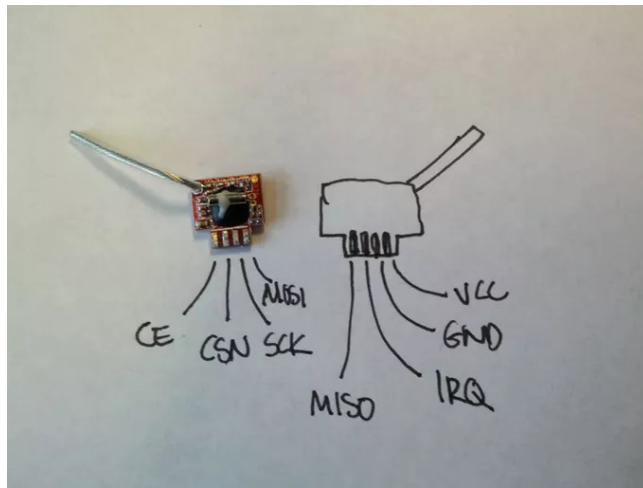


Figure 32 – Puce Radio

On peut donc supposer que la puce radio est une puce BK2421²⁸, très similaire à une puce nRF24L01+²⁹.

5.3 Analyse du bus SPI

5.3.1 Connexion au bus

Nous avons soudé, sur la télécommande, des fils qui vont nous permettre de connecter ce qu'on suppose être le bus SPI à un analyseur logique.

Après une première soudure ratée qui a arrachée une partie des pistes, la manette a pu être réparée grâce à Monsieur Michel Gorraz de l'ENAC. Il nous a également soudé les fils souhaités sur les pistes menant à la puce radio et partant du microcontrôleur principal de la télécommande. Nous l'en remercions chaleureusement.

5.3.2 Analyse grâce à un analyseur logique

Nous avons ensuite connecté sur ces fils un analyseur logique Saleae et ouvert le logiciel d'analyse. Nous avons rapidement pu constater qu'il y avait effectivement 4 pins ressemblant très fortement à un bus SPI (voir capture d'écran sur la page suivante).

27. <https://www.hackster.io/geekphysical/controlling-toy-quadcopter-s-with-arduino-6b4dcf>

28. http://www.bekencorp.com/en/Botong.Asp?Parent_id=2&Class_id=8&Id=13

29. <https://forbot.pl/forum/applications/core/interface/file/attachment.php?id=894>

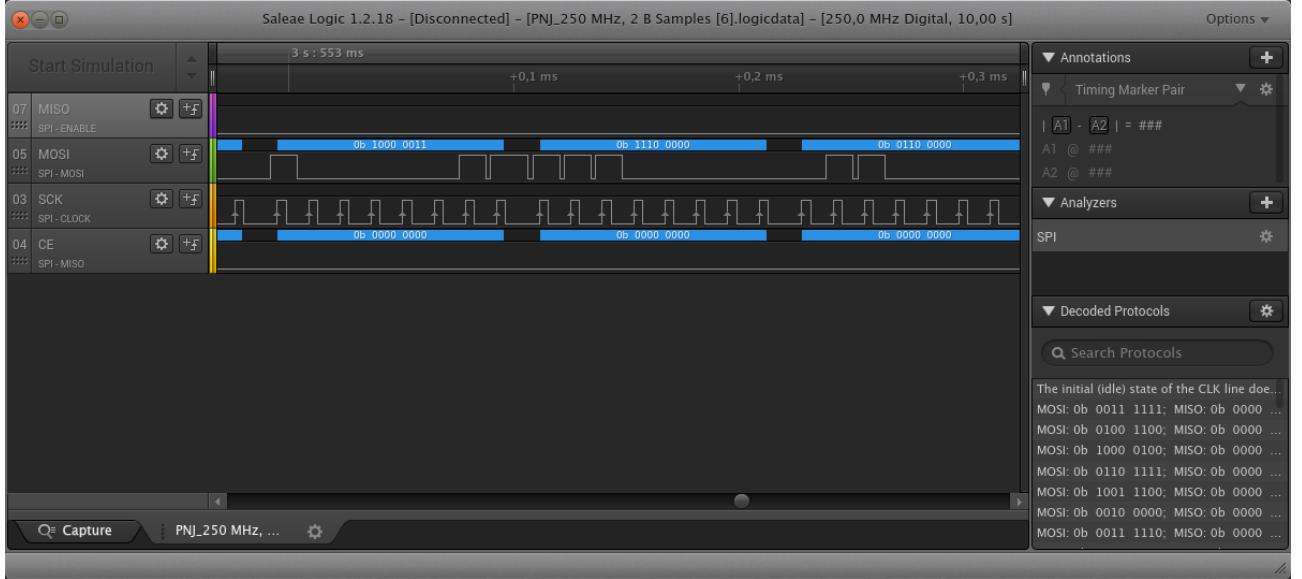


Figure 33 – Analyseur Logique

En rajoutant un analyseur, nous pouvons voir ce qui passe sur le bus et le comparer à la *datasheet* du module BK2421³⁰. C'est ce que nous présentons sur le tableau suivant :

Time [s]	Packet ID	MOSI	COMMAND	COMMAND DATA	TX PAYLOAD	MISO
0.845679976000000		0 0010 1010	W REGISTER (OBSERVE_TX)			0000 1110
0.845796416000000		0 0010 0000				0000 0000
0.845912864000000		0 1001 0010				0000 0000
0.846029312000000		0 0100 0000				0000 0000
0.846161680000000		0 0000 0001				0000 0000
0.846278128000000		0 0010 0000				0000 0000
0.846397552000000	1	0011 0000	W REGISTER (TX_ADDR)	0x201409220		0000 1110
0.846514000000000		1 0010 0000				0000 0000
0.846630448000000		1 1001 0010				0000 0000
0.846746888000000		1 0100 0000				0000 0000
0.846863336000000		1 0000 0001				0000 0000
0.846979776000000		1 0010 0000				0000 0000
0.847174864000000		2 0000 0111	R REGISTER (STATUS)			0000 1110
0.847286328000000		2 1111 1111				0000 1110
0.847418704000000		3 1110 0001	FLUSH_RX			0000 1110
0.847547096000000		4 0010 0111	W REGISTER (STATUS)			0000 1110
0.847663536000000		4 0111 0000				0000 0000
0.847812832000000	5	0010 0101	W REGISTER (RF_CH)	70		0000 1110
0.847929272000000		5 0100 0110				0000 0000
0.848062632000000		6 1110 0001	FLUSH_RX			0000 1110
0.848197992000000		7 1010 0000	W_TX_PAYLOAD			0000 1110
0.848314432000000		7 1100 0100			C4	0000 0000
0.848430872000000		7 0000 0000			0	0000 0000
0.848547320000000		7 1111 1111			FF	0000 0000
0.848663760000000		7 1110 0000			E0	0000 0000
0.848780208000000		7 0110 0000			60	0000 0000
0.848896648000000		7 0010 0000			20	0000 0000
0.849013096000000		7 0010 0000			20	0000 0000
0.849145464000000		7 0000 1111			F	0000 0000
0.849271856000000	8	0010 0000	W REGISTER (CONFIG)			0000 1110
0.849388304000000		8 0000 1110				0000 0000
0.852209400000000		9 0000 0111	R REGISTER (STATUS)			0000 1110
0.852320856000000		9 1111 1111				0000 1110
0.852453232000000		10 1110 0001	FLUSH_RX			0000 1110
0.852581616000000	11	0010 0111	W REGISTER (STATUS)			0000 1110

Figure 34 – Analyse du bus SPI par tableau

La comparaison nous indique entre autres la liste des canaux utilisés. Une analyse avec un SDR nous confirme de l'activité sur les canaux 70, 63, 56 et 48.

Malheureusement, nous nous sommes arrêtés là faute de temps. Cependant, nous pourrions tout à fait poursuivre l'analyse du *dump* en nous focalisant sur le trafic lorsque le drone est déjà appairé, dans le but de comprendre le protocole PNJ.

30. https://framagit.org/tigre-bleu/predator-drone/blob/master/doc/PNJ%20Drone/SPI_RC_ON_and_Binding.ods

6 Conclusion

6.1 Résultats et limitations

L'objectif de ce projet était de développer une solution embarquée permettant de détourner un drone en vol. Nous avons implémenté un outil permettant la capture des drones WiFi Parrot AR 2.0 et 2.4 GHz Syma X5C-1.

Nous avons montré qu'une capture “propre” était possible aux conditions suivantes :

- Pour le Parrot AR 2.0, que le code de `ardrone-webflight` soit modifié afin d'inhiber l'ordre d'atterrissement au moment de la connexion au drone.
- Pour le Syma X5C-1, que la manette du pilote du prédateur soit proche de la position courante du drone volé au moment de la prise de contrôle.

L'outil `predator-drone` est suffisamment léger pour pouvoir fonctionner sur un Raspberry Pi Zero W. Nous avons donc montré qu'il est possible d'embarquer notre solution sur un autre drone aéroporté.

Le code fourni a été écrit dans un souci de modularité. En effet, nous avons intégré les attaques Parrot et Syma X5C-1 sous formes de modules d'un logiciel principal. Cette approche permettra éventuellement d'étendre le périmètre de l'outil si des contributeurs souhaitaient rajouter le support d'autres drones.

Nous devons néanmoins noter les limitations suivantes :

— **Autonomie du Raspberry Pi :**

L'ensemble Rasberry Pi Zero W avec un dongle WiFi et un module radio nRF24L01+ tient environ 3 heures avec une batterie de 800 mAh et des attaques occasionnelles. C'est une performance suffisante pour un grand nombre de cas mais pourrait être une limitation pour une opération plus longue sur le théâtre d'opérations.

— **Autonomie du drone :**

L'AR.Drone 2.0 sur lequel nous avons embarqué notre matériel prédateur n'offre que quelques minutes d'autonomie en vol. Dans un cadre d'utilisation réaliste, il serait nécessaire de monter le matériel prédateur sur une plateforme plus performante. Nous pouvons noter qu'il n'y a pas de nécessité de faire du vol stationnaire du moment que la cible reste à portée radio. Ainsi, il est possible d'imaginer utiliser une voilure fixe, souvent dotée d'une meilleure autonomie.

— **Portée de l'attaque :**

Le fait d'embarquer l'outil `predator-drone` sur un drone prédateur permet d'approcher celui-ci de sa cible et donc de toujours se placer à portée. Le corollaire est qu'il faut maintenir la connexion entre le sol et le drone prédateur. Le module WiFi intégré et le port USB du Raspberry Pi Zero W étant utilisés pour l'attaque WiFi, notre solution utilise le bluetooth du Raspberry Pi pour cet effet. Nous sommes donc dépendants de sa portée (10 à 20 mètres pour du Classe 2 comme sur les Raspberry, 100 mètres pour du classe 3). Une solution pourrait être d'utiliser une connectivité 3G/4G mais il faudrait alors ajouter un modem.

— **Puissance nécessaire à `ardrone-webflight` :**

Nous comptions initialement embarquer le logiciel `ardrone-webflight` directement sur le Raspberry Pi Zero W. Ainsi, l'attaquant n'aurait eu besoin que d'une connexion SSH et d'un navigateur pour faire l'attaque. Cependant le logiciel –développé en Node.js– s'est avéré trop gourmand en ressources pour la carte. Afin de réussir cette étape, il faudrait envisager des cartes plus puissantes sans compromettre le poids de l'ensemble.

6.2 Perspectives et améliorations

À court terme, l'outil predator-drone pourrait être enrichi par l'ajout du support de drones supplémentaires, notamment en terminant le travail initié sur le drone PNJ Discovery.

Plus globalement, une amélioration possible serait de réaliser la connexion entre le drone prédateur et le contrôle au sol via une connexion 3G/4G. En effet, nous pourrions ainsi jouir d'une portée illimitée. On pourrait alors imaginer plusieurs équipes locales envoyer des drones autonomes couvrir des zones différentes et que la partie "attaque" soit opérée dans un centre de contrôle unique fixe, à des centaines de kilomètres. Cette amélioration serait utile uniquement pour des drones à capturer qui peuvent se contrôler sans vue directe (par exemple le Parrot AR.Drone 2.0).

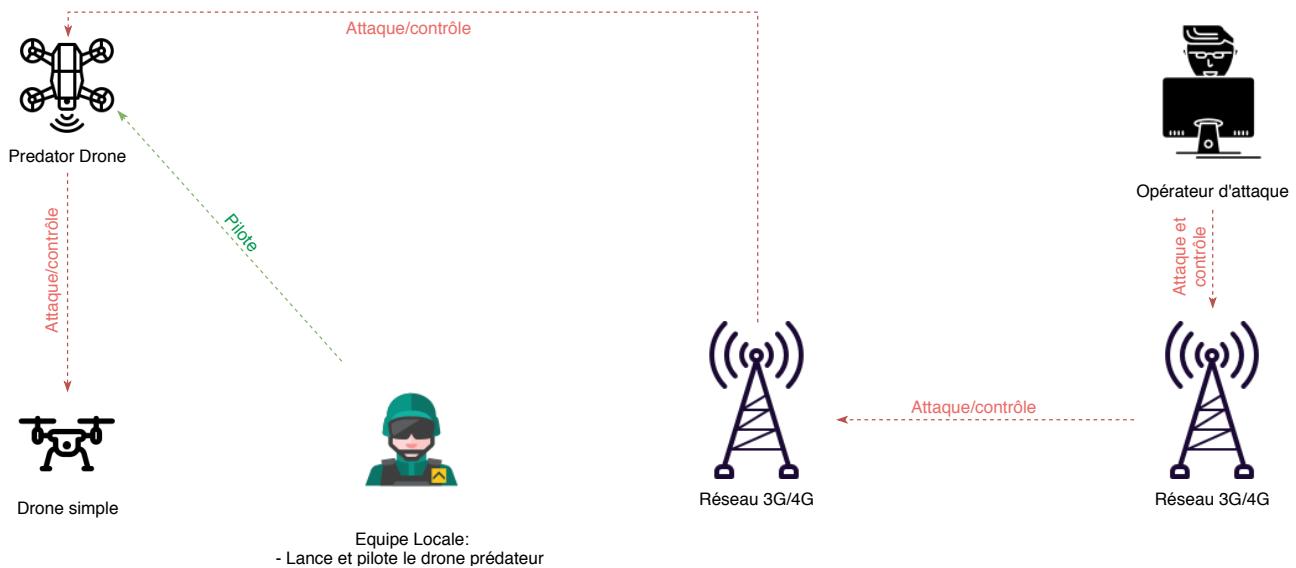


Figure 35 – Contrôle de drone autonome

Pour les autres, il serait toujours nécessaire que l'équipe locale puisse choisir les commandes en fonction de leur vision du drone à capturer.

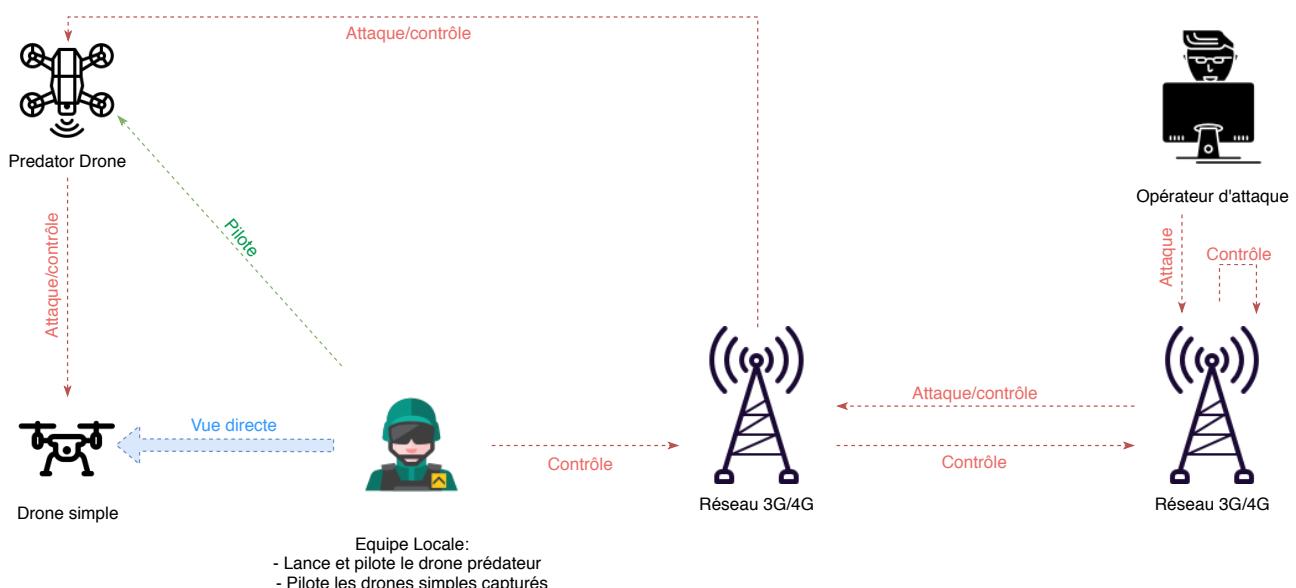


Figure 36 – Contrôle de drone simple

6.3 Compétences aquises

Ce projet nous a permis d'acquérir de nombreuses compétences. En particulier sur :

- L'utilisation du mode *monitor WiFi*
- Les attaques par désauthentification WiFi (utilisation de la suite Aircrack-ng)
- L'utilisation de la bibliothèque Scapy
- L'utilisation d'un SDR
- L'analyse de signaux RF
- L'utilisation du module nRF24L01+ et son protocole de liaison de données
- L'espionnage de bus SPI

Nous avons également pu mettre en pratique des techniques de base d'électronique : soudure, etc.

Remerciements

Nous remercions tout d'abord l'ENAC pour nous avoir permis d'accéder à la volière de drones et à tout le matériel nécessaire à nos expérimentations. Nous souhaitons remercier en particulier Messieurs Nicolas Larrieu, Xavier Paris et Michel Gorraz.

Nous remercions également Monsieur Damien Roque de l'ISAE pour ses explications sur la modulation GFSK et sur GnuRadio, ainsi que Monsieur Benoit Morgan de l'INP ENSEEIHT pour ses conseils.

Enfin, nous tenons à remercier globalement l'ensemble des intervenants de la formation TLS-SEC³¹, qui bien qu'exigeante est très enrichissante.

31. <https://tls-sec.github.io/>