# Projet 3A - Piratage d'objets connectés - Lampe Philips Hue

Alvaro LEFEVRE - Thibaud VIAL - Jeremie PERES

23 mars 2017

CentraleSupélec

# Table des matières

# 1 Introduction

En 2012, Philips a lancé les premières ampoules LED connectées : les lampes Hue. Il s'agit d'ampoules LED pouvant changer de couleur et de luminosité et controllables soit via une télécommande soit via des applications de smartphone utilisant le réseau WiFi. Les ampoules sont vendues à l'unité ou avec un bridge : une interface entre le réseau WiFi de la maison et l'ensemble des lampes Hue présentes. De cette façon, l'utilisateur peut gérer à distance l'éclairage d'une pièce de sa maison, automatiser des allumages ou extinctions, changer l'ambiance d'une pièce, etc.

Le succès croissant de ces lampes a amené de nombreux chercheurs et white hat hackers à s'intéresser à la sécurité de ces lampes. En effet, si on y réfléchit un peu, la prise de contrôle mal intentionnée de ces lampes peut poser de nombreux problèmes de sécurité : extinction inopportune de l'éclairage dans un hôpital, déclenchement de crises épileptiques en faisant varier la fréquence d'allumage/extinction, extraction de données et de nombreuses autres attaques insoupçonnées [1] .

Une publication réalisée par Eyal Ronen et Colin O'Flynn nous a particulièrement intéressé et a suscité pas mal d'émoi au sein et même en dehors de la communauté scientifique. Leurs travaux de recherche ont par ailleurs poussé Philips à réaliser des mises à jour de leurs lampes. Ils ont démontré qu'en piratant une lampe et en réalisant une attaque OTA d'une lampe à une autre, ils pouvaient installer un worm sur une grande étendue de lampes de sorte à prendre le contrôle de toutes ces lampes. En raisonnant sur une ville de la taille et densité de Paris, ils estiment qu'avec une densité critique de lampes à travers la ville, ils pourraient faire chuter le réseau électrique en forçant régulièrement l'allumage puis l'extinction simultanés de toutes ces lampes [2].

Bien que nous sachions que la sécurité de ces lampes avait sûrement été grandement renforcée par Philips suite aux nombreux travaux de recherche, nous avons voulu nous pencher sur la prise de contrôle de ces lampes. En effet, ces lampes sont tout d'abord financièrement accessibles et par ailleurs s'adressent au grand public ce qui accroît les risques liés à leur sécurité et donc, de notre point de vue, leur intérêt.

# 2 Objectifs

Notre objectif basique est de prendre le contrôle à distance d'une lampe Philips Hue sans accès à sa télécommande ou à son bridge. Toutefois, en raison des évolutions de notre travail, et notamment des barrières rencontrées, cet objectif a évolué à plusieurs reprises.

On peut résumer notre travail en 4 grandes étapes :

— attaque par rejeu du signal émis par la télécommande à la lampe (partie HackRF) ;
— analyse et décodage des trames Zigbee (FreakUSB sur lampe) ;
— prise de contrôle d'un bridge Hue (root)
— brouillage intelligent des communications entre télécommande et lampe

# 3 Présentation du sytème Philips Hue

En 2014, Philips commercialise la première version de sa gamme Hue pour laquelle Philips a mis en œuvre la technologie ZigBee Light Link (ZLL). Un réseau ZigBee est maillé et constitué des nœuds suivants :

— un élément central appelé coordinateur qui contrôle le réseau. Il joue aussi le rôle de passerelle (gateway) vers le monde IP. Ici, c'est le Philips Hue Bridge
— des éléments de type routeur qui permettent de relayer les informations vers les nœuds les plus éloignés en plus de leurs fonctions propres. Ici, ce sont les ampoules Hue
— des éléments end node qui n'assurent que des fonctions applicatives. Ici, ce sont les télécommandes


FIGURE 1 – Bridge Philips Hue, ampoules et dimmer switch

## 3.1 Ampoule Philips Hue

En 2014, Philips commercialise la première version de sa gamme Hue. Philips HUE est une ampoule LED qui permet de créer et de personnaliser l'éclairage de son domicile sans fil. L'ampoule Hue présente deux fonctionnalités principales :

— Contrôler intelligemment son éclairage depuis son smartphone ou les télécommandes dimmer switch en connectant les ampoule au Hue Bridge : paramétrer des minuteurs, des notifications, des alarmes.
— Faire varier facilement l'intensité lumineuse

L'ampoule HUE permet d'économiser jusqu'à 80% d'énergie par rapport aux ampoules traditionnelles. Fin 2015, la version 2 est commercialisée, une mise à jour importante car elle intègre de nombreuses nouvelles fonctionnalités, notamment l'intégration dans de multiples produits (Nest, Somfy, Honeywell, Logitech, etc. . . ), l'incontournable application IFTTT et l'application HomeKit d'Apple.

Caractéristiques :
Philips Hue white A19 Bulb

— Ampoule standard culot à vis - E27
— 9,5 W
— Consommation en mode veille 0,5 W
— Durée de vie nominale 25 000 heures

Flux lumineux Philips Hue White
— Flux lumineux 806 lm
— Température de couleur 2 700 K (blanc chaud)
— Lumière blanche à intensité réglable de qualité élevée
— Largeur de faisceau de 180 degré
— Supérieure à 80 IRC (Indice de Rendu des Couleurs)
Puissance absorbée
— 100-240 V CA / 50-60 Hz
— Tension de sortie : 5 V CC, 600 mA
— Consommation en mode veille : 0.1 W max

## 3.2 Bridge Philips Hue

Beaucoup de constructeurs d'ampoules connectées ont fait le choix du Bluetooth pour contrôler leurs luminaires. Philips, lui, emploie une liaison Zigbee, une liaison bas débit qui a l'avantage de consommer très peu d'énergie, de chiffrer toutes les communications et surtout d'offrir une bonne portée contrairement au Bluetooth, de 70 mètres environ, notamment grâce à la possibilité de créer un réseau maillé permettant d'étendre la zone de réception. Elle est donc bien adaptée à ce type de produits répartis dans toute la maison.

L'inconvénient de ce protocole Zigbee, c'est qu'il faut un appareil central pour piloter toutes les ampoules : le Philips Hue Bridge. Ce pont est le cœur du système et permet de connecter jusqu'à 50 lampes et 12 organes de commandes. Il est connecté au réseau local par câble Ethernet et diffuse un réseau sans fil pour les périphériques en utilisant la technologie ZigBee. Le Bridge Hue expose également des API REST côté réseau IP, ce qui permet à de multiples applications de piloter l'éclairage.

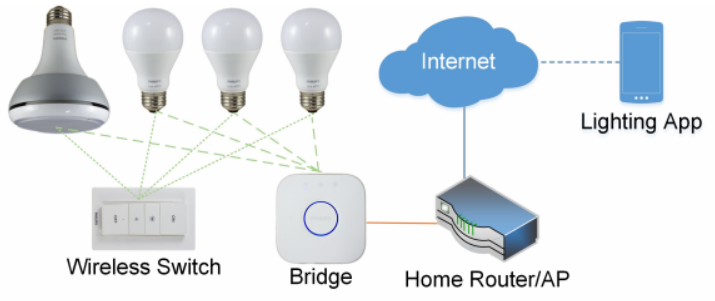


FIGURE 2 – Fonctionnement du Bridge Philips Hue

Caractéristiques :
— Maximum 50 ampoules ou luminaires par pont et 12 accessoires Philips Hue par pont
— Consommation d'énergie : 300 mA typ
— Protocole Zigbee Light link : 1.0 certifié
— Bande de fréquence 2 400 – 2 483,5 MHz

### 3.3 Philips Hue dimmer switch

Le dimmer switch est une petite télécommande portable à quatre boutons permettant :

- d'allumer les ampoules, mettre l'intensité au maximum ou changer de scénario avec le bouton allumer
- de régler l'intensité de l'éclairage avec les boutons augmenter/diminuer
- d'éteindre toutes les ampoules avec le bouton éteindre

Caractéristiques :

- Max 12 interrupteurs en l'absence d'autres accessoires dans le système
- Alimenté par des piles
- Portée : minimum 12 m intérieur
- Protocole Zigbee light link IEEE 802.15.4
- Bande de fréquence 2 400 – 2 483,5 MHz

## 4 Installation des outils

### 4.1 Méthode 1 : Installation automatique

La méthode la plus simple pour installer l'ensemble des outils GnuRadio est de passer par un script dédié pour Fedora et Ubuntu. Un script [3] est disponible.

```
# Creation d'un repertoire dedie
mkdir SDR
cd SDR
# Recuperation du script
wget http://www.sbrac.org/files/build-gnuradio
# Execution du script
chmod u+x build-gnuradio
./build-gnuradio
```

Le script prend un certain temps (de l'ordre de plusieurs heures), mais assure une installation propre et complète.

### 4.2 Méthode 2 : Installation manuelle

#### 4.2.1 PyBombs

PyBombs est un outil développé pour faciliter l'installation de GnuRadio. Bien que celui-ci soit disponible sur les dépôts officiels, il n'est pas toujours à jour. L'installation se fait comme suit [4] :

Installation de PyBombs via PIP

```
sudo pip install pybombs
```

Ajout de méthodes

```
pybombs recipes add gr-recipes
    git+https://github.com/gnuradio/gr-recipes.git
pybombs recipes add
    gr-etcetera git+https://github.com/gnuradio/gr-etcetera.git
```

#### 4.2.2 GRC - GnuRadio Companion

GnuRadio est un outil open-source massivement utilisé dans le domaine des télécoms. Il permet de programmer des modules en Python ou C/C++. GnuRadio dispose nativement de la plupart des fonctions utiles à la radio logicielle. Il est complété par GnuRadio-Companion (GRC), une interface graphique pour créer des graphes. Chaque bloc sous GRC représente un module GnuRadio. Un script, fourni par la communauté développant GnuRadio, permet d'installer l'ensemble des outils nécessaires.

http ://www.sbrac.org/files/build-gnuradio

#### 4.2.3 GR-OsmoSDR

Le paquet GR-OsmoSDR permet d'utiliser d'autres émetteurs/récepteurs que les USRP déjà inclus dans GnuRadio, notamment toutes les clés RTL-SDR et le HackRF One. L'installation se fait simplement avec PyBombs :

```
sudo pybombs install gr-osmosdr
```

# 5 Essai d'une attaque par rejeu

Le principe d'une attaque par rejeu est d'enregistrer le signal radio émis au moment de la commande, et de le rejouer au moment voulu. Ce type d'attaque est possible uniquement si la transmission se fait de manière unilatérale, et si le message est identique à chaque envoi (à opposer aux systèmes intégrant des "rolling code").

Pour recevoir puis émettre le signal enregistré, nous utiliserons le HackRF One de Great Scott Gadgets, conçu par Michael Ossmann [5].

Le HackRF présente principalement l'avantage de pouvoir recevoir et émettre sur la bande de fréquences allant de 1 MHz à 6 GHz.

## 5.1 Capture du signal

Pour capturer le signal, nous avons utilisé le logiciel GnuRadio via son interface graphique (GRC). Pour ce type d'attaque, nous avons décidé d'enregistrer les données dans un format brut (RAW I/Q). Le graphe utilisé est le suivant :

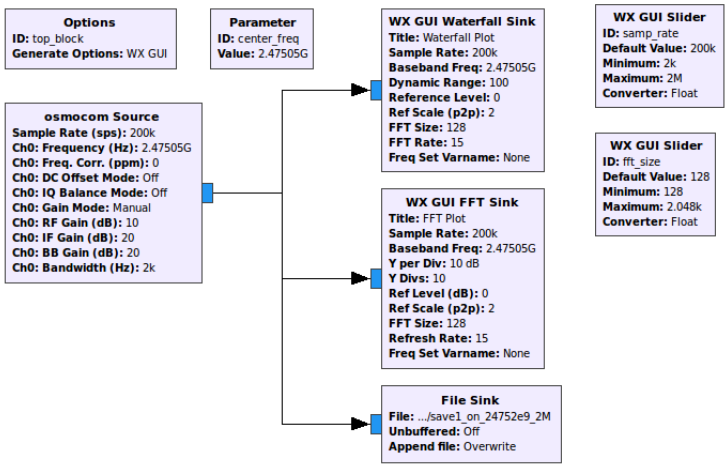


FIGURE 3 – GRC - Enregistrement du signal brut - Fichier hackrfReceiverSaveRawSignal.grc

**Le bloc Osmocom Source** permet de s'interfacer avec le HackRF (ou l'USRP). Le taux d'échantillonnage et la fréquence centrale sont réglés par les blocs samprate et centerfreq.

**Les blocs Waterfall Sink et FFT Sink** permettent d'afficher en temps réel le signal reçu par le bloc Osmocom Source. Cela nous permet de vérifier immédiatement que le signal est bien reçu, et de régler la fréquence centrale à l'aide du Slider.

**Le bloc File Sink** nous permet d'enregistrer le signal au format brut. Les échantillons I/Q sont non filtrés, ce qui a pour conséquence de générer des fichiers conséquents. L'étape d'enregistrement brut est donc utile uniquement lorsque l'on a la connaissance du moment d'émission du signal.

## 5.2 Rejeu du signal

L'objectif de cette attaque est d'utiliser le signal enregistré dans la partie précédente pour simuler la commande de la télécommande. C'est l'attaque la plus facile, car il n'est même pas nécessaire de démoduler et décoder le signal. Si elle fonctionne, il sera possible de contrôler la lampe en enregistrant simplement les quatre boutons de la télécommande.

Nous utilisons également le HackRF pour réémettre le signal brut enregistré à l'étape précédente. Le schéma GnuRadio utilisé est le suivant :

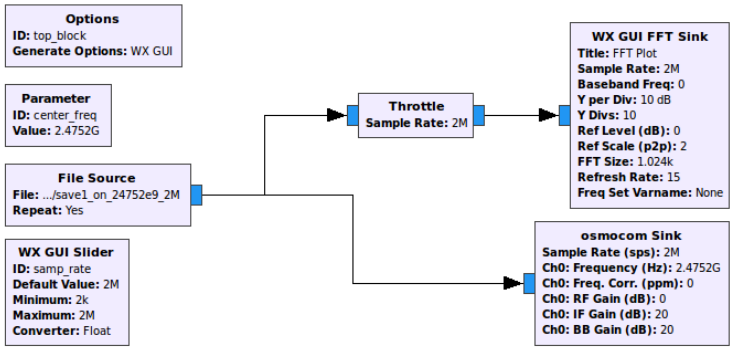


FIGURE 4 – GRC - Rejeu du signal brut - Fichier hackrfReplayRawSignal.grc

**Le bloc Throttle** permet de limiter l'échantillonnage au niveau du processeur. Cela évite que l'ordinateur calcule des échantillons qui ne seront pas utilisés.

**Le bloc Osmocom Sink** permet de s'interfacer avec la partie émission du HackRF.

Malheureusement, après plusieurs essais, nous avons pu constater que l'attaque par rejeu ne fonctionne pas avec les lampes Philips.

# 6 Analyse du signal

Nous avons ensuite décidé de nous intéresser au protocole utilisé par les lampes Philips, afin de tenter de le comprendre et de le reproduire. Nous avons testé deux méthodes, l'une avec le HackRF (aspect purement SDR) et l'autre avec une carte dédiée au protocole Zigbee.

## 6.1 Avec le HackRF One

Dans un premier temps, nous avons essayé de récupérer le signal avec le HackRF, avec GnuRadio. Nous avons utilisé le fichier suivant pour étudier le signal.

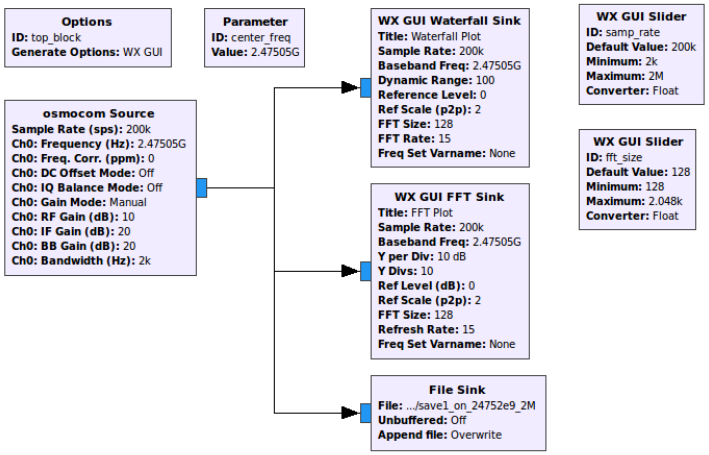


FIGURE 5 – GRC - Analyse du signal - Canal 25

Toutefois, malgré toutes nos tentatives, nous n'avons pas réussi à capturer le signal ni même à le voir à l'aide du HackRF. Il semblerait que le signal soit trop bref et de faible puissance. Par ailleurs, nous n'étions pas sûrs du canal utilisé par la télécommande. D'après les spécifications du constructeur, ils peuvent utiliser 16 canaux différents compris entre 2.405 et 2.48 GHz de largeur 3 MHz et espacés de 5 MHz entre chaque canal [6]. Le fait que cette partie du spectre soit celle dédiée au WiFi notamment ne nous facilitait pas non plus la tâche du fait de la "pollution" du spectre. Finalement, nous avons fini par trouver que le système ne faisait pas de frequency hopping et que le canal à 2.475 GHz était utilisé. Nous avons alors essayé de le capturer à l'aide d'un analyseur de spectre.

Le résultat de cette analyse apparait ci-dessous : nous pouvons voir la présence de 4 pics autour de 2475 MHz, témoins de la modulation OQPSK utilisée. Afin de parvenir à cette visualisation, nous avons réduit la largeur de bande à 2 MHz, accru la fft à 4001 points et réduit le durée d'échantillonnage.

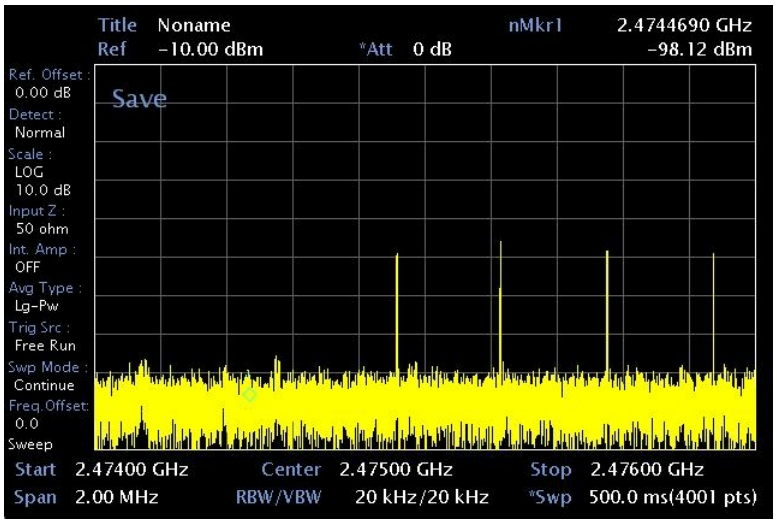


FIGURE 6 – Analyseur de spectre - capture à 2475 MHz

## 6.2 Avec le FreakUSB

Après l'échec du HackRF, nous nous sommes tournés vers une solution dédiée au protocole Zigbee. Le FreakUSB [7] est une plateforme compatible Arduino, et dédiée à l'émission et à la réception de trames 802.15.4. Le FreakUSB est programmé pour envoyer sur le port série les trames 802.15.4 qu'il reçoit. Un script python est chargé de récupérer les trames, et de les convertit en paquets PCAP. Ces derniers sont pris en charge par Wireshark, sur une interface virtuelle créée par le script python.

**Installation de la carte FreakUSB** La carte FreakUSB est composée d'un FTDI, d'un microcontrôleur Atmega et d'une puce RF Zigbee. Grâce au microcontrôleur Atmega, il est compatible avec l'IDE Arduino. Nous avons donc utilisé la librairie chibiArduino, qui intègre une couche 802.15.4, pour démoduler les trames Zigbee.

```
# Telechargement du depot Git
git clone
    https://github.com/freaklabs/sensniff-freaklabs
cd sensniff-freaklabs/host
# Compilation
make
# Demarrage du script sensniff avec l'interface ttyUSB0
python sensniff.py -d /dev/ttyUSB0



# Lancement de Wireshark avec ecoute de l'interface
sudo wireshark -k -i /tmp/sensniff
```

La carte FreakUSB est programmé avec un code de démonstration de la librairie chibiArduino. Cette dernière permet au microcontrôleur de commander la puce RF Zigbee, et de la configurer en mode Monitor. De cette manière, après sélection du canal, la puce va pouvoir recevoir toutes les trames 801.15.4, et ce même si elles ne lui sont pas destinées.
Le programme Arduino fonctionne avec un script Python lancé sur l'ordinateur. Celui-ci va communiquer avec l'Arduino grâce au port FTDI. Nous pouvons ainsi régler le canal d'écoute, pour reçevoir les informations de la télécommande.
Le signal est démodulé par la puce Zigbee, et est envoyé au script Python. Celui-ci va récupérer et convertir les paquets, pour les rendre compatible avec Wireshark. Une interface virtuelle est créée pour que Wireshark puisse s'y connecter.
Wireshark est un outil d'analyse de protocoles compatible avec Zigbee. Il nous a permis d'identifier les communications bidirectionnelles établies entre la télécommande et l'ampoule.

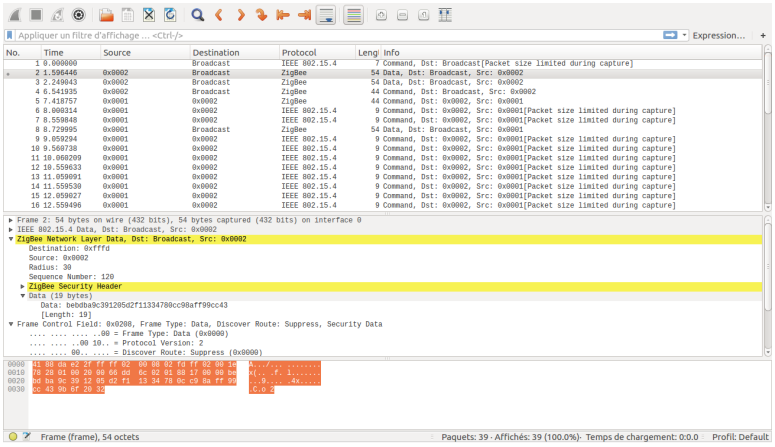


FIGURE 7 – Wireshark - Analyse des paquets

Grâce à Wireshark, nous avons pu établir que le protocole Zigbee utilisé implémentait une sécurité de type AES 128 bits. Après quelques recherches, nous avons pu nous rendre compte qu'il nous était impossible, dans le temps imparti, de casser ce type de clé. Nous nous sommes donc orientés vers d'autres alternatives.

# 7 Prise de contrôle root du bridge

Comme présenté dans la première partie, le système complet intègre, en plus des télécommandes et des ampoules, un bridge. Celui-ci fait l'interface entre le réseau Zigbee et la box internet. Il est équipé d'une connexion Ethernet (ainsi que d'une puce Wifi, découverte pendant le démontage, mais qui n'est actuellement pas activée). Cela lui permet d'être relié à internet, et d'être contrôlé par une application smartphone. Le bridge intègre un serveur web, qui permet la réception de requêtes pour connaitre l'état et contrôler les lampes. Nous avons donc voulu accéder à l'ensemble des fonctionnalités. Après quelques recherches,

il s'est avéré que le Bridge était en fait une plateforme Linux. Nous avons donc voulu devenir "root", à savoir avoir les droits d'effectuer toutes les modifications que nous voudrions.

Nous nous sommes basés sur les travaux de Colin O'Flynn [8]. Un port UART est accessible, et transmet des informations sur la séquence de démarrage du Bridge. En démontant le boitier, nous avons pu accéder à ce port.

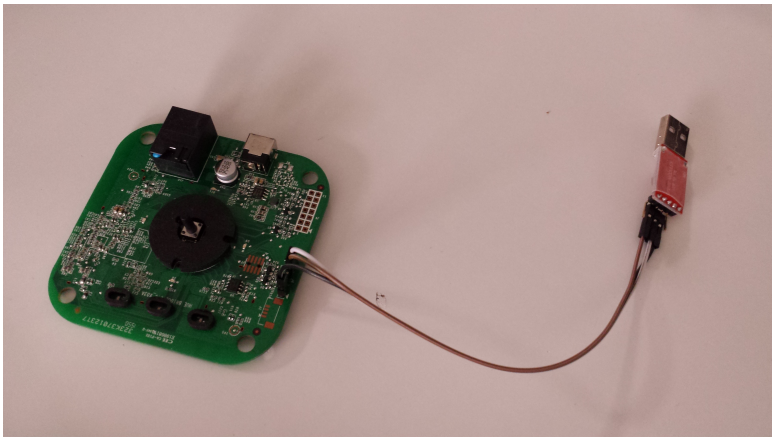

FIGURE 8 – Port UART au niveau du bridge

Nous avons utilisé un convertisseur Sériej-¿USB pour nous connecter au boitier, et observer la séquence de démarrage U-BOOT. La variable d'environnement définissant le temps avant démarrage est fixée à 0 secondes, ce qui nous empêche d'entrer des commandes via cette interface. Cependant, en observant les informations affichées, il est possible de constater que la séquence U-BOOT va récupérer le noyau Linux dans la mémoire NAND Flash, accessible par SPI. La broche CS (Chip Select) doit être mise à l'état bas pour que la mémoire puisse être lue. En forçant cette broche à l'état haut, la mémoire ne répond pas aux commandes U-BOOT, et le chargement du noyau est interrompu. Il devient alors possible de rentrer des commandes au niveau de la séquence U-BOOT.

Une fois la commande accessible, il est possible de modifier la durée d'attente avant démarrage. Elle est définie par une variable d'environnement, dont il est possible de changer la valeur à 3 secondes avec la commande suivante :

```
setenv bootdelay 3
saveenv
```

Après redémarrage, il est maintenant possible d'arrêter la séquence de boot pour rentrer de nouvelles commandes, sans court-circuiter la mémoire NAND. Un script est lancé à chaque démarrage pour vérifier que le mot de passe root présent dans une variable globale est identique à celui actuellement défini, et modifie ce dernier en cas de différence. En modifiant la variable d'environnement, il est donc possible de changer le mot de passe root.

```
setenv security \$1\$AeKNkgji\$haI72VcQ8Yi9K5gtL5T1F0
saveenv
```

Le mot de passe root est maintenant root. A partir de là, il nous a été possible de récupérer l'ensemble des fichiers système, d'activer laccès distant par SSH et telnet, d'installer des paquets grâce au gestionnaire OPKG, etc. Cela permet également de créer ses propres scripts, qui seront ensuite lancés au démarrage. On peut facilement imaginer créer une porte dérobée, pour maintenir l'accès même à distance (rappelons que le Bridge est utile uniquement s'il est connecté à internet, ce qui est donc toujours le cas). La poursuite de cette démarche dépasse le cadre de nos compétences en informatique. Cependant, il est certain qu'un tel accès offre un moyen d'accès durable dans le temps, permettant à quiconque disposant de cet accès de contrôler les lampes associées au Bridge.

# 8 Brouillage du signal émis par la télécommande

La dernière option que l'on avait envisagée depuis le départ était d'empêcher la communication entre l'utilisateur ayant sa télécommande en main et la lampe.

Cette option était envisagée en dernier recours puisqu'elle consiste en une méthode un peu basique et "sale" : on risque de brouiller d'autres utilisateurs et d'autres systèmes. Par ailleurs, brouiller consiste en l'émission à une puissance plus élevée que le signal de la télécommande (ou de la lampe) ce qui implique donc une consommation accrue. On est donc très loin d'une solution idéale.

### Brouillage permanent

Dans un premier temps, nous avons cherché à le faire de manière très basique : on brouille tout le temps de sorte à interrompre l'appairage entre la télécommande et la lampe. Pour cela nous émettons à l'aide du HackRF sur une largeur de bande de 5 MHz centrée sur 2475 MHz. On se rend bien compte que ceci brouille efficacement la communication entre la lampe et la télécommande puisque le voyant de cette dernière clignote en rouge, signe d'un échec d'appairage.

### Brouillage optimisé - déclenchement automatisé

Dans un deuxième temps, nmisé ous avons cherché à optimiser le brouillage en automatisant le démarrage du brouillage uniquement lorsque l'utilisateur veut commander sa lampe. Le cas idéal auquel on pense est :

1. l'utilisateur appuie sur sa télécommande ;
2. le premier paquet du signal est donc émis ;
3. on détecte ce premier paquet et on enclenche le brouillage avant l'envoi du dernier paquet ;
4. la connexion est alors perdue !

Toutefois, ceci est bien sûr loin d'être simple pour diverses raisons. D'une part, nous devrons forcément nous passer du HackRF étant donné que l'on ne parvenait pas à détecter le signal émis par la télécommande ou la lampe avec cet instrument. On utilisera donc le FreakUSB. Le problème principal qui se pose ensuite est le délai entre le moment où le premier paquet est émis et

le moment où le FreakUSB démarre le brouillage sachant qu'entre temps un délai aura eu lieu pour que ce paquet soit détecté par le FreakUSB. Idéalement, il faudrait que le brouillage démarre avant que le dernier paquet émis par la télécommande n'atteigne la lampe, de sorte à faire échouer la communication dès le début. Toutefois, cela requiert une réactivité très grande de la part du FreakUSB. D'après les spécifications du constructeur, les transitions d'état qui font passer le FreakUSB de l'état de réception à l'état de transmission requièrent une durée bien supérieure à la durée d'émission d'une liste de paquets propre à la commande d'allumage ou d'extinction de la lampe.

Dès lors, le brouillage n'aurait lieu qu'après que la lampe ait été efficacement commandée une première fois. Si l'utilisateur veut règler la luminosité ou commander l'opération inverse, il n'y arrivera plus par la suite. Etant donné que l'on cherche à réduire le brouillage par rapport à la première démarche, le brouillage va cesser au bout d'un temps que l'on aura défini donc l'utilisateur pourra ensuite reprendre à nouveau le contrôle le temps d'une commande.

Par ailleurs, il faut rappeler que la télécommande envoie en broadcast des signaux toutes les 15 secondes de sorte à s'appairer avec les lampes environnantes. Quoique l'on fasse (autre que de brouiller en permanence), si l'on n'arrive pas à déclencher le brouillage du signal avant que la fin de l'émission des paquets par la télécommande, notre méthode n'aura aucune conséquence si ce n'est gêner l'utilisateur lorsqu'il voudra exécuter plusieurs commandes successives.

# 9 Conclusion

Bien que nous n'ayons pu aboutir à un piratage à distance en règle, tel qu'on l'avait envisagé au début du projet, nous avons finalement abordé de nombreuses approches assez variées (RF, électronique et un peu d'informatique) qui nous ont permis de mettre en évidence quelques failles exploitables si l'on dispose de plus ou moins de temps. Le projet a finalement énormément évolué au cours des semaines, en fonction des verrous que l'on rencontrait comme par exemple la clé AES qui n'était pas cassable ou alors on s'engageait trop dans le domaine informatique qui ne nous intéressait pas énormément, et ceci nous a finalement permis de nous adapter et d'aborder les différents domaines. La méthodologie nous a ainsi paru très intéressante.

Finalement, la partie la plus aboutie en termes de piratage est le fait d'avoir réussi à devenir root sur le bridge ce qui nous permet une infinité de possibilités telles que la redirection de trafic sensible, l'écoute (notamment grâce à la puce WiFi qui est présente sur le bridge mais non utilisée !), la création d'un bot, etc. Nous n'aurons pas pu développer cet aspect faute de temps mais il serait surement très intéressant d'y consacrer un autre projet étudiant afin de mettre en avant toutes les conséquences de cette faille hardware.

# A Annexe 1 : Liste des canaux Zigbee

| Channel | Frequency (MHz) |
|---|---|
| 0 | 2405 |
| 1 | 2410 |
| 2 | 2415 |
| 3 | 2420 |
| 4 | 2425 |
| 5 | 2430 |
| 6 | 2435 |
| 7 | 2440 |
| 8 | 2445 |
| 9 | 2450 |
| 10 | 2455 |
| 11 | 2460 |
| 12 | 2465 |
| 13 | 2470 |
| 14 | 2475 |
| 15 | 2480 |

FIGURE 9 – Zigbee - Fréquences et canaux associés

# Références


[1] E. Ronen and A. Shamir, "Extended functionality attacks on iot devices : The case of smart lights (invited paper)," 2016 IEEE European Symposium on Security and Privacy.

[2] E. Ronen, C. O'Flynn, and al., "Iot goes nuclear : Creating a zigbee chain reaction," http ://iotworm.eyalro.net/.

[3] "Script gnuradio," http ://www.sbrac.org/files/build-gnuradio.

[4] "Pybombs - python build overlay managed bundle system," https ://github.com/gnuradio/pybombs/.

[5] M. Ossmann, "Great scott gadgets - hackrf," https ://greatscottgadgets.com/hackrf/.

[6] IEEE, "Ieee 802.15.4 wpan$^{TM}$ task group 4 (tg4)," http ://www.ieee802.org/15/pub/TG4.html.

[7] "Freakusb," http ://www.freaklabsstore.com/index.php ?main$_page$ = $product_infocPath = 22products_id = 215$.

[8] C. O'Flynn, "Getting root on philips hue bridge 2.0," http ://colinoflynn.com/2016/07/getting-root-on-philips-hue-bridge-2-0/.