Institut Universitaire de Technologie,
Aix-Marseille Université

RAPPORT DE STAGE
Diplôme Universitaire de Technologie
Spécialité Réseaux et Télécommunications

Étude et documentation de systèmes pour la recherche de vulnérabilités

Alexandre COL

EDF DIPDE

Responsable entreprise : Jean-Hugues ZORIO

Responsable académique : Éric SOCCORSI

2019

Table des matières

1 Introduction

Du 8 avril au 14 juin 2019 j'ai effectué mon stage à EDF DIPDE (Division de l'Ingénierie du Parc, de la Déconstruction et de l'Environnement), dans la cellule C3D (Centre de Compétences en Cyber-sécurité de la DPNT - Direction du Parc Nucléaire et Thermique).

Mon sujet de stage a été adapté par rapport à celui indiqué sur la fiche mission pour des raisons de sécurité informatiques (informations confidentielles), ce que je précise dans la partie 'Missions et Objectifs'.

Pendant cette période, j'ai principalement préparé et documenté des systèmes qui seront utilisés pour des audits de cyber-sécurité.

J'ai eu 2 missions principales : le cassage de mots de passe, et l'étude de transmissions en fréquence radio (écoute, rejeu, modification de signal, …).

2 Présentation de l'entreprise

2.1 EDF

EDF est actuellement une société anonyme intégrée comprenant principalement les branches en propre ou au travers de filiales : production (EDF DPNT, HYDRO, E.N.), distribution (RTE, ENEDIS) et commercialisation (EDF Commerce) d'électricité. Les parties constructions neuves de production (EDF DIPNN) et recherche et développement (EDF R.&D.) sont aussi des composantes importantes d'EDF.

La branche production-ingénierie dans laquelle j'ai réalisé mon stage fait partie de la branche DPNT à laquelle appartient la DIPDE.

La cellule « C3D », entité de la DPNT, est hébergée à la DIPDE Marseille.

2.1.1 La DIPDE

2.1.1.1 Organisation

La DIPDE (environ 1900 personnes) est plus particulièrement en charge de toute la maintenance et l'entretien de l'ensemble de la partie nucléaire des centrales nucléaires françaises en fonctionnement, dit le « Parc Nucléaire », soit 58 tranches nucléaires 900, 1300 et 1450 MWe, réparties sur 19 sites sur toute la France (les CNPE – Centre Nucléaire de Production d'Eléctricité).

La DIPDE est basée à Marseille (environ 1200 personnes) pour les services supports (administratifs, RH, logistique), les études de l'îlot nucléaire (DEIN), les approvisionnements/achats de prestations et travaux (PCO – Performance COntractuelle / Contrôle de Gestion Opérationnel) et les responsables Projets et travaux (DPP – Département Performance des Projets, et DR – Département Réalisation).

Une antenne de la DIPDE est sur Lyon (environ 180 personnes), pour les études d'environnement et de déconstruction (DEED), et des agents relais de la DIPDE (environ 520 personnes) sont sur chaque site au travers des Equipes Communes (DIPDE + DPN-CNPE) afin de suivre les travaux de maintenance.

Des organigrammes (EDF et DIPDE) sont joints en Annexe a et Annexe b.

2.1.1.2 Activités

La DIPDE étudie et applique des modifications pour optimiser les performances des centrales et s'assurer de leur sûreté.
Depuis 2014, un programme industriel, le 'Grand Carénage', vise la prolongation de la durée de fonctionnement des réacteurs pour permettre au Parc d'atteindre ses objectifs de production en toute sûreté. 80% des activités de la DIPDE sont dédiés à ce programme, principalement les projets de déconstruction des centrales nucléaires définitivement mises à l'arrêt (9 réacteurs) ainsi que la construction des nouvelles installations nécessaires à la gestion des déchets.

2.1.2 C3D

La cellule « C3D » travaille pour l'ensemble du Parc nucléaire (DPNT) : ilot nucléaire (cœur et circuit primaire), ilot conventionnel (partie de la centrale nucléaire non directement liée au réacteur (circuits d'eau secondaires et tertiaires, transformation de la vapeur en électricité par la turbine, etc.)), parc existant et nouveaux paliers, activités de déconstruction, outils d'ingénierie et systèmes informatisés de la protection de site. Elle travaille aussi en collaboration avec la cellule C2I (Contrôle Commande et Informatique Industrielle) qui dispose de son côté de simulateurs numériques de centrales nucléaires. Ces simulateurs permettant de tester la conformité et sécurité des programmes avant diffusion sur les sites en exploitation.

Photo 1 - Salle de commande de la centrale de Chooz (Ardennes). ©Alexandre Sargos/EDF Mediathèque

2.1.2.1 Doctrine

La cellule C3D assure un suivi des doctrines et référentiels techniques opérationnels en participant aux instances internationales de sécurité informatique du nucléaire et à l'élaboration des normes ou guides qu'elles produisent (AIEA, IEC…), puis en décline les principes dans la doctrine et les référentiels d'EDF. Elle participe aussi aux échanges en matière de sécurité informatique du nucléaire avec les autorités nationales et émet un avis au projet et en COSIN (Comité Opérationnel de la Sécurité Informatique du Nucléaire).

2.1.2.2 Projets

Elle est sollicitée sur divers projets pour y apporter un appui et une expertise, notamment dans la définition des actions relatives à la sécurité informatique à mener au cours du projet, l'attribution des systèmes dans un degré de sécurité, l'expression des exigences de sécurité dans les spécifications internes ou à destination des fournisseurs, l'étude d'architecture en sécurité, la recherche de vulnérabilités dans les logiciels et équipements, le conseil quant à la mise en place de fonction de sécurité ou la correction de vulnérabilités identifiées et la conduite du volet de sécurité des phases de recette. Elle formule un avis indépendant sur la suffisance des dispositions prises en matière de sécurité informatique.

2.1.2.3 Vulnérabilités

Elle maintient l'infrastructure sécurisée de gestion des vulnérabilités de l'informatique du nucléaire et participe à l'analyse technique des avis de sécurité.

2.1.2.4 Cyber-surveillance et incidents

C3D définit les scénarios d'alerte à couvrir et les événements à remonter, conçoit les outils de surveillance des événements de cyber-sécurité afin de détecter le plus en amont possible tous les incidents de sécurité informatique, appuie l'exploitant pour l'analyse des alertes cyber-sécurité et définit les procédures élémentaires de traitement d'incident en cyber-sécurité.

2.1.2.5 Animation du domaine technique

Elle assure la gestion du Plan de Développement des Compétences dans le sous-domaine de la cyber-sécurité et contribue au bilan de santé du domaine sur ce thème. Elle pilote aussi les projets de la R&D en matière de sécurité informatique du nucléaire.

3 Contextualisation

3.1 Missions et objectif de stage

Ma mission initiale était : 'aide la mise en place d'une infrastructure segmentée en différents réseaux métiers comprenant un SAN, plusieurs DMZ, plusieurs coupe-feux…'. Après des échanges par mail avec un membre de la cellule, une mission supplémentaire m'a été confiée : la mise en place d'un cluster* de machines pour le craquage de mot de passe.
Mon responsable en entreprise, Jean-Hugues ZORIO n'étant pas disponible pendant deux semaines au début de mon stage, c'est la personne qui m'a confié la mission de craquage de mots de passe qui m'a supervisé (pendant cette période).
Au milieu de la durée du stage, ma mission initiale liée au réseau a été remplacée pour des raisons de confidentialités par l'étude d'outils d'analyse des communications radio.

Le principal objectif de mes missions était d'apprendre à manipuler des logiciels qui seront utilisés par la cellule C3D pour des audits de sécurité, et d'en constituer de la documentation d'utilisation. Dans un cadre plus général, elles m'ont permis de voir et de comprendre comment casser et exploiter une sécurité logique et physique.

3.2 Contexte technique

3.2.1 Craquage de mots de passe

3.2.1.1 Stockage des mots de passe

Les entreprises aussi grandes qu'EDF possèdent un parc informatique très complexe et très grand, avec un grand nombre d'utilisateurs.
À chaque groupe d'utilisateurs (administrateurs informatique, privilégiés ou utilisateurs simples, non privilégiés) est attribué un mot de passe ainsi qu'une politique de mot de passe. Une politique de mot de passe définit la taille, les caractères autorisés et/ou obligatoires dans les mots de passe. Des équipes de sécurité, tel C3D, peuvent être sollicitées afin de tester la robustesse des mots de passe et le respect des politiques.
En informatique, il est inacceptable de stocker des mots de passe 'en clair', c'est-à-dire lisibles directement. On applique sur le mot de passe une fonction dite de 'hachage' qui va renvoyer une chaîne de caractères, appelée 'hash' ou 'empreinte', mais unique pour chaque mot. Ces fonctions sont irréversibles, et très nombreuses. Les plus connues sont le LM, NTLM (utilisé par Windows), MD5, SHA-1, SHA-256, SHA-512. Elles possèdent toutes des caractéristiques différentes, et donc renvoient des hash différents.
Par exemple, en MD5, le hash du mot 'test' est : `098f6bcd4621d373cade4e832627b4f6`
En SHA-1, c'est : `a94a8fe5ccb19ba61c4c0873d391e987982fbbd3`
Et en NTLM : `0CB6948805F797BF2A82807973B89537`

3.2.1.2 Concept de brute-force

Sachant que ces fonctions sont irréversibles, on peut se demander comment il est possible de retrouver un mot de passe. Il existe plusieurs méthodes pour retrouver ce que contient un hash : l'attaque par force brute, l'attaque par dictionnaire, l'utilisation de tables arc-en-ciel (communément appelées *Rainbow tables*, cf. Documentation Rainbow tables dans les annexes)…
Le premier point commun entre ces méthodes est en général l'utilisation du principe de collision : en force brute, le principe universel est de tester toutes les possibilités jusqu'à tomber sur la bonne combinaison. Pour des hash, le but est de comparer un hash dit de référence (de notre mot de passe) avec plusieurs hash générés à partir de mots aléatoires. Ces mots aléatoires sont hachés en utilisant la même fonction de hachage que le hash de référence. S'ils correspondent, il y a 'collision' et le mot de passe est le mot aléatoire qui vient d'être haché.

Le second point commun est le besoin de puissance de calcul. Plus on a une grande puissance disponible, plus le temps de cassage pour le brute-force sera réduit, et le temps de génération des tables arc-en-ciel sera réduit.

Mais il est rare de craquer des mots de passe sans un minimum d'optimisation. Par exemple pour les attaques par dictionnaires il est plus judicieux de prendre des listes de mots de passe les plus utilisés qu'une liste de mots générés aléatoirement. Lors de mes travaux sur les mots de passe j'ai utilisé l'attaque par force brute et les tables arc-en-ciel.

3.2.2 Analyse de radio fréquence

De nos jours, on a tendance à commander à distance, par radio fréquences, la plupart de nos appareils : portails, lumières, portières de voiture, alarmes…
EDF a lancé depuis 2014 un projet de rénovation du parc nucléaire, le projet 'Grand Carénage'. De nombreux chantiers de construction et rénovation ont été lancés depuis ces années. Ces chantiers doivent être approuvés en termes de sécurité, donc la cellule C3D peut y être sollicitée, comme récemment, début Juin 2019. Pour préparer cet audit, j'ai transmis mes connaissances acquises au cours de ma découverte des logiciels à Jean-Hugues ZORIO et un autre membre de C3D. Le principal objectif a été d'essayer de faire du rejeu de signal.
Le principe du rejeu de signal est la technique la plus commune, la plus médiatisée (dans les films ou séries) pour tester la sécurité ou le manque de sécurité d'un système commandé. Le rejeu décrit l'action de renvoyer tel quel un signal capturé. Si le rejeu fonctionne sur un système, alors c'est que ce dernier possède un codage des données 'statique', qui sert soit simplement à chiffrer le signal, soit a été décidé par le constructeur.
Cependant, les systèmes modernes sont dotés de codes plus complexes, à ce qu'il paraît…
Le premier code 'complexe' qui empêche le rejeu est le code tournant. Une valeur ou plusieurs valeurs dans le signal sont changées à chaque utilisation. C'est là qu'il devient nécessaire d'effectuer de la rétro-ingénierie*. Dans mon cas, il est question de savoir comment sont agencées et transmises les données d'un signal.
Après avoir trouvé la ou les valeurs qui changent, il est courant d'essayer de brute forcer le signal, c'est-à-dire d'envoyer toutes les combinaisons possibles les unes à la suite des autres. Je n'ai pas eu l'occasion de tester ça pendant ma mission.

Le piratage de radio fréquence, est devenu 'à la mode' (grâce au « car hacking », le fait de pouvoir ouvrir et démarrer une voiture informatiquement avec un émetteur et un pc) et il est donc nécessaire de s'assurer de la sécurité du système.

4 Travaux réalisés

4.1 Craquage de mots de passe

La cellule cyber-sécurité dans laquelle j'ai effectué mon stage a choisi d'exploiter la puissance de calcul de carte graphique.
Le logiciel répondant le mieux à cette demande est Hashcat. Il existe un logiciel plus connu, plus ancien nommé John The Ripper, mais sa dernière version (1.8 au moment où j'ai débuté mes travaux) datait de 2015, et ne permet pas d'exploiter le GPU*.
Hashcat se montre bien supérieur à John, supportant des technologies modernes de calcul exploitant le GPU, plus de 200 types de hash, plusieurs types 'd'attaques'.
La dernière version sortie de John The Ripper date du 14 mai 2019, mais nous ne nous sommes pas plus penchés dessus, l'installation de la 'station de craquage' étant déjà effectuée.
Le principal défaut des versions de Hashcat antérieures à la dernière sont qu'elles ne stockent pas les mots testés (générés et hachés), ce qui implique que le logiciel re-calcule tous les mots déjà calculés lors d'une session précédente. Dans la dernière version, le logiciel possède une fonctionnalité nommée « Brain » qui permet de stocker les mots déjà haché pour soulager les calculs.

Le second logiciel que j'ai utilisé est Rainbowcrack. Il permet de générer et d'utiliser des tables arc-en-ciel pour retrouver un mot de passe haché.
J'ai mis à disposition en annexe la documentation que j'ai réalisée sur ce logiciel, ainsi que des explications sur le fonctionnement des tables arc-en-ciel (création, fonctionnement de la recherche) dont les recherches m'ont beaucoup intéressé. L'origine de ces tables est l'amélioration des compromis temps-mémoire proposés par Martin Hellman vers 1980.

4.1.1 Force brute avec Hashcat

L'idée de base de C3D était de créer un cluster de machines pour craquer les mots de passe. Je disposais donc d'une machine, et un alternant d'une autre possédant les mêmes caractéristiques (carte graphique, processeur, carte mère, mémoire RAM (Random Access Memory, mémoire vive permettant à un appareil de stocker temporairement des informations à traiter)).

Pour éviter d'abîmer le système avec des tests, et d'être sûr de pouvoir les effectuer tranquillement, j'ai dû passer par une machine virtuelle*. Une fois sur ma VM (Virtual Machine), j'y ai installé Hashcat pour commencer les essais.
Hashcat ne prend en charge que les appareils possédant l'API* OpenCL*. Pour vérifier la compatibilité des composants de la machine, j'ai entré la commande `hashcat --opencl-info` dans un terminal.

```
col@col-VirtualBox:~$ hashcat --opencl-info
hashcat (v4.0.1) starting...

No devices found/left.
```

D’après le résultat, le logiciel n’est pas parvenu à détecter une carte graphique ou un processeur. Cependant, en vérifiant directement sur la machine physique, Hashcat a détecté 2 composants compatibles.

```
hashcat (v4.0.1) starting...

OpenCL Info:

Platform ID #1
  Vendor  : xxxx
  Name    : xxxx
  Version : xxxx

  Device ID #1
    Type           : GPU
    Vendor ID      : xxxx
    Vendor         : xxxx
    Name           : xxxx
    Version        : xxxx
    Processor(s)   : xxxx
    Clock          : xxxx
    Memory         : xxxx
    OpenCL Version : xxxx
    Driver Version : xxxx

Platform ID #2
  Vendor  : xxxx
  Name    : xxxx
  Version : xxxx

  Device ID #2
    Type           : CPU
    Vendor ID      : xxxx
    Vendor         : xxxx
    Name           : xxxx
    Version        : xxxx
    Processor(s)   : xxxx
    Clock          : xxxx
    Memory         : xxxx
    OpenCL Version : xxxx
    Driver Version : xxxx
```

Le problème vient de la machine virtuelle qui n’utilise pas complètement la carte graphique. J’ai cherché comment il était possible de pallier à ce problème, mais les résultats n’étaient pas concluant, ou trop instables (mauvaise utilisation de la carte, trop peu de puissance).

Je suis donc passé directement sur la machine physique.

J’ai continué de faire des recherches sur la « clusterisation », les logiciels, les méthodes, puis au cours d’une réunion, nous avons décidé d’abandonner l’idée du cluster de machines, à cause d’une mise en place longue et pas nécessaire. Avec l’équipe travaillant sur ce projet, nous avons décidé de rassembler les deux cartes graphiques dans le même boîtier, pour au final posséder un seul ordinateur, et une meilleure exploitation des GPU.

J'ai effectué une série de tests de craquage de mots de passe en brute force avec Hashcat pour apprendre à manipuler le logiciel.
Par exemple pour le mot « test », haché avec du MD5, j'ai saisi la commande suivante :

```
hashcat -a 3 -m 0 -o found.txt 098f6bcd4621d373cade4e832627b4f6
```

L'option –a précise le mode d'attaque utilisé, ici 3 pour le brute force, l'option –m précise la fonction de hachage utilisée, le 0 correspond au MD5, l'option –o précise le fichier dans lequel je récupère le mot de passe craqué, et la suite de caractères correspond au hash MD5 du mot.
Cependant, Hashcat utilise automatiquement un 'masque' par défaut si il n'est pas précisé. Pour le logiciel, un masque sculpte le mot de passe. Il permet d'affecter à chacun des caractères une plage de valeurs (minuscules, majuscules, chiffres, caractères spéciaux).
Le modèle prédéfinit de Hashcat est très gênant car il ne balaye pas tous les mots de passe possibles : Majuscule seulement au premier caractère, caractères spéciaux présent à partir du 7e caractère…

Je me suis donc penché sur l'utilisation des masques.

La documentation que j'ai réalisée est disponible en annexe (Documentation Hashcat), il est conseillé d'y jeter un œil pour clarifier les principes de masques qui constituent un élément important d'optimisation.

Hashcat est un logiciel très complet qui renvoie beaucoup de détails sur le cassage en cours :

```
Session..........: hashcat
Status...........: Running
Hash.Type........: sha512crypt $6$, SHA512 (Unix)
Hash.Target......:
$6$0zVNEu2cke1$UE3dN2zJvjAdaHdWzI3OH9sd7XrpliTbs5Si...9hJpD1
Time.Started.....: Tue Jun 11 15:36:37 2019 (50 secs)
Time.Estimated...: Tue Jun 11 20:15:25 2019 (4 hours, 37 mins)
Guess.Mask.......: ?1?1?1?1 [4]
Guess.Charset....: -1 ?l?d?u, -2 Undefined, -3 Undefined, -4
Undefined
Guess.Queue......: 1/4 (25.00%)
Speed.Dev.#1.....:      xxxx H/s (xxxxms)
Recovered........: 0/1 (0.00%) Digests, 0/1 (0.00%) Salts
Progress.........: 32768/14776336 (0.22%)
Rejected.........: 0/32768 (0.00%)
Restore.Point....: 0/238328 (0.00%)
Candidates.#1....: mari -> mErc
HWMon.Dev.#1.....: Temp: 52c Fan: 25% Util: 99% Core: xxxx Mem:
xxxx Bus:xxxx
```

Ces informations sont envoyées à chaque « keyspace »*. Par exemple pour cet espace de possibilités, il y a 62^4 = 14 776 336 combinaisons. Il y a autant de keyspace que de caractères spécifiés dans le masque.
Des informations sur le temps estimé et écoulé sont données mais elles correspondent seulement à la keyspace en cours. Cela signifie qu'il faut attendre qu'Hashcat calcule toutes les possibilités de l'ensemble actuel avant de connaître le temps estimé du prochain, ce qui est très embêtant si on souhaite seulement estimer le temps de cassage total. Quand on arrive sur des essais de plus de 10 caractères, avec une fonction de hachage complexe, cela peut prendre plusieurs jours (suivant la puissance disponible).
J'ai remarqué cet élément au cours de mes tests, et j'en ai discuté avec mon équipe pour savoir s'il était possible d'obtenir le temps total final (donc le temps maximum que prendrait le cassage).

N'ayant pas trouvé de réponses auprès des développeurs du logiciel ni sur des forums, j'ai pris l'initiative de coder un script* bash* (cf. partie Script dans les annexes) qui renvoie le temps total estimé.
Pendant le traitement, le logiciel offre la possibilité d'effectuer plusieurs actions :

```
[s]tatus [p]ause [r]esume [b]ypass [c]checkpoint [q]uit =>
```

'Status' affiche l'état actuel du calcul (les informations montrées plus haut), 'pause' arrête momentanément le calcul jusqu'à la saisie de la lettre 'r', 'bypass' permet de sauter le calcul de la keyspace actuelle et d'en afficher les informations juste après, 'checkpoint' sert à créer un point de restauration pour reprendre un autre moment la session de calcul, et 'quit' permet d'arrêter totalement le calcul.

Pour mon script, je me suis servi de la propriété de l'action bypass. Elle permet de récupérer les informations de temps de calcul, tout en passant ces temps-là.
Il est juste de se dire que j'aurai simplement pu utiliser la vitesse de calcul en hash/s et le nombre de possibilité pour calculer le temps total, mais la vitesse de calcul change pour chaque fonction de hachage, et doit se stabiliser pendant les 10 premières secondes du calcul d'une keyspace.

C'est pour cela que mon script fait tourner en arrière-plan une commande Hashcat donnée en argument, puis toutes les 30 secondes il lui envoie des demandes de bypass, jusqu'à la dernière qui arrête à chaque fois le logiciel.
Pour exécuter la commande en arrière-plan j'ai utilisé le multiplexeur de terminaux 'screen'. Il permet d'exécuter plusieurs terminaux dans une console. L'avantage de cet outil est qu'il est possible de rediriger le contenu de la fenêtre créée en arrière-plan dans un fichier. J'ai exploité cette propriété pour envoyer la sortie de la commande Hashcat dans un fichier et en récupérer à l'aide des outils de recherches de chaînes de caractères de Linux ('grep', 'awk', 'tr') les données de la partie `Time.Estimated`.
Mon script utilise ces données pour incrémenter des variables qui seront finalement additionnées pour obtenir le temps maximal estimé (exemple de sortie ci-dessous) :

```
Le temps maximal total de calcul est de :
97 année(s) 92 jour(s) 3 heure(s) 42 minute(s) 0 seconde(s)
```

Il est important de noter que les temps indiqués par Hashcat sont les temps **maximaux**. Cela signifie que le cassage peut prendre moins de temps.

Même avec des optimisations de recherches, l'attaque par force brute prend du temps. J'ai continué mes travaux sur le cassage de mots de passe en m'intéressant aux tables arc-en-ciel.

L'avantage de ces tables est qu'elles permettent de trouver très rapidement un mot de passe, et elles restent un des moyens les plus rapides et efficaces pour casser un mot de passe. Cependant elles nécessitent un grand espace de stockage, et la taille des mots hachés à l'intérieur est limitée car la génération de ces tables prendrait trop de temps.

4.2 Analyse de radio fréquences

Pour cette seconde mission, la cellule C3D m'a confié un boîtier SDR, Software-Defined Radio, qui est un récepteur et émetteur utilisant principalement des logiciels et limite un maximum l'utilisation matériel.
C'est un outil très pratique car il permet de faire à peu près ce qu'on veut avec et est contrôlable par commandes.

Figure 1 - Un HackRF One

Pour gagner du temps sur une analyse de signal, il est conseillé de se renseigner sur l'objet qu'on étudie, notamment sur sa fréquence d'émission. Pour mes premiers essais, j'ai utilisé un boîtier de commande de portail FAAC. Après de courtes recherches, j'ai trouvé sa fréquence d'utilisation qui est autour de 868.35 Mhz (Annexe c - Boîtier de commande FAAC).

Pour l'utiliser, il est nécessaire d'installer les paquets `hackrf` et `libhackrf-dev`, le brancher à un ordinateur avec un câble USB – Micro USB et taper `sudo hackrf_info` dans une invite de commande pour vérifier le matériel (Figure 2).

```
hackrf_info version: unknown
libhackrf version: unknown (0.5)
Found HackRF
Index: 0
Serial number: 0000000000000000453c64c82287258f
Board ID Number: 2 (HackRF One)
Firmware Version: 2015.07.2 (API:1.00)
Part ID Number: 0xa000cb3c 0x008a4769
```

Figure 2 - Résultat de la commande hackrf_info

4.2.1 Détection avec Gqrx

Gqrx est un logiciel de visualisation et d'écoute de fréquences. Il est capable de renvoyer sous forme sonore les signaux reçus (oui, on peut écouter la radio avec, j'ai testé).

La première fenêtre qui apparaît est la sélection du SDR (Figure 3).

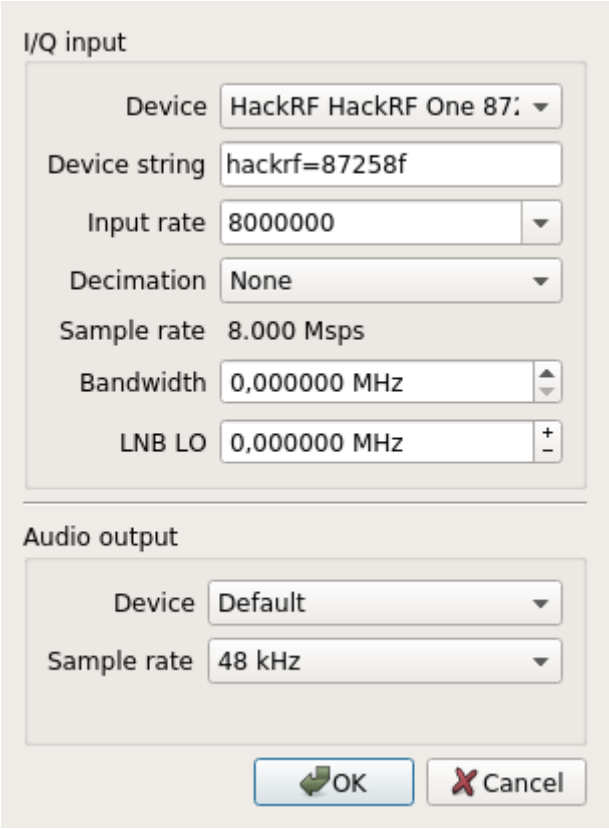


Figure 3 - Sélection du récepteur-émetteur, ici le HackRF

Après avoir choisi mon appareil de réception, le HackRF, j'ai réglé quelques paramètres (Figure 4).
Dans la rubrique *Input Controls*, j'ai coché la case *DC Remove* pour enlever le pic central. Ce pic est dû au convertisseur et n'est pas important, il peut être gênant si la fréquence de l'appareil émetteur est vraiment proche.
J'ai ensuite réglé la fréquence (*Receiver Options* > *Frequency*) sur 868.35 Mhz (la fréquence trouvée), et le mode sur *Demod Off* (car on ne connaît pas encore la modulation utilisée).

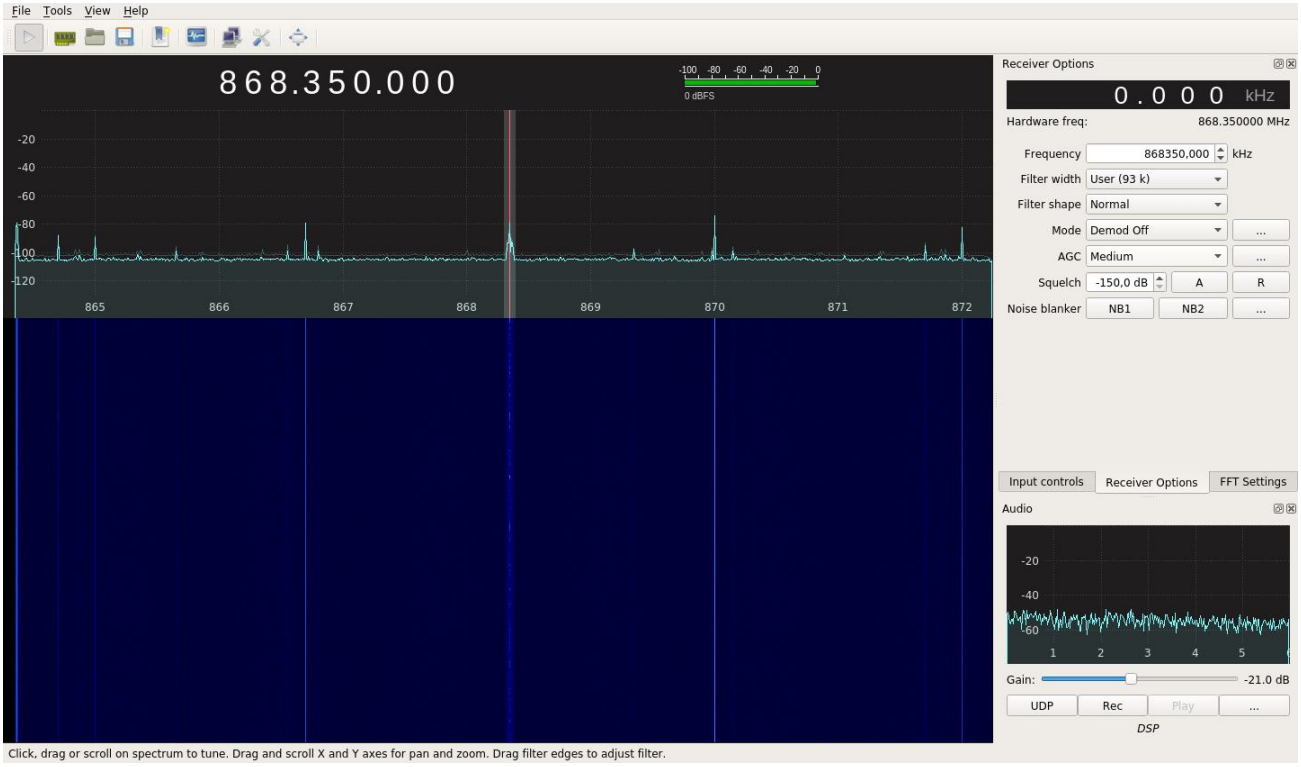


Figure 4 - Paramétrage du logiciel

Si on veut être vraiment précis, il faut régler la fréquence sur une valeur supérieure ou inférieure à la fréquence que l'on veut analyser car il y aura tout de même des pics dû au CAN - convertisseur analogique-numérique - au centre (donc sur la fréquence réglée) du spectre.
La valeur de *Filter width* a été choisie automatiquement par le logiciel en fonction des paramètres du SDR sélectionné.

Il faut savoir qu'on travaille seulement avec un signal numérique. Un ordinateur n'est pas en mesure de traiter une infinité de points (signal analogique). Gqrx échantillonne le signal reçu, c'est-à-dire qu'il sélectionne des points du signal à des intervalles de temps réguliers.
Dans *FFT Settings,* il faut sélectionner une grande valeur pour l'option *FFT Size*. FFT signifie Fast Fourier Transform ou transformation de Fourier rapide. C'est un algorithme de calcul de la transformée de Fourier discrète massivement utilisé en télécommunication. Une transformée de Fourier est une représentation en fréquence d'un signal. La transformée de Fourier discrète est la représentation spectrale (en fréquences) d'un signal discret (échantillonné).
Cela va augmenter le nombre de points sélectionnés pendant l'échantillonnage, donc permettre de se rapprocher du signal de base, et donc d'avoir des valeurs plus précises/correctes. Plus la valeur est grande, plus le CPU aura besoin de ressources.

Sur la figure ci-dessus, la partie supérieure est la transformée de Fourier discrète du signal captée par l'antenne du HackRF. La partie inférieure est appelée « cascade ». Elle représente l'évolution de la puissance des fréquences (en couleurs, les plus foncées représentant souvent du bruit, puissance faible) en fonction du temps (de haut en bas).

Lorsque j'ai appuyé sur le boîtier, j'ai observé l'apparition d'un pic légèrement en dessous de la fréquence de référence (Figure 5).

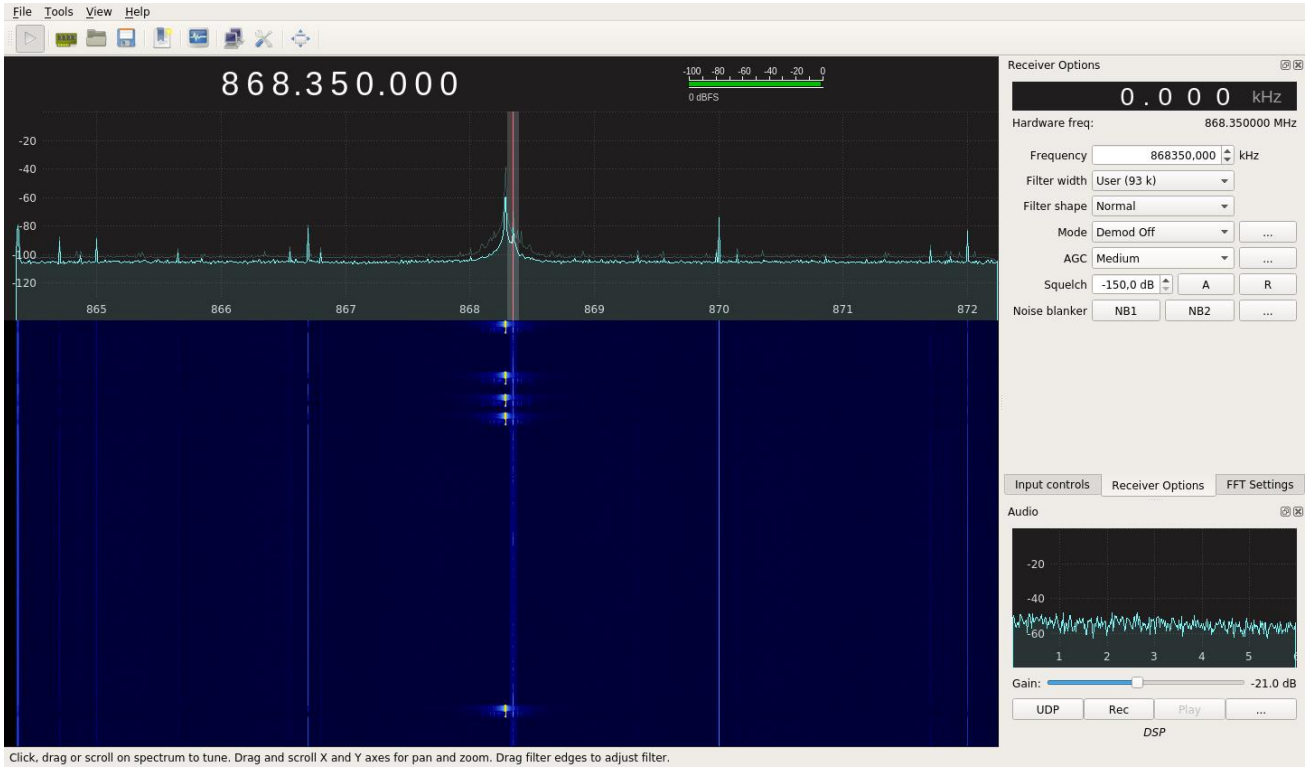


Figure 5 - Réception du signal du boîtier

Sur la capture, dans l'affichage en cascade, on peut voir distinctement qu'un signal a été reçu. Il y a plusieurs impulsions visibles car j'ai appuyé plusieurs fois sur le boîtier pour des essais. Après avoir prélevé la fréquence du boîtier, je suis passé sur URH.

4.2.2 Analyse avec Universal Radio Hacker (URH)

URH est un logiciel très complet qui permet d'effectuer de la rétro-ingénierie sur des signaux.

J'ai téléchargé les fichiers d'installation proposés par l'auteur sur Github (https://github.com/jopohl/urh). Il faut installer un grand nombre de paquets python pour pouvoir installer URH. La commande ci-après, à taper dans un terminal, permet d'installer tout ce qu'il nous faut d'un coup.

```
sudo apt install python3-numpy python3-psutil python3-zmq python3-pyqt5 g++ libpython3-dev python3-pip cython3
```

Pour installer URH, il faut se placer dans le dossier contenant les fichiers d'installation, et le fichier 'setup.py', l'installeur, et taper la commande `sudo python3 setup.py install`.
Le logiciel doit être lancé avec les droits administrateur (`sudo`).

Pour effectuer les analyses, il est nécessaire d'avoir un fichier contenant des données d'un signal. Il est possible d'ouvrir directement un fichier, ou d'enregistrer directement un signal avec le logiciel et le récepteur.

On arrive sur une fenêtre dans laquelle il faut choisir l'appareil de capture et ajuster la fréquence dans la rubrique *Frequency (Hz)*. J'ai indiqué qu'on utilise le HackRF, puis j'ai rentré la fréquence trouvée avec Gqrx (868.287 Mhz). Il faut aussi cocher la case *DC correction* pour éviter le pic central (vu plus haut avec Gqrx).
On lance l'enregistrement (*Start*) puis on active notre émetteur. Ici j'ai appuyé 3 fois sur mon boîtier, et j'ai obtenu le signal Figure 6.

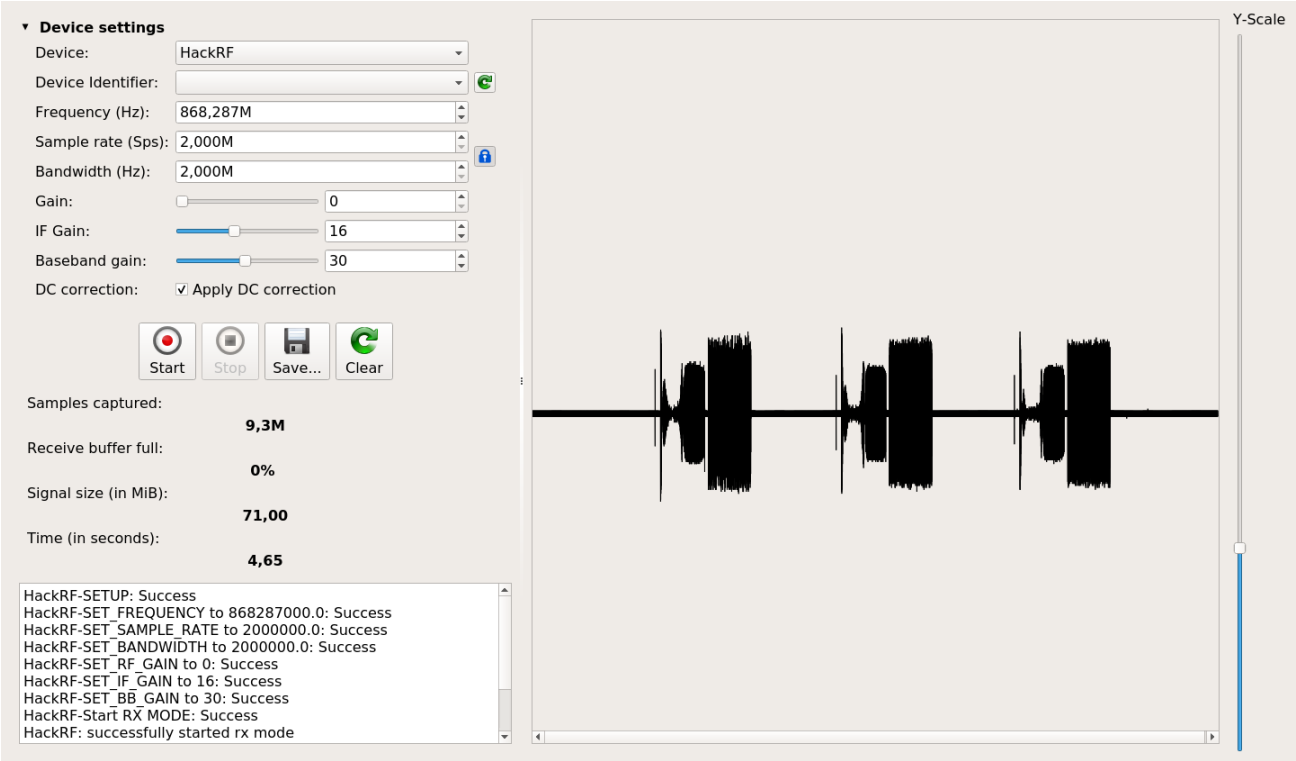


Figure 6 - Capture du signal de 3 appuis du boîtier

J'ai enregistré mon signal, puis je suis retourné sur l'écran initial d'URH. Pour lancer les analyses, il faut ouvrir le fichier qu'on vient d'enregistrer. Il est possible (et obligatoire pour effectuer les analyses) de zoomer sur le signal pour une vision plus clair.
Dans mon exemple, l'analyse se fera sur la deuxième partie du signal (le dernier gros 'paquet' de sinusoïdes) car c'est là que les variations sont le plus présentes et traduisent la présence d'informations.
J'ai dû effectuer le traitement en 2 étapes :
• déterminer la modulation utilisée
• trouver combien d'échantillons (sample en anglais) composent un bit (l'option *Bit Length*, Figure 7).
Ces informations permettront ensuite d'arranger de manière cohérente les données reçues, de façon à ce qu'elles correspondent aux variations du signal.

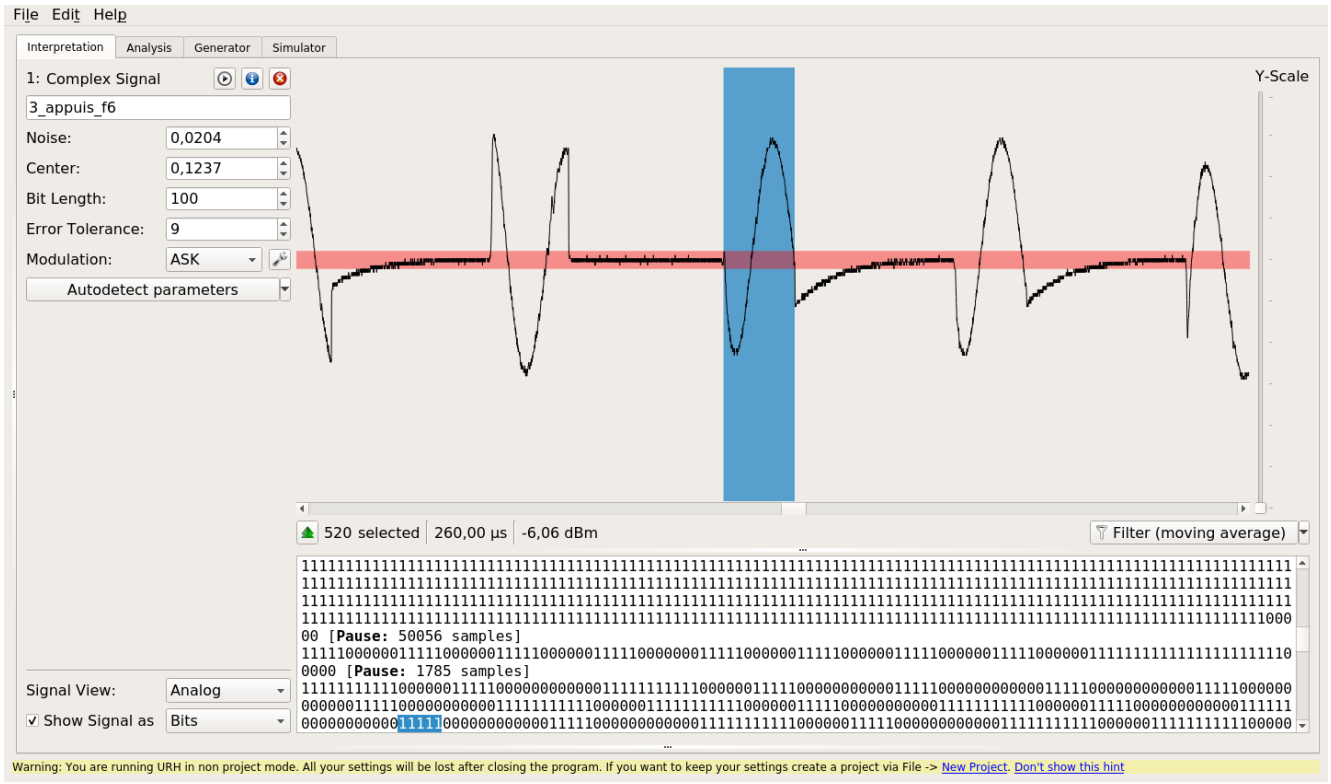


Figure 7 - Zoom et sélection sur la 2ième partie du signal

Sur la capture ci-dessus j'ai conclu que la modulation était une modulation d'amplitude (Amplitude Shift Keying, le « ASK » de l'option *Modulation* à gauche), c'est-à-dire que l'amplitude du signal va varier en fonction des bits à transmettre (cf. Annexe d - Exemple de modulation d'amplitude)

On remarque très facilement sur la Figure 7 que les bits '1' sont représentés par une amplitude non nulle, et inversement pour les bits '0'. J'ai donc sélectionné le plus court symbole possible dans mon signal, qui est la représentation d'un bit, et j'ai reporté le nombre d'échantillons indiqués (*520 selected*) dans *Bit Length*.

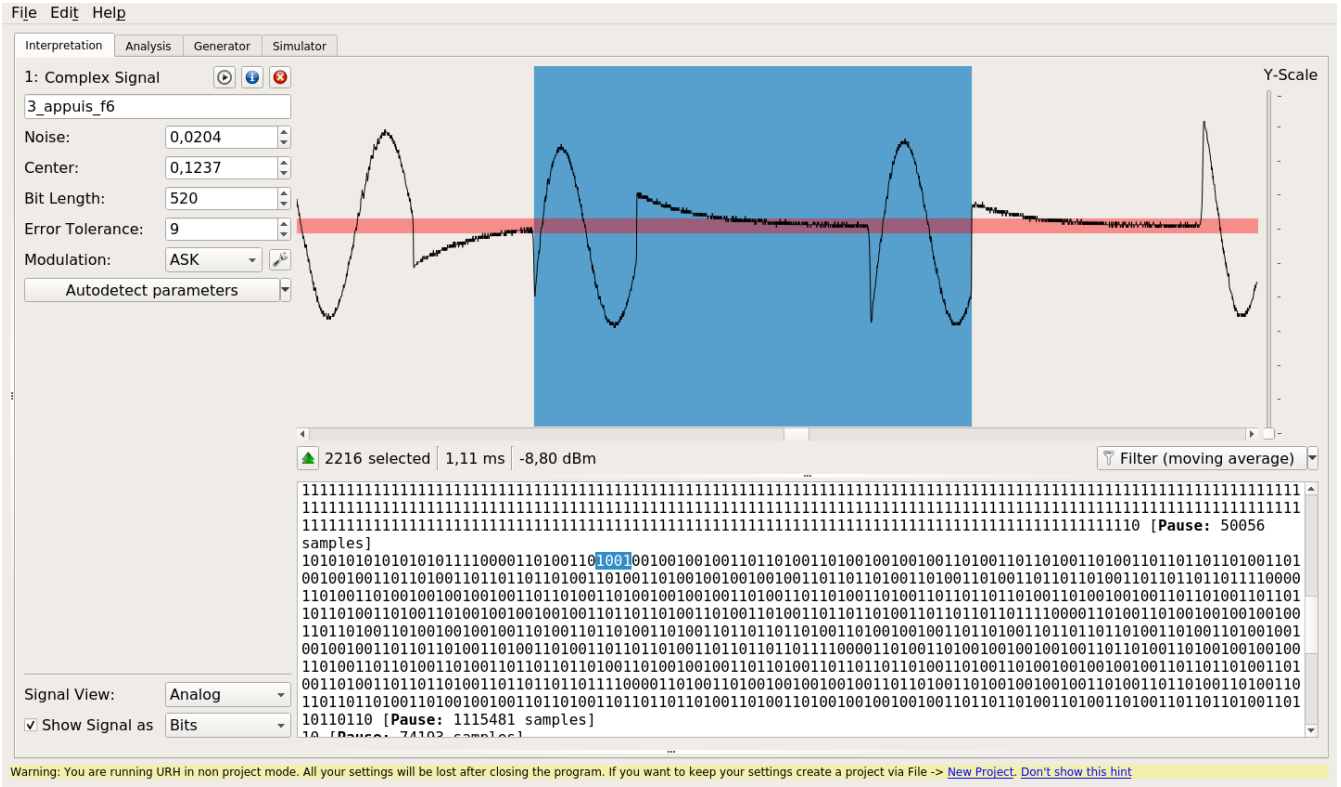


Figure 8 - Vérification des valeurs

Pour vérifier que la valeur soit correcte, j'ai sélectionné une suite de bits sur la trame sous le signal, et on peut voir sur la Figure 8 qu'ils correspondent à la forme du signal : la sélection commence par un bit '1', et l'amplitude du signal est non nulle. Ce bit est suivi par 2 bits '0', traduits par une amplitude nulle sur le signal. Le dernier bit de la sélection est '1', et on retrouve le symbole correspondant.

Je suis ensuite retourné dans la partie *Analysis*. Les transmissions radio à courtes distances utilisent le plus souvent le codage Manchester mais dans notre cas le boîtier de commande utilise un codage propriétaire SLH LR (brevet FAAC). C'est un code qui varie à chaque utilisation (Self Learning Hopping code, code tournant) et qui est reconnu seulement si le signal des émetteurs a été codé par le récepteur.

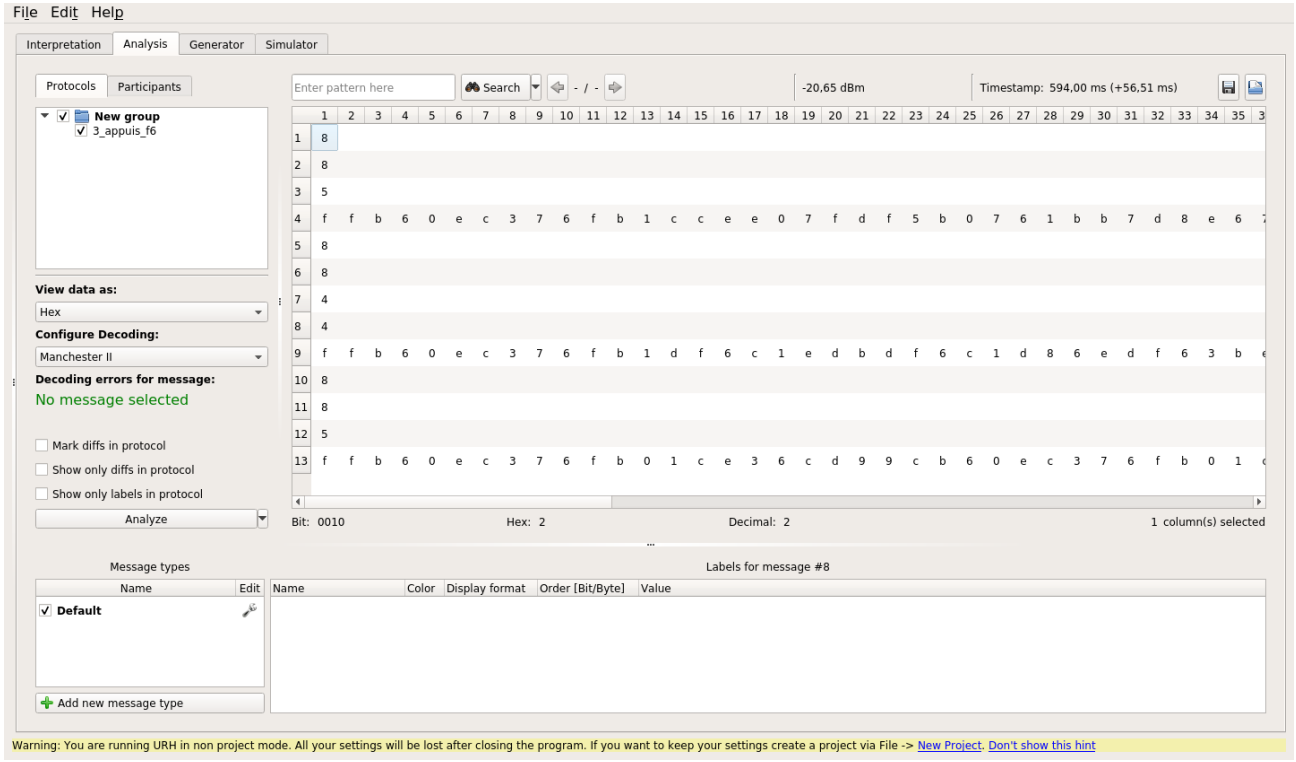


Figure 9 - Trame peu lisible

La Figure 9 permet d'observer que les données ne sont pas claires, n'ont pas de formes particulières. Je suis retourné dans *Interpretation* et j'ai remarqué que la dernière partie de chaques signaux pouvait être sectionnée en 6 sous-parties (Figure 10).

Figure 10 - 2e partie du signal

Mais le logiciel ne sélectionne pas les données 'à l'œil', il le fait en fonction du nombre de bit '0' à la suite. Cette option est réglable en cliquant sur la clé à molette à côté de l'option de modulation, puis en modifiant la valeur de *Pause Threshold*. J'ai mis la valeur 2 car elle divise les données (la suite de bits) en 6 parties et correspond à mon observation précédente.

En retournant dans la partie *Analysis*, j'ai sélectionné le codage Manchester II pour voir la forme des données. On remarque Figure 11 qu'elles sont beaucoup plus 'lisibles', mais inexploitable de manière correcte car le code réel utilisé est le codage SLH LR de FAAC.

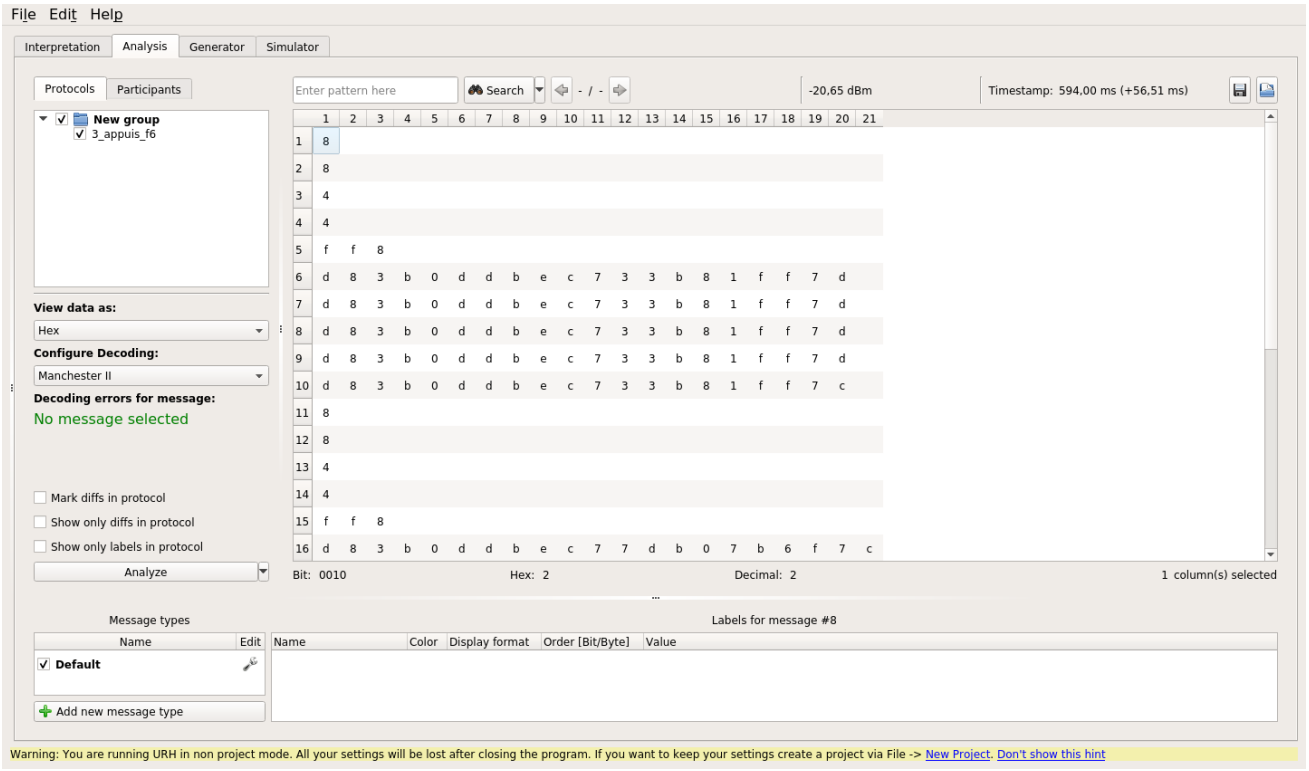


Figure 11 - Données structurées

Afin de continuer mes tests, j'ai changé de boîtier de commande pour un modèle plus ancien. Le T224-1 de FAAC (oui, encore eux), fonctionnant sur une fréquence de 224.7 Mhz.

La technique de codage de ce boîtier est le dip-switch (mini-interrupteur). Ce codage est réglé manuellement par l'utilisateur en changeant l'état de mini-interrupteurs présents dans le boîtier. La séquence doit être la même que le récepteur. Dans le boîtier T224-1, le système possède 8 mini-interrupteurs à 3 états (1,0,-1).

Contrairement au boîtier précédent, le signal de celui-ci est beaucoup plus lisible et on remarque à l'œil Figure 12, la présence d'un motif qui se répète.

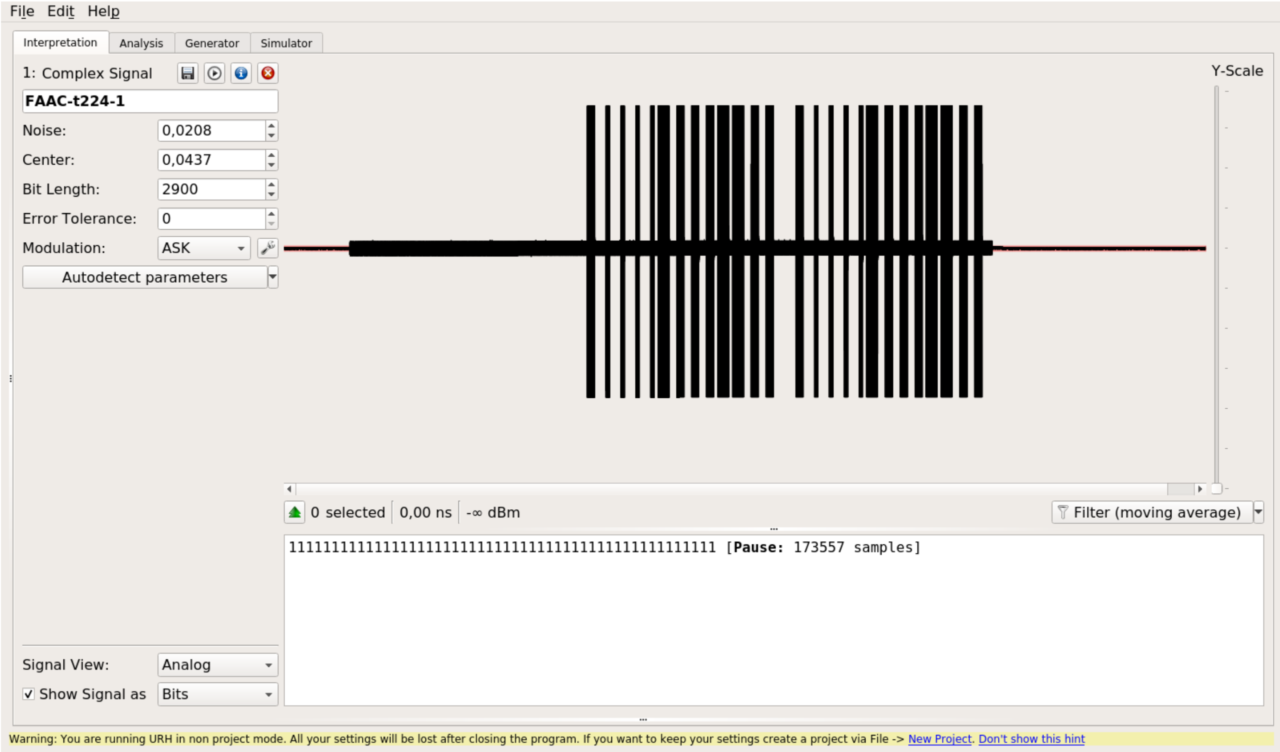


Figure 12 - Signal reçu, 1 appui

Cependant la trame en bit est composée seulement de ‘1’. Cela est dû au seuil minimal d’amplitude défini automatiquement par le logiciel lors de la démodulation. Pour régler ça je suis allé dans la vue démodulée du signal (*Signal View : Demodulated*) puis j’ai ajusté manuellement le niveau de la partie ‘0’ (Figure 13, en rose). Par la même occasion j’ai sélectionné le nombre d’échantillons d’un bit (même fonctionnement que pour le boîtier précédent), qui est de 3020. Ici l’option *pause threshold* (seuil qui permet de sectionner les messages) a été correctement auto paramétrée.

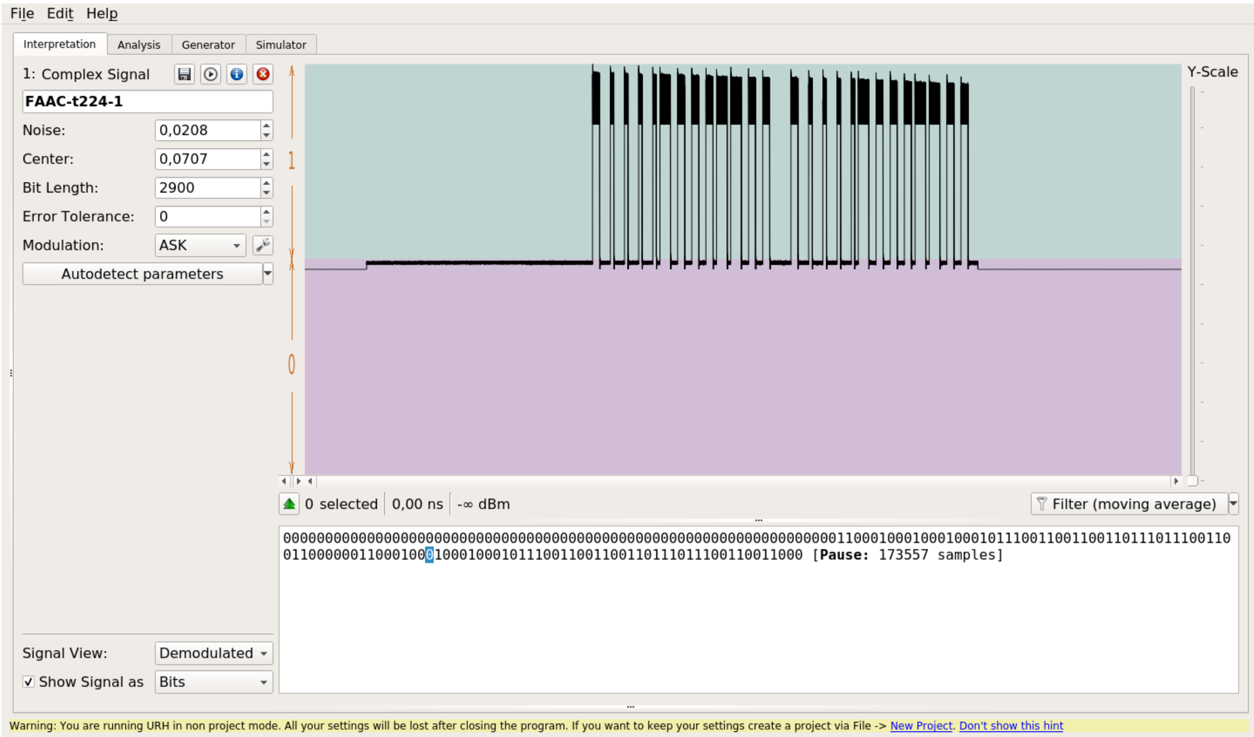


Figure 13 - Vue du signal démodulé

La séquence des mini-interrupteurs du boîtier est 1111-1000. Pour comprendre comment le code fonctionne, j’ai changé la séquence précédente pour comparer ensuite les données.
Nouvelle séquence : 11111111.

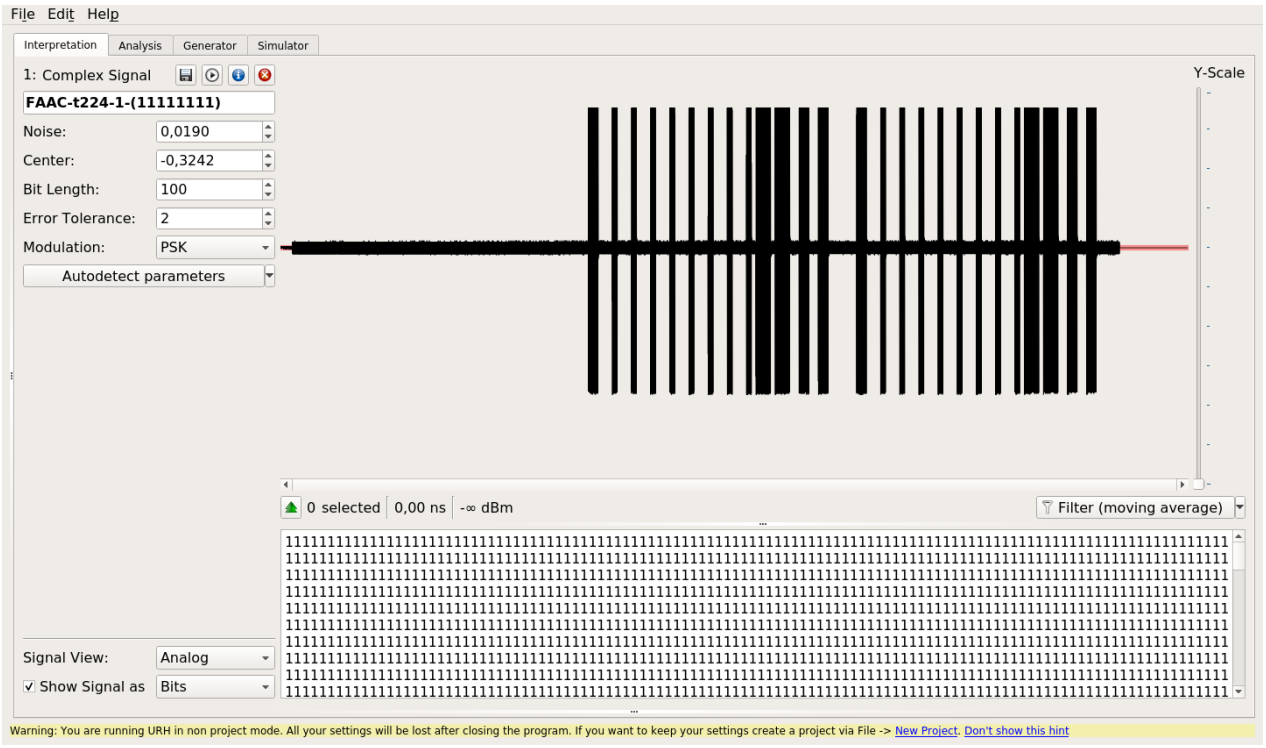


Figure 14 - Signal reçu pour la séquence '11111111'

Encore une fois, un motif se répète. Le fonctionnement du code commence à être clair : la première impulsion est un préambule qui annonce le début du code, s'ensuit le code des mini-interrupteurs. J'ai clairement pu observer les 8 impulsions fines représentant les 8 '1' (Figure 14) que j'ai réglé. La fin d'émission est annoncée par 2 impulsions plus longues, suivies de 2 impulsions moyennes.
J'ai une fois de plus changé de séquence : -1-1-100111. On peut voir Figure 15 que le signal possède toujours les mêmes caractéristiques : impulsion d'amorce, 8 impulsions correspondant aux mini-interrupteurs, 4 impulsions de fin de signal. Pour l'envoie de la séquence des interrupteurs, les impulsions longues traduisent la valeur -1, les impulsions moyennes la valeur 0, et les impulsions courtes la valeur 1.

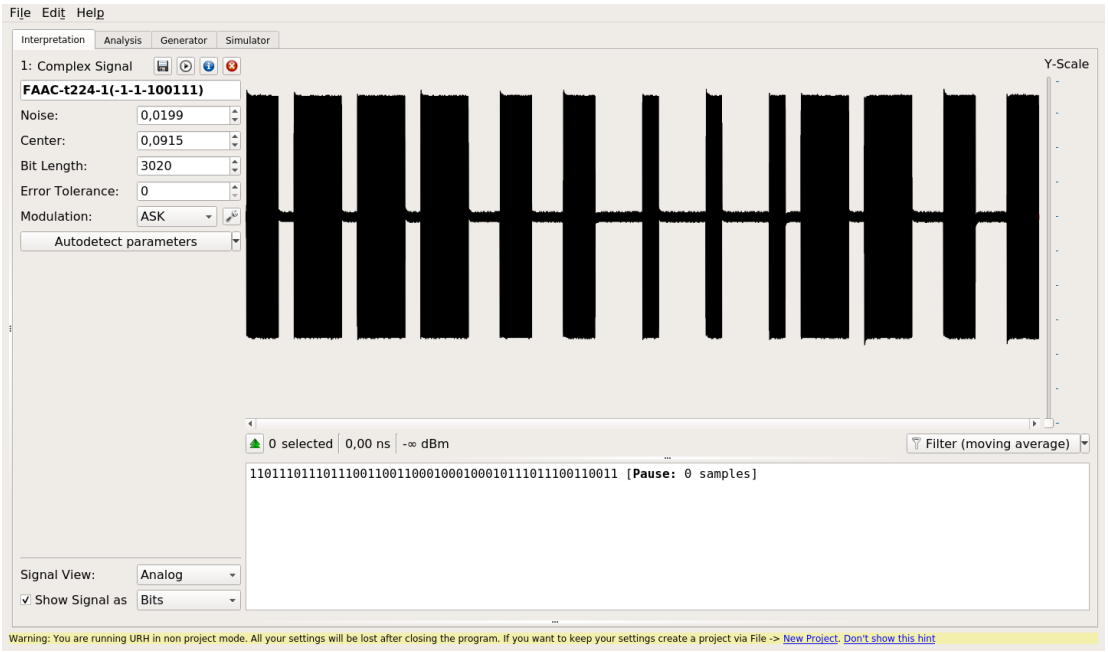


Figure 15 – Signal reçu pour la séquence '-1-1-100111'

Dans la figure suivante, j'ai labellisé (surligné les différentes parties) et rassemblé les 3 signaux pour les comparer. De haut en bas, le signal avec le code initial '1111-1000', le code '11111111', et le code '-1-1-100111'.

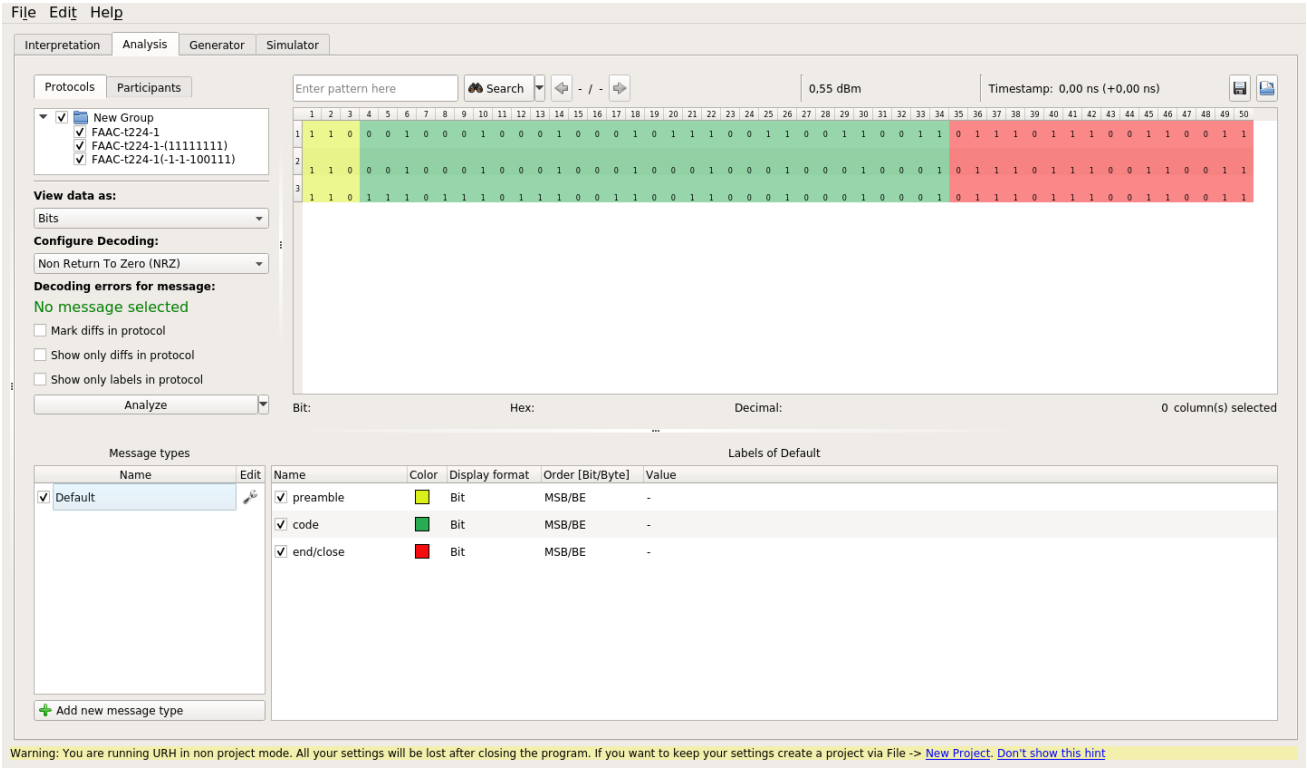


Figure 16 - Comparaison et labellisation des 3 signaux

4.2.2.1 Mise en patique

Sachant que le codage est simple, sans roulement, j'ai essayé d'effectuer un rejeu sur le portail correspondant à mon boîtier. Le principe du rejeu est de capter le signal d'ouverture, et de le rejouer tel quel, sans modification.
Ce portail est chez moi, tout est légal. Pour le rejeu j'ai utilisé le HackRf One.

```
sudo hackrf_transfer -r 224.7Mhz-10M.complex16s -f 224700000
call hackrf_set_sample_rate(10000000 Hz/10.000 MHz)
call hackrf_set_freq(224700000 Hz/224.700 MHz)
Stop with Ctrl-C
19.9 MiB / 1.000 sec = 19.9 MiB/second
19.9 MiB / 1.000 sec = 19.9 MiB/second
19.9 MiB / 1.000 sec = 19.9 MiB/second
20.2 MiB / 1.000 sec = 20.2 MiB/second
19.9 MiB / 1.000 sec = 19.9 MiB/second
^CCaught signal 2
19.9 MiB / 0.994 sec = 20.1 MiB/second

Exiting...
Total time: 5.99456 s
hackrf_stop_rx() done
hackrf_close() done
hackrf_exit() done
fclose(fd) done
exit
```

Figure 17 - Commande d'écoute et son résultat

J'ai indiqué à mon SDR qu'il doit enregistrer le signal dans un fichier avec l'option –r (**r**ecord in english), sur la fréquence de mon boîtier (244.7Mhz) avec l'option –f.

Je me suis placé à proximité du récepteur de mon portail puis j'ai lancé la commande `hackrf_transfer`. Cette fois c'est –t, pour **t**ransfer, qui indique quel fichier doit être utilisé. Pour le rejeu, j'ai tout simplement utilisé le fichier que j'ai créé avec la commande d'enregistrement. On retrouve l'option –f pour la fréquence d'émission, et une nouvelle option, -x, qui indique le gain de la puissance d'émission du hackRf en décibel. Cette option est indispensable car même à côté d'un récepteur, sans cette option, le signal émis peut ne pas être reçu.

```
sudo hackrf_transfer -t 224.7Mhz-10M.complex16s -f 224700000 -x 40
call hackrf_set_sample_rate(10000000 Hz/10.000 MHz)
call hackrf_set_freq(224700000 Hz/224.700 MHz)
Stop with Ctrl-C
19.9 MiB / 1.000 sec = 19.9 MiB/second
19.9 MiB / 1.000 sec = 19.9 MiB/second
19.9 MiB / 1.000 sec = 19.9 MiB/second
20.2 MiB / 1.000 sec = 20.2 MiB/second
11.0 MiB / 1.000 sec = 11.0 MiB/second

Exiting... hackrf_is_streaming() result: streaming terminated (-
1004)
Total time: 5.00103 s
hackrf_stop_tx() done
hackrf_close() done
hackrf_exit() done
fclose(fd) done
exit
```

Figure 18 - Commande d'émission et son résultat

Sur la Figure 18 on peut voir que le HackRF est en émission : 'hackrf_is_streaming( )'. En me plaçant à côté du portail (Annexe f), j'ai réussi à l'ouvrir avec cette commande, donc du rejeu. Le HackRF n'est pas un matériel très précis, l'antenne est universelle, il y a beaucoup de pertes, c'est pour ça qu'il faut se placer à proximité de notre objectif, ou alors utiliser des amplificateurs (autres que l'option –x, de vrai amplificateurs) et autres techniques.

Les travaux que j'ai effectués et détaillés au cours de cette partie vont aider la cellule C3D dans laquelle j'ai travaillé à réaliser des audits dans le domaine industriel.

5 Conclusion

Au cours de mon stage au sein de la cellule C3D, malgré un environnement très confidentiel, j'ai pu apporter mon soutien dans la réalisation et la mise en place de systèmes qui permettront à la cellule d'auditer des projets de la DIDPE et de la DPNT de manière plus générale.

Ce stage d'immersion en entreprise m'a permis de découvrir le milieu de la cyber-sécurité, qui est très demandeur aujourd'hui, et qui devient indispensable avec la numérisation des données.

Ce domaine nécessite une grande rigueur, et des connaissances rédactionnelles avec de la communication adaptée pour pouvoir énoncer clairement les besoins en termes de sécurité. Contrairement aux idées reçues, « les ingénieurs en cyber-sécurité ne font pas que de la programmation », mais sont sollicités en grande partie pour la rédaction de rapports et des propositions de solutions de sécurités.

J'ai pu observer des systèmes réseaux et du matériel opérationnel, comme les salles de contrôle commande, des salles serveurs et beaucoup d'équipements liés à la cyber-surveillance.

Travailler dans ce milieu m'a éclairci dans mes choix de carrières. La sécurité informatique est un domaine qui m'intéresse, autant que la programmation, deux domaines dans lesquels j'aimerais travailler.

Remerciements

Je remercie M. Yann HILIQUIN - Chef de Service de la Cellule C3D de m'avoir accepté en tant que stagiaire au sein de son groupe, ainsi que ses membres (autres ingénieurs, alternants) pour avoir été présents pour me conseiller et m'accompagner dans mes travaux.

Je tiens à remercier tout particulièrement mon tuteur, M. Jean-Hugues ZORIO – Ingénieur dans la cellule C3D, pour ses connaissances, compétences et historiques techniques très enrichissantes qu'il m'a fait partager, et pour m'avoir encadré.

Je remercie aussi plus globalement les autres services d'EDF-DIPDE, dont C2I, qui ont pu me montrer divers équipements informatiques lors de ce stage (baies de commandes, serveurs, etc.).

Glossaire

API : Application Programming Interface. C'est une façade par laquelle un logiciel offre des services à d'autres logiciels.

Bash : Abréviation de « Bourne-Again shell ». C'est un interpréteur en ligne de commande présent sur de nombreuses distributions Linux.

C3D : Centre de Compétences en Cyber-Sécurité de la DPNT.

Charset : Fusion des mots « character » et « set », traduisible directement par « un ensemble de caractères ».

Cluster : Un cluster ou grappe de serveurs et permet de rassembler les performances de plusieurs machines, de fusionner leurs ressources, à travers un réseau.

DIPDE : Division de l'Ingénierie du Parc, de la Déconstruction et de l'Environnement.

DIPNN : Division de l'Ingénierie du Parc, Nouveau Nucléaire.

DPNT : Direction du Parc Nucléaire et Thermique

EDF : Electricité de France.

ENEDIS : Gestionnaire du réseau de distribution d'électricité, anciennement **ERDF** (pour *Électricité Réseau Distribution France*) (haute tension A (ex. moyenne) et basse tension – 20 kV à 230-220 V).

GPU : Graphics Processing Unit. C'est un circuit intégré présent sur la plupart des cartes graphiques qui effectue les fonctions de calcul et d'affichage.

Keyspace : Désigne l'ensemble des possibilités ou combinaisons pour un mot d'une certaine taille, composé de certains caractères. Le nombre de possibilité pour un mot = (nombre de valeurs pour 1 caractère)^(nombre de caractères).

Machine virtuelle : Simulation d'un appareil informatique, sur un ordinateur physique à l'aide d'un logiciel. Abrégé en 'VM' (Virtual Machine).

OpenCL : « *Open Computing Language est une API permettant de tirer parti de la puissance des GPU, en d'autres termes, OpenCL permet au programme d'utiliser la carte graphique pour faire des calculs, parallèlement ou séparément des calculs faits par le processeur* » (reneca, 2 Août, 2014, Wiki ubuntu-fr). Cette API est désormais présente dans presque tous les GPU et processeurs.

Rétro-ingénierie : La rétro-ingénierie est le principe d'observer le fonctionnement interne d'un système.

RTE : Gestionnaire du Réseau Transport d'Electricité (haute tension B de 400kV à 63kV).

Script : Programme simple et court permettant d'automatiser l'exécution des tâches simples et répétitives, comme des suites de commandes, des modifications de fichiers etc…

Bibliographie

Plaquette de présentation EDF-DIPDE.

Site internet EDF.

Sites intranet VEOL (Vivre EDF On Line).

Wiki ubuntu-fr, 'Reneca' 2014 [consulté en mai 2019], OpenCL. Via https://doc.ubuntu-fr.org/opencl

Hashcat, 2019 [consulté en avril 2019], via https://hashcat.net

Supinfo, Adrien FLAHAUT, 2017 [consulté en avril 2019]. Découverte de hashcat. Via https://www.supinfo.com/articles/single/6242-decouverte-hashcat

4armed, William Hurer-Mackay, 2016 [consulté en avril 2019]. How to perform a mask attack using Hashcat. Via https://www.4armed.com/blog/perform-mask-attack-hashcat/

Chiffrer, "Max", fladnaG.net [consulté en avril 2019]. Via https://chiffrer.info

Wikipedia, 2019 [consulté en avril 2019]. Chiffrement. Via https://fr.wikipedia.org/wiki/Chiffrement

Reddit, 2013 [consulté en avril 2019]. John The Ripper vs oclHashcat-lite. Via https://www.reddit.com/r/crypto/comments/yuqyi/john_the_ripper_vs_oclhashcatlite/

Rainbowcrack, 2019 [consulté en mai 2019]. Via https://project-rainbowcrack.com/

A Stich in Time, Paul Under 2009 [consulté en mai 2019]. Rainbow Tables – Part 5 (Chains and Rainbow Tables). Via https://stichintime.wordpress.com/2009/04/09/rainbow-tables-part-5-chains-and-rainbow-tables/

Kestas Kuliukas, Kestas Kuliukas [consulté en mai 2019]. How Rainbow Tables work. Via http://kestas.kuliukas.com/RainbowTables/

Stackexchange, "Crunge" 2018 [consulté en mai 2019]. What are rainbow tables and how are they used?. Via https://security.stackexchange.com/questions/379/what-are-rainbow-tables-and-how-are-they-used/440#440

Wikipedia, 2018 [consulté en mai 2019]. Rainbow table. Via https://fr.wikipedia.org/wiki/Rainbow_table

Mieux Coder, "aranud" 2008 [consulté en mai 2019]. Rainbow tables. Via http://mieuxcoder.com/2008/01/02/rainbow-tables/

Pwnation, Doopel 2017 [consulté en mai 2019]. Garage Door RF Communication. Via https://pwnation.github.io/post/garage_door/rfcommunication/

Gqrx, Alexandru Csete 2015 [consulté en mai 2019]. Practical tricks and tips. Via http://gqrx.dk/doc/practical-tricks-and-tips#more-229

Console Wowboys, Ficti0n 2017 [consulté en mai 2019]. Hacking Everything with RF and Software Define Radio – Part 1. Via http://console-cowboys.blogspot.com/2017/10/hacking-everything-with-rf-and-software.html

Institut Universitaire de Technologie,
Aix-Marseille Université

ANNEXES
Diplôme Universitaire de Technologie
Spécialité Réseaux et Télécommunications

Étude et documentation de systèmes pour la recherche de vulnérabilités

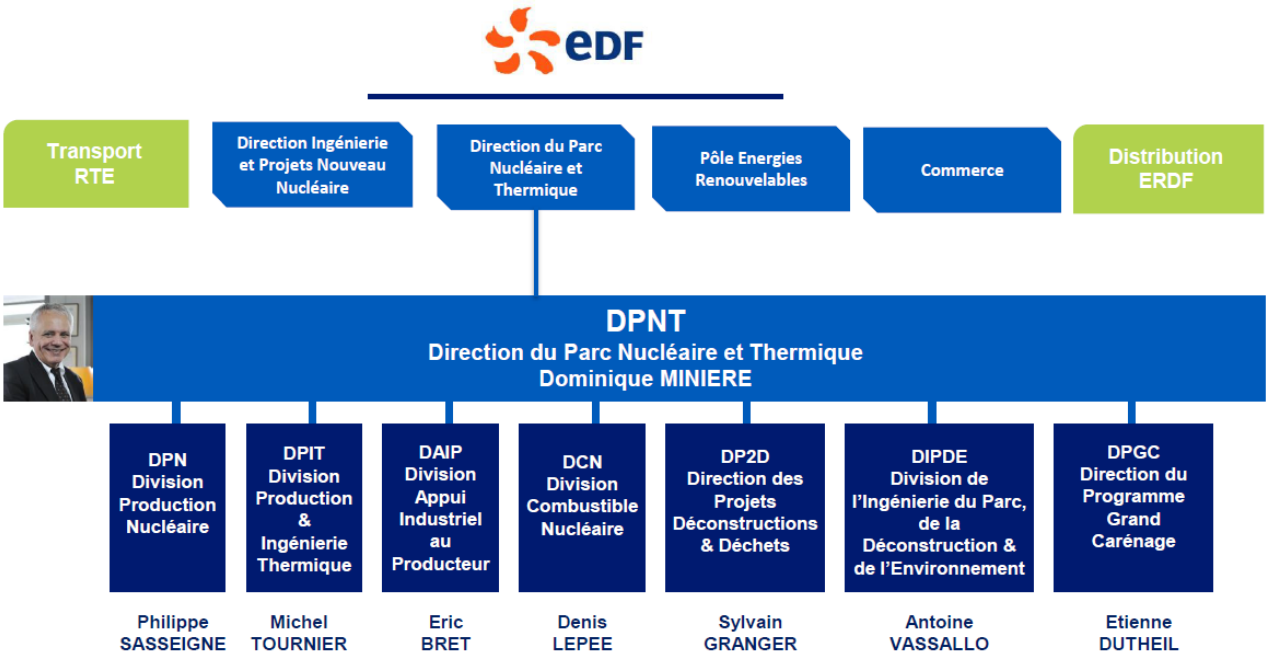
Alexandre COL

EDF DIPDE

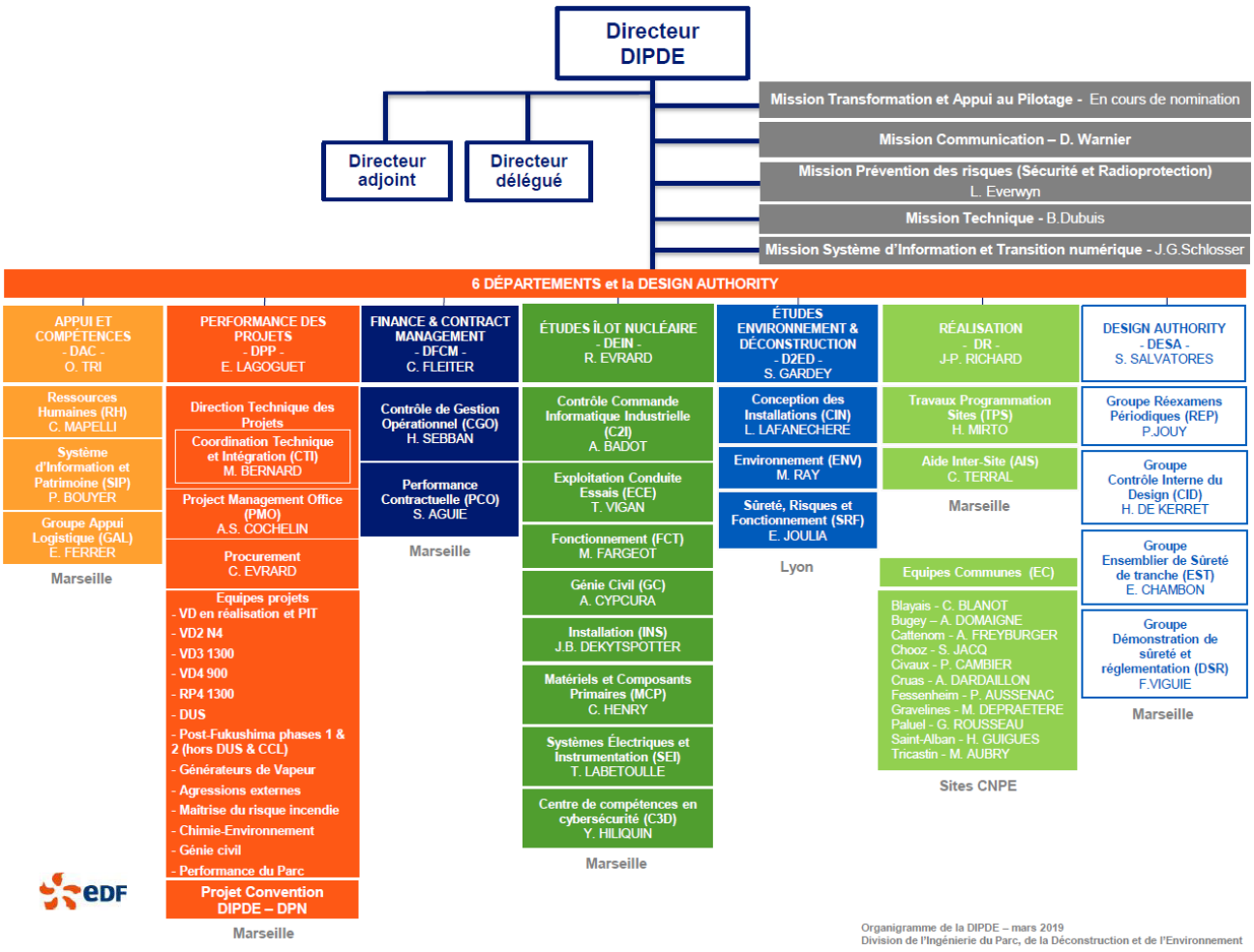
Responsable entreprise : Jean-Hugues ZORIO

Responsable académique : Éric SOCCORSI

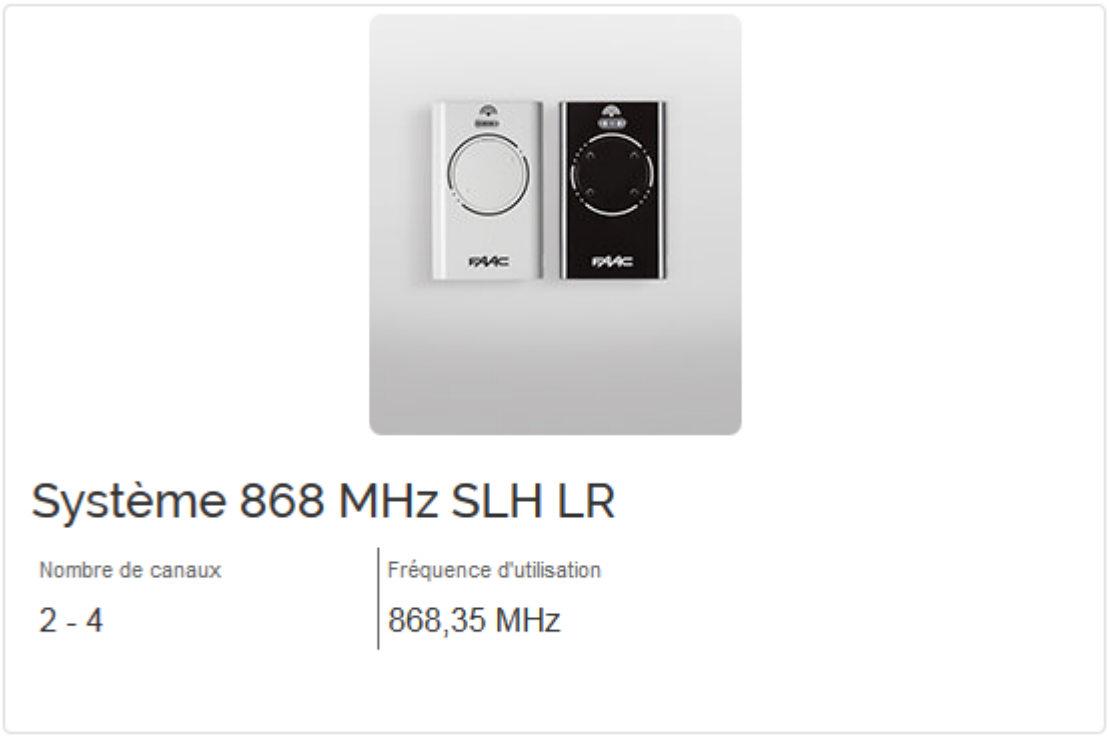
Annexes



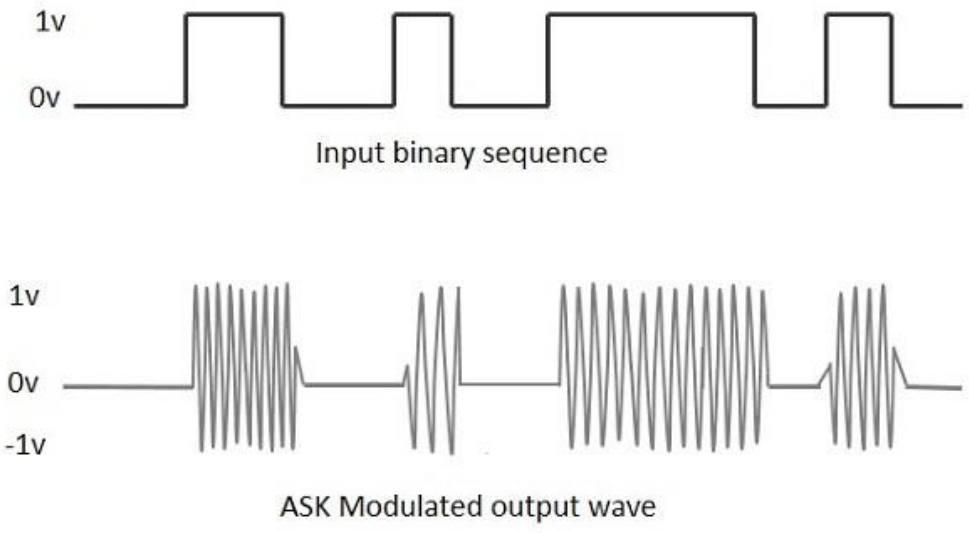
Annexe a - Organigramme global d'EDF & de la DPNT (2017, ERDF est maintenant ENEDIS)



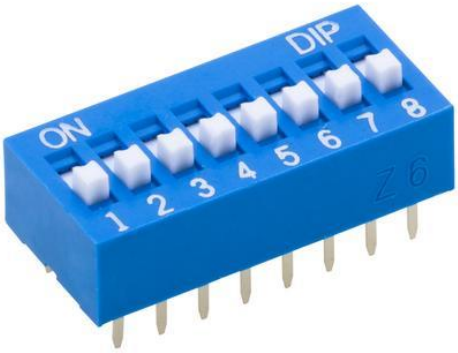
Annexe b - Organigramme détaillé de la DIPDE (2019)



Annexe c - Boîtier de commande FAAC



Annexe d - Exemple de modulation d'amplitude



Annexe e – Un ‘dip-switch’ de 8 interrupteurs

Annexe f – Mon installation d’essai de rejeu

Documentation Hashcat

OPTIONS

-a (--attack-mode) : type d'attaque (3 pour le brute-force)
-m (--hash-type) : type de hash.
-o (--output-file) : fichier qui contiendra les mdp crackés.
-O (--optimized-kernel-enabled) : Utilisation de noyaux hashcat plus performants, mais limitation de la taille des mots de passe (en fonction de l'algo. de hash). Cependant la vitesse de calcul est multipliée par 2.
-w (--workload-profile) : défini le mode de consommation d'énergie et donc la puissance.
-1 (--custom-charstet1) : défini un charset applicable à un masque.
Max 4 charset (-1 -2 -3 -4).
--increment --increment-min= / --increment-max= : permet de délimiter l'espace de recherche.
fichier_de_hash : fichier contenant les hash de mot de passe.
Masque : la forme du mot de passe

Forme typique :
hashcat -a [#] -m [#] -o [nom_fichier] -O -w [#] -1 [?x…] --increment --increment-min=[#] [fichier_de_hash] [masque]

EXPLICATIONS

Fichier de hash

Si le fichier contient des hash de différents type, hashcat va ignorer ceux qui ne correspondent pas à la méthode annoncée avec l'option -m.

Le masque

Dernière option d'une commande hashcat, le masque sculpte le mot de passe. Il permet d'affecter à chaque caractère une plage de valeurs (par valeurs il est entendu minuscules, majuscules, chiffres, caractères spéciaux…).
Par exemple, pour définir que le/les mots de passe recherchés commencent par une majuscule seulement, puis sont composés de 3 minuscules et terminés par un chiffre, il faut écrire **?u?l?l?l?d** (cf « Valeurs »). Il ne faut pas oublier le « ? ».

Si un charset a été défini, il est utilisable comme ceci : **?1?1?1 …** (1 correspond au numéro du charset).
Dans le masque, un charset peut être combiné avec une plage de valeurs : **?1?u?1?d**
Il faut spécifier un masque personnalisé (voir paragraphe rouge ci-dessous).

Définition d'un charset

Un charset permet de combiner plusieurs plages de valeurs, sous une seule donnée. L'utilisation des charset se fait lorsque des caractères peuvent valoir différentes plages de valeurs (minuscules et chiffres…).
L'option -1 (ou -2 etc…) doit être suivie des valeurs que pourront prendre les caractères affectés à ce charset.

Par exemple, il est possible de dire que le charset 1 est composé de majuscules ou de minuscules en écrivant **-1 ?l?u** (l'ordre de placement de **?l** et **?u** [exemple] n'as pas d'importance, ce n'est pas un masque, mais un regroupement de plages de valeurs).
En écrivant **?1?1?1** comme masque par exemple, on dit que le/les mots de passe sont composés de 3 caractères pouvant être soit des minuscules (**?l**) soit des majuscules (**?u**).

Si aucun charset n'est défini, hashcat prendra ceux définis par défaut et les appliquera à un masque lui aussi par défaut (majuscule seulement au premier caractère, caractères spéciaux présent à partir du 7e caractère). Il faut donc spécifier un charset et un masque que vous pensez en adéquation avec la forme de votre mot de passe/qui pourraient correspondre à n'importe quel mot de passe.

Increment

Lors de l'utilisation des options d'incrémentations, il faut toujours ajouter **--increment**. Increment-min définit la longueur (nombre de caractères) minimale du mot de passe, increment-max la longueur maximale.
Par exemple, si on sait qu'un mot de passe a une longueur supérieure ou égale à 5, il faudra écrire --increment --increment-min=5. Increment-max est disponible, mais n'est pas utile dans notre cas (longueur max = masque). Si on connaît la taille du mot de passe, pas besoin de l'option increment (cf Masque & increment).

Masque & increment

•Si pas d'option d'incrémentation, hashcat ne recherchera que des mots de passe de la taille du masque (avec les charset/formes de caractères donnés) !
•Si la valeur de increment-min > longueur du masque, ça ne marche pas ('Invalid Mask').
•Si la valeur de increment-min < longueur du masque, la recherche commencera par les x premiers caractères du masque, x étant la valeur de increment-min.
•Si la valeur de increment-max < longueur du masque, la recherche s'arrêtera à y caractères du masque, y étant la valeur de increment-max.
•Si la valeur de increment-max > longueur du masque, la recherche s'arrêtera à la longueur maximale du masque (increment-max n'aura aucune influence).

Exemple :
hashcat -a 3 -m 0 -o found.txt -O -w 4 -1 ?l?u?d --increment --increment-min=3 password.hash ?1?1?1?1?1?1

-a 3	➔ brute-force
-m 0	➔ raw MD5
-o *found.txt*	➔ mdp crackés sont écrits dans found.txt
-O	➔ **Il est vivement conseillé d'utiliser cette option d'optimisation !**
-w 4	➔ Consommation maximale, pleine puissance
-1 ?l?u?d	➔ Charset 1, ici : soit min. soit maj. soit chiffres de 0 à 9
--increment	➔ On avertit qu'on utilise les options d'incrémentation
--increment-min=3	➔ Le crack commence directement par des mdp de 3 caractères
password.hash	➔ fichier contenant le/les hash à cracker
?1?1?1?1?1?1	➔ Masque de 6 caractères pouvant prendre les valeurs décrites dans le charset 1.

Le programme va exécuter une attaque brute-force sur un hash de type raw MD5 (écrit dans *password.hash*). Le charset 1 définit la règle suivante : le caractère ne pourra être que minuscule ou majuscule ou chiffre. Il est spécifié que la recherche doit commencer à partir des mots de passe de 3 caractères. Hashcat utilisera donc les 3 premiers caractères définis dans le masque pour commencer.
Après avoir essayé toutes les combinaisons pour 3 caractères, il passe aux mots de passe de 4 caractères, et utilise les 4 premiers caractères du masque, etc… jusqu'à la longueur maximale du masque, c-a-d 6 caractères.
Cette commande permet de rechercher des mots de passe de 3 à 6 caractères composés de minuscules, majuscules, chiffres.

VALEURS

-m

#	Nom
0	MD5
100	SHA-1
1400	SHA-256
1700	SHA-512
500	MD5 (Unix), Cisco-IOS, 1
7400	SHA-256 (Unix), 5
1800	SHA-512 (Unix), 6
POUR LA LISTE COMPLETE ➔ hashcat --help (partie « Hash modes »)	

-w

#	Performance	Conso. énergie	Effet sur le bureau (affichage)
1	Faible	Faible	Minimal
2	Défaut	Eco.	Remarquable
3	Elevée	Elevée	Ne répond plus (ou presque)
4	Nightmare	Insane	Headless

-1 (-2 -3…) / masque

?	Charset
l	abcdefghijklmnopqrstuvwxyz
u	ABCDEFGHIJKLMNOPQRSTUVWXYZ
d	0123456789
s	!“#$%&’()*+,-./:;<=>?@[\]^_`{|}~
a	?l?u?d?s

Script

```
#!/bin/bash

#Author: Alexandre COL
#19/04/2019
#
#Dernieres modifications: 03/06/2019

#
#Le script se charge de faire tourner une commande hashcat en arrière-plan et
envoie
#des demandes de bypass régulièrement (toutes les 30 sec) ce qui va permettre
d'obtenir
#les infos sur le temps de chaque keyspace (si plusieurs).
#Il calcul ensuite le temps total indiqué par hashcat.
#

#-Le texte d'information utilisateur-
echo -e "\033[4m                                                                \033[0m
|
|  FONCTIONNEMENT
|
|  Saisir une commande hashcat.
|  Le script se charge de la faire tourner en arrière-plan (screen) et envoie
|  des demandes de \033[36mbypass \033[0mtoutes les \033[31m30 secondes\033[0m.
|  Cela va permettre de récupérer le temps de calcul annoncé de chaque keyspace
(si plusieurs).
|  Le fichier de log de la cmd screen est \033[31mtemps_total_log.txt\033[0m, il
est supprimé et recréé à chaque lancement de ce script.
|  Un calcul du temps maximal total en fonction des temps reçus est renvoyé.
|
\033[4m|                                                                \033[0m"

#-Saisie par l'utilisateur de la commande hashcat-
read -p "Commande Hashcat :" commande

#-Le test pour le fichier de log de la commande "screen"-
if [ -e temps_total_log.txt ]; then
  rm temps_total_log.txt
  echo -e "\033[33m'temps_total_log.txt' existe, il a été recréé\033[0m"
fi
touch temps_total_log.txt

#-La commande "screen" qui va lancer la commande hashcat paramètre en arrière-
plan-
screen -L -Logfile temps_total_log.txt -d -m -S Hashcat $commande
sleep 5 #petit délai (en secondes)

#-Tant que la commande hashcat tourne, on envoie des demandes de bypass. Le
screen s'arrête dès que hashcat est terminé-
t=0
while screen -list | grep "Hashcat" > /dev/null; do
  screen -S Hashcat -p 0 -X stuff "b"
  sleep 30
  ((t+=1))
  echo "Bypass: ${t}"
done

sleep 3 #encore un petit delai
```

```
#---Calcul et affichage du temps total---

#Variables qui s'incrémentent et qui permettront le calcul du temps total
sec=0
min=0
hour=0
day=0
year=0

#hashcat donne des temps sous la forme: (XX unité, XX unité). Ici les 2 parties
séparées par la virgule sont traitées.
cat temps_total_log.txt | grep "Time.Estimated" | awk -F "(" '{print $2}' | tr -d
',' | tr -d ')' | ( while read ligne; do
#La parenthèse avant le while permet de rester dans le processus de la boucle
(pour afficher les variables de la boucle après le "done")
  val1=$(echo ${ligne} | cut -d " " -f1)
  unite1=$(echo ${ligne} | cut -d " " -f2)
  val2=$(echo ${ligne} | cut -d " " -f3)
  unite2=$(echo ${ligne} | cut -d " " -f4)

#1ere partie (premiere chaine de type "XX unité") des temps donnés par hashcat
  case $unite1 in
    *sec*)
      sec=$((sec+val1))
      ;;
    *min*)
      min=$((min+val1))
      ;;
    *hour*)
      hour=$((hour+val1))
      ;;
    *day*)
      day=$((day+val1))
      ;;
    *year*)
      year=$((year+val1))
      ;;
  esac

#2e partie des temps donnés par hashcat
  case $unite2 in
    *sec*)
      sec=$((sec+val2))
      ;;
    *min*)
      min=$((min+val2))
      ;;
    *hour*)
      hour=$((hour+val2))
      ;;
    *day*)
      day=$((day+val2))
      ;;
    *year*)
      year=$((year+val2))
      ;;
  esac
```

```
#Calcul physiquement correcte (pour éviter les 25 heures 78 minutes 112 secondes
par exemple)
  sec2=$((sec-60))
    if (( $sec2 >= 0 )); then
      min=$((min+1))
      sec=$sec2
    fi

  min2=$((min-60))
    if (( $min2 >= 0 )); then
      hour=$((hour+1))
      min=$min2
    fi

  hour2=$((hour-24))
    if (( $hour2 >= 0 )); then
      day=$((day+1))
      hour=$hour2
    fi

  day2=$((day-365))
    if (( $day2 >= 0 )); then
      year=$((year+1))
      day=$day2
    fi
done

#Affichage du temps

echo -e "\033[4m                                                              \033[0m
|
|  Le temps maximal total de calcul est de:
|  \033[33m${year}\033[0m année(s) \033[33m${day}\033[0m jour(s)
\033[33m${hour}\033[0m heure(s) \033[33m${min}\033[0m minute(s)
\033[33m${sec}\033[0m seconde(s)
|
\033[4m|                                                    \033[0m"
) #Fermeture de la fameuse parenthèse

#-----FIN DU SCRIPT-----#
```

Documentation Rainbow tables

Une table arc-en-ciel est un rassemblement astucieux de milliers (et millions !) de hash. Ces tables permettent des recherches très rapides de mots de passe, mais leur fonctionnement est complexe.

Fonction de réduction : processus qui va créer un mot de passe en texte clair à partir d'un hash présent dans la table. Une fonction de réduction doit être cohérente. C'est-à-dire qu'elle doit retourner le même mot de passe quand on lui donne le même hash (empreinte).

Fonctionnement de la création d'une rainbow Table :
Hache (ou créé une empreinte) un mot de passe généré par l'algorithme de création de table, applique une fonction de réduction sur ce hash, re-hache le mot de passe obtenu. Ce processus s'effectue autant de fois que « `chain_len` », en utilisant une fonction de réduction différente à chaque fois. **Une rainbow table ne stockera que le mot initial, et le hash final**. Cela permet d'obtenir une grande quantité de mots contenus dans une quantité réduite de hash, et d'éviter de calculer toutes les valeurs de toute la table (voir fonctionnement d'une recherche).
Problème : les fonctions de réduction produisent des collisions, c'est-à-dire la présence de 2 chaînes identiques, car il y a plus souvent de mots de passe possibles que de hash. Mais les rainbow table résolvent ce problème en utilisant différentes fonctions de réductions. En plus de ce procédé, un « postprocessing » final peut permettre de supprimer chaînes dupliquées.
Plus la taille des chaînes (`chain_len`) est grande, plus la génération sera longue, mais le nombre de mots de passe sera plus élevé.
Plus le nombre de chaîne (`chain_num`) est élevé, plus la taille de la table sera grande, mais le taux de réussite lors d'une recherche sera plus élevé.

Fonctionnement de la recherche avec une rainbow table :
On test si le hash initial (dont on recherche le mot) correspond à un hash dit « final » (dans la dernière colonne).
Si il est trouvé, la chaîne correspondante contient notre mot de passe. Il faut donc recalculer la chaîne comme à la création de la table. On prend le mot de passe initial (celui en début de chaîne), on le hache, on test si le hash correspond à notre hash initial. Si oui, le mot de passe recherché est celui qui vient d'être hashé. Si non, on réduit le hash avec la 1ère fonction de réduction, on hache la réduction, et on re-test l'égalité avec notre hash. Ainsi de suite jusqu'à trouver notre hash, en changeant de fonction de réduction à chaque étape.
Si on ne parvient pas à trouver le hash dès le départ, on le réduit avec la dernière fonction de réduction, et on recherche le mot obtenu dans la 1ère colonne de la table. Si le mot n'est pas présent, on le hash, et le réduit avec l'avant dernière fonction de réduction (on recule pour faire simple).

Générer une rainbow table :

```
rtgen [hash_algo] [charset] [plaintext_len_min] [plaintext_len_max]
[table_index] [chain_len] [chain_max] [part_index]
```

Options:

`hash_algorithm`	:Un seul algorithme de hachage par rainbow table.
`charset`	: définition d'un ensemble de caractères (maj., min, chiffres,…).
`plaintext_len_min`	} Limites de la taille des mots de passe à générer.
`plaintext_len_min`	}
`table_index`	: Sélection de la fonction de réduction (de 0 à 5).
`chain_len`	: Taille des rainbow chain.
`chain_num`	: Nombre de rainbow chain
`part_index`	: Indicateur de fin de table

Exemple avec du MD5 et des mots de passe numérique (chiffre de 0 à 9) entre 1 et 4 caractères :

```
./rtgen md5 numeric 1 4 0 3800 8000000 0
./rtgen md5 numeric 1 4 1 3800 8000000 0
./rtgen md5 numeric 1 4 2 3800 8000000 0
./rtgen md5 numeric 1 4 3 3800 8000000 0
./rtgen md5 numeric 1 4 4 3800 8000000 0
./rtgen md5 numeric 1 4 5 3800 8000000 0
```

Après la génération de table avec différentes fonction de réduction, il faut les « ranger »:

```
./rtsort *.rt
```

`rtsort` va ranger les rainbow chains de chaque tables par point d'arrivé pour permettre la recherche binaire. Cette recherche compare la valeur centrale d'une liste de chaînes rangées (par ordre numérique/aplhabétique…) avec la valeur initiale. Si elles sont différentes, la moitié qui correspond à la valeur subit le même principe : sélection de la valeur centrale, comparaison…

Il faut mettre les tables de mêmes algo de hash dans un même dossier !
Taille d'une table (en octets)= chain_len*16

Exemple de crackage :

```
./rcrack [table_directory] -l [hashes_file]
```