



Write-Up Black Badge Challenge

Par Orikata

Avec l'aide de Phenol



Table of Contents

Step 0 : Entrée dans le challenge	3
Step 1 : Fichier Polyglotte.....	4
Step 2 : Crypto AES et Équation de XOR	6
Step 3 :Reverse de code Micro:Bit	8
<i>Micro:Bit Intel-Hex</i>	8
<i>Micro:Bit en mode REPL</i>	11
<i>Crackme P X Z</i>	11
<i>Xorl et Xorlm</i>	12
Xorl	12
Xorlm	12
<i>Trouver le mot de passe</i>	13
Step 4 : Software Defined Radio.....	15
Step 5 : Stalking et Social engineering	17

Step 0 : Entrée dans le challenge

Lors de mon arrivée, on m'a remis un badge bleu (VIP 😊). Ce badge est fait à partir d'un circuit imprimé ce qui a tout de suite attisé ma curiosité.



On remarque clairement sur une des faces, une LED montée en surface reliée à une bobine gravée. (image de gauche). Premier réflexe, approcher le badge sur le lecteur NFC d'un téléphone, sans grande surprise la LED clignote mais pas de réaction du téléphone.

Le badge est juste étudié pour être alimenté par induction électromagnétique. En regardant la face sans LED on peut lire en effet miroir « Place me somewhere and be enlightened » et un logo en tête de chat dans le coin inférieur gauche.



Ce petit chat nous mène chez <https://virtualabs.fr/>, mais pas de référence au challenge sur la page. Un petit **CTRL+U** pour afficher le code source de la page

<!--

```
#####  
leHACK 2019 :: Black Badge challenge  
#####
```

Want to win a black badge that would give you a lifetime access to leHACK ?

Follow the rabbit and remember: there is only one black badge to win, and it will be given to the first person to complete all the tasks !

==> <https://virtualabs.fr/lh19/lh19-step1.zip>

-->

On télécharge un zip, le challenge commence.

Step 1 : Fichier Polyglotte

Une fois le fichier zip décompressé, on obtient 3 GIFs animés nous faisant comprendre que l'on fait fausse route.



Une fois passé dans un lecteur hexadécimal (Binwalk aurai aussi fait l'affaire) on remarque qu'il s'agit en réalité d'un PDF.

```
00000000: 2550 4446 2d31 2e34 0a25 c3a4 c3bc c3b6 %PDF-1.4.%.....
00000010: c39f 0a32 2030 206f 626a 0a3c 3c2f 4c65 ...2 0 obj.<</Le
00000020: 6e67 7468 2033 2030 2052 2f46 696c 7465 ngth 3 0 R/Filte
00000030: 722f 466c 6174 6544 6563 6f64 653e 3e0a r/FlateDecode>>.
00000040: 7374 7265 616d 0a78 9cb5 5db9 8e24 b911 stream.x...].$.
00000050: f5e7 2bda 16d0 2dde 9909 340a 98be 0c79 ..+...-...4....y
00000060: 0b0c 2063 204f 0720 4380 d6d1 ef8b c133 .. c O. C.....3
00000070: 820c 6691 59b3 bbd8 d99e ea2a 6630 ce17 ..f.Y.....*f0..
```

Mais bien également d'un Zip (le format zip n'impose pas de démarrer au premier octet du fichier)

```
001597b0: 5fdd 591d 7a61 c15a 6970 d1f3 f409 f50d _Y.za.Zip.....
0032b770: 6d6d b66e 7a49 7079 d3d3 f3e5 cd73 5bde mm.nzlpy.....s[.
```

C'est donc un fichier polyglotte Zip → PDF :

L'URL vers l'étape 2 est écrite en blanc sur blanc dans le PDF.
<https://virtualabs.fr/lh19/ea4f5e6a7b1732d7d52cb3beae31f9a1.zip>

grenouilles (c'est toujours plus que les *mammifères*, qui sont environ 5000 espèces). Le plus petit amphibien (et plus petit vertébré) au monde est une grenouille de *Nouvelle-Guinée*, *Paedophryne amauensis* qui mesure seulement 7,7 mm. Le plus grand amphibien vivant est la *Salamandre géante de Chine* (*Andrias davidianus*) avec 1,8 m de long, toutefois bien en deçà des 9 m de *Prionosuchus*, espèce éteinte qui vivait durant le Permien au Brésil.

<https://virtualabs.fr/lh19/ea4f5e6a7b1732d7d52cb3beae31f9a1.zip>

Step 2 : Crypto AES et Équation de XOR

Le step 2 démarre sur un zip contenant 3 fichiers « encrypt_files.py », « next-step.txt », « lehack.org_cds.png ».

Ces 2 derniers fichiers sont illisibles en l'état et encrypt_files.py nous permet de comprendre comment ces fichiers ont été chiffrés.

```
from Crypto.Cipher import AES
from Crypto.Util import Counter
from Crypto import Random

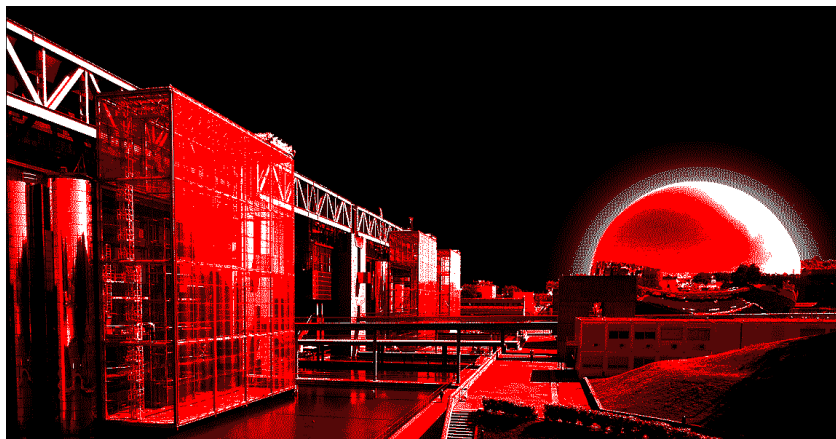
image = open('lehack.org_cds.clear.png','rb').read()
nextstep = open('next-step.clear.txt','rb').read()

nonce = Random.get_random_bytes(8);
key = Random.get_random_bytes(32);

# encrypt image
countf_img = Counter.new(64, nonce)
enc = AES.new(key, AES.MODE_CTR, counter=countf_img)
enc_image = enc.encrypt(image)
open('lehack.org_cds.png','wb').write(enc_image)

#encrypt text
countf_txt = Counter.new(64, nonce)
enc = AES.new(key, AES.MODE_CTR, counter=countf_txt)
enc_text = enc.encrypt(nextstep)
open('next-step.txt','wb').write(enc_text)
```

Le fichier png fait référence à l'image cds.png présente sur le site de lehack.org



Comme on possède un fichier en clair et sa version chiffrée et le texte chiffré avec la même clé, on en déduit l'équation suivante avec pour inconnue le texte en clair.

cyphered text **XOR** plain text== cyphered image **XOR** plain image

plain text== cyphered image **XOR** plain image **XOR** cyphered text

Voici le texte en clair :

leHACK19 Black Badge Challenge
=====

Hi, friend. Do you mind if I call you 'friend' ? Well, we don't know each other, but it looks like you and I are quite stuck here, in a way.

You did great with the two first tasks, and that's awesome ! Well, I have to admit that the last one was a bit tricky (you may call it guessing), but sometimes life is not that obvious too. Anyway, I'm sure you are waiting for the next task to solve, so please follow the little rabbit in his hole:

<https://virtualabs.fr/lh19/0b2fa7c9e4eebc322aa30e85bdb6c53c.wtf>

I wish you luck, and see you in Wonderland !

-- The MadCoder.

C'est parti pour l'étape 3

Step 3 :Reverse de code Micro:Bit

Micro:Bit Intel-Hex

En faisant strings 0b2fa7c9e4eebc322aa30e85bdb6c53c.wtf ou en ouvrant le fichier dans un navigateur web voici ce que l'on peut voir :

```
:10D8C000FFFFFFFFFFFFFFFFFFFFFFFFFFFFF68
:10D8D000FFFFFFFFFFFFFFFFFFFFFFFFFFFFF58
:10D8E000FFFFFFFFFFFFFFFFFFFFFFFFFFFFF48
:10D8F000FFFFFFFFFFFFFFFFFFFFFFFFFFFFF38
:10D90000FE17076D61696E2E7079696D706F7274A4
:10D9100020637261636B6D650A66726F6D206D695D
:10D9200063726F62697420696D706F7274202A0A65
:10D930000A756172742E696E69742839363030291F
:10D940000A7768696C6520547275653A0A20202050
:10D9500020756172742E77726974652863726163D1
:10D960006B6D652E786F726C28637261636B6D6589
:10D970002E5029290A20202020707764203D207411
:10D98000736227270A202020207768696C65202889
:10D990006C656E28707764293E3020616E642028A3
:10D9A0007077645B2D315D20213D20307830442933
:10D9B00029206F72206C656E28707764293D3D3098
:10D9C0003A0A202020202020202063203D2075615D
:10D9D00072742E726561642831290A20202020206B
:10D9E00020202069662063206973206E6F74204EAA
:10D9F0006F6E653A0A20202020202020202075EC
:10DA00007420207077642B3D630A20202020202082
:10DA1000202020202020756172742E777269746531
:10DA20002862272A27290A20202020756172742E57
:10DA300077726974652862275C725C6E27290A20F8
:10DA4000202020696620707764206973206E6F74CF
:10DA5000204E6F6E653A0A20202020202020206969
:10DA6000662028637261636B6D652E6373756D2824
:10DA70007077645B3A365D293D3D307835336576A5
:10DA8000756139643237293A0A202020202020206D
:10DA90002020202020643D637261636B6D652E78C9
:10DAA0006F726C6D28637261636B6D652E5A2C20EA
:10DAB0007077645B3A365D290A20202020202020E0
:10DAC0002020202020696620645B3A325D3D3D6263
:10DAD000274F4B273A0A202020202020202020DA
:10DAE000202020202020756172742E777269746561
:10DAF00028645B323A5D2B62275C725C6E27297763
:10DB0000760A202020202020202020202020656C44
:10DB100073653A0A202020202020202020202069
:10DB200020202020756172742E77726974652863D5
:10DB30007261636B6D652E786F726C2863726163BE
```

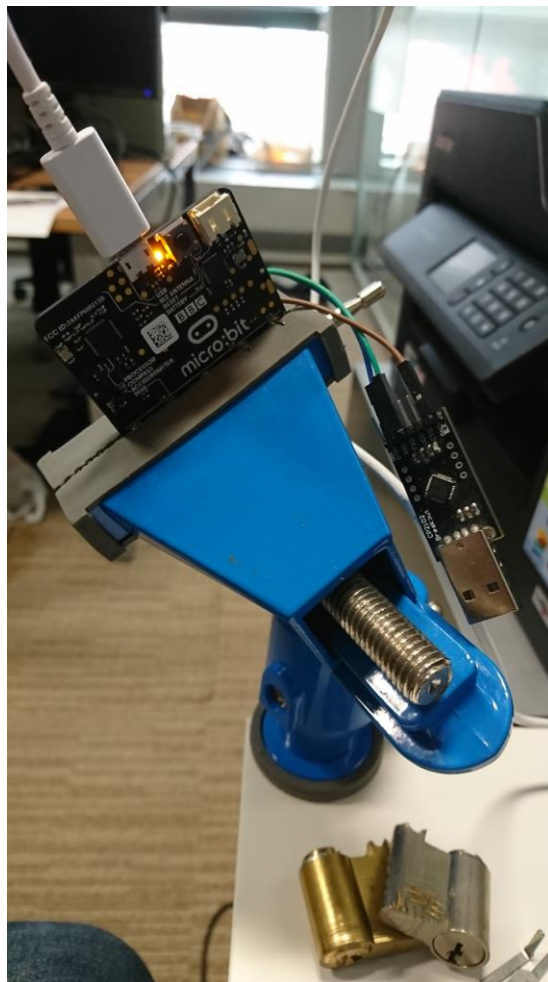
On remarque que cela ressemble au format intel hex. (https://en.wikipedia.org/wiki/Intel_HEX)

Je le décompile dans Ghidra et on constate que ce binaire s'exécute sur une carte de développement Micro:bit

micro:bit v1.0.1+a92ca9b-dirty on 2019-07-05;

```
00036a71 31      ??      31h    1
00036a72 00      ??      00h
00036a73 6d 69 63  ds      "micro:bit v1.0.1+a92ca9b-dirty on 2019-07-05;...
          72 6f 3a
          62 69 74 ...
00036ad0 6d 69 63  ds      "micro:bit with nRF51822"
          72 6f 3a
          62 69 74
```

Le code a d'abord envoyé non modifié directement le code dans une vraie carte micro :bit fournie par Gatewatcher pour l'occasion. Il était également possible de l'émuler avec qemu v4.0 (la seule version à pouvoir émuler un micro:bit)



Sur la sortie série USB interceptée dans un terminal avec Minicom, le Micro:Bit attend un « password: » et si on se trompe on a le message « Nope, try again! »

Après pas mal de recherche sur le fonctionnement du Micro:bit et quelques heures dans Ghidra, le binaire fonctionne comme suit :

- Une VM Micro Python se lance au démarrage.
- Elle va chercher un main.py et le compile en assembleur pour le microcontrôleur.
- La librairie Crackme.py est déjà compilée donc difficile de lire le code pour la « reverse » à moins de comprendre l'assembleur

Voici les différentes sections identifiables dans le fichier au format intel hex.

VM MicroPython Micro:BIT Compilée
Code compilé de crackme.py
vide
Code python main.py lisible
vide

Voici le main.py :

```
0x3d903 main.pyimport crackme
0x3d919 from microbit import *
0x3d931 uart.init(9600)
0x3d941 while True:
0x3d94d uart.write(crackme.xorl(crackme.P))
0x3d975 pwd = tsb"
0x3d985 while (len(pwd)>0 and (pwd[-1] != 0x0D)) or len(pwd)==0:
0x3d9c2 c = uart.read(1)
0x3d9db if c is not None:
0x3d9f5 ut pwd+=c
0x3da0a uart.write(b'*)
0x3da27 uart.write(b'\r\n')
0x3da3f if pwd is not None:
0x3da57 if (crackme.csum(pwd[:6])==0x53evua9d27):
0x3da89 d=crackme.xorlm(crackme.Z, pwd[:6])
0x3dab9 if d[:2]==b'OK':
0x3dad6 uart.write(d[2:]+b'\r\n')wv
0x3db02 else:
```

```
0x3db14 uart.write(crackme.xorl(crackme.X)+b'\r\n')
0x3db50 else:
0x3db5e uart.write(crackme.xoxwrl(crackme.X)+b'\r\n')
```

crackme.csum(**\$pwd[:6]**)==0x53evua9d27 cette ligne nous indique que le mot de passe doit contenir 6 caractères, par contre le checksum contient des caractères inhabituels (v et u).

On constate également que le code fait aussi un « Pré-test » en vérifiant que la chaîne de caractères décodée commence par « OK » (**if d[:2]==b'OK':**)

Micro:Bit en mode REPL

Le fichier au format Intel hex a été tronqué du main.py ce qui permet d'utiliser le port série/USB du Micro:Bit dans l'environnement de développement MU-Python et interagir avec en mode REPL(read-eval-print loop).

Le programme uploadé ne fait donc que charger la VM MicroPython et la librairie pré-compilée crackme.py

Après un import de crackme, il est possible d'utiliser individuellement chaque fonctions qu'il propose.

```
>>> import crackme.py
```

Crackme P X Z

Toutes les chaînes de caractères P X Z contenues dans crackme.py sont illisibles en l'état, seul **xorl** permet de révéler leur contenu.

Z est le Graal de cette étape mais il ne peut être lu sans un **xorlm** avec le bon mot de passe (ou presque).

```
>>> crackme.P
b'Qcpwriul3'
>>> crackme.xorl(crackme.P)
b'Password:' (On retrouve le texte nous demandant le mot de passe)
```

```
>>> crackme.X
b'Omsa)&szp*jklga1'
>>> crackme.xorl(crackme.X)
b'Nope, try again!' (On retrouve le message d'échec)
```

```
>>>crackme.Z
```

```
b'\x17\x1c3_KE1}\x1aW\x02R
```

```
:\x14\n\x0c@)4\x10K\rV`-
```

```
\x03G\x07Af\x148rmr\x0f\x18pz(oR\x14#i5b\x1a\x1edb4y@'
```

Xorl et Xorlm

Xorl et Xorlm ne sont clairement pas de simples XOR, il faut « jouer » avec des valeurs comme 0000000 pour voir si on obtient des résultats ...

Xorl

```
>>> crackme.xorl(b'123456')
```

```
b'000000'
```

Comme on obtient 000000 on peut en déduire que xorl fait un **XOR** de 123456 avec le paramètre d'entrée (ici 123456 **XOR** 123456 = 000000)

```
>>> crackme.xorl(b'\x00'*255)
```

```
b'\x01\x02\x03\x04\x05\x06\x07\x08\t\n\x0b\x0c\r\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f !"#&%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuvwxyz{|}~\x7f\x80\x81\x82\x83\x84\x85\x86\x87\x88\x89\x8a\x8b\x8c\x8d\x8e\x8f\x90\x91\x92\x93\x94\x95\x96\x97\x98\x99\x9a\x9b\x9c\x9d\x9e\x9f\x a0\xa1\xa2\xa3\xa4\xa5\xa6\xa7\xa8\xa9\xaa\xab\xac\xad\xae\xaf\xb0\xb1\xb2\xb3\xb4\xb5\xb6\xb7\xb8\xb9\xba\xbb\xbc\xbd\xbe\xbf\x00\x01\x02\x03\x04\x05\x06\x07\x08\x09\x0a\x0b\x0c\x0d\x0e\x0f\x10\x11\x12\x13\x14\x15\x16\x17\x18\x19\x1a\x1b\x1c\x1d\x1e\x1f\x20\x21\x22\x23\x24\x25\x26\x27\x28\x29\x2a\x2b\x2c\x2d\x2e\x2f\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\x3a\x3b\x3c\x3d\x3e\x3f\x40\x41\x42\x43\x44\x45\x46\x47\x48\x49\x4a\x4b\x4c\x4d\x4e\x4f\x50\x51\x52\x53\x54\x55\x56\x57\x58\x59\x5a\x5b\x5c\x5d\x5e\x5f\x60\x61\x62\x63\x64\x65\x66\x67\x68\x69\x6a\x6b\x6c\x6d\x6e\x6f\x70\x71\x72\x73\x74\x75\x76\x77\x78\x79\x7a\x7b\x7c\x7d\x7e\x7f\xff'
```

Xorlm

```
>>> crackme.xorlm(b'123456', b'000000')
```

```
b'\x00\x00\x00\x00\x00\x00'
```

```
>>> crackme.xorlm(b'123456', b'123456')
```

```
b'\x01\x02\x03\x04\x05\x06'
```

Ces résultats nous font penser que XORLM (x1, x2) = (x1 XOR 123456) XOR x2

```
>>> crackme.xorlm(b'0', b'1')
```

```
b'\x00'
```

```
>>> crackme.xorlm(b'1', b'2')
```

```
b'\x02'
```

```
>>> crackme.xorlm(b'2', b'1')
```

```
b'\x02'
```

```
>>> crackme.xorlm(b'0', b'2')
```

[illegible]

En regardant le code, on en déduit que le mot de passe fait 6 caractères (`pwd[:6]`).

- Soit trouver le mot de passe qui permet d'obtenir le bon checksum
- Soit faire du brute force par déduction car les opérations XOR sont réversibles (c'est la méthode utilisée ici)

En brute forçant un peu XORL et XORLM on trouve les 2 premiers caractères du mot de passe **YU**. Comme on obtient des choses lisibles à intervalle régulier, il faut faire du brute force avec les derniers caractères du mot de passe et remonter jusqu'aux premiers.

```
>>> crackme.xorlm(crackme.Z, b'YUB747')
b'OKrlzto Qj=itaY-)ecuGj.y bZl.h a[gzask\x15e5r mLp.ery\x15a5x.'
```

b'OKgi to DlgiLaL+secuRlty bOjth aNa ask\x00cor mYvtery\x00gox.' (On chauffe)

b'OKgo to DigitaL.secuRity bOoth aNd ask\x00for mYstery\x00box.' (c'est pas parfait mais on comprend le message)

b'OKGo to digital.security booth and ask for mystery box.' (Bingo!!!)

Normalement, c'est à ce moment que la LED du badge aurait dû être utile sur le stand de Digita.Security. Je suis donc allé directement dans les locaux de Digital.Security car j'ai trouvé le mot de passe bien après LeHack 2019, de là j'ai obtenu plein de goodies (merci) et le mail de Damien Cauquil (Virtu) pour passer à l'étape 4.

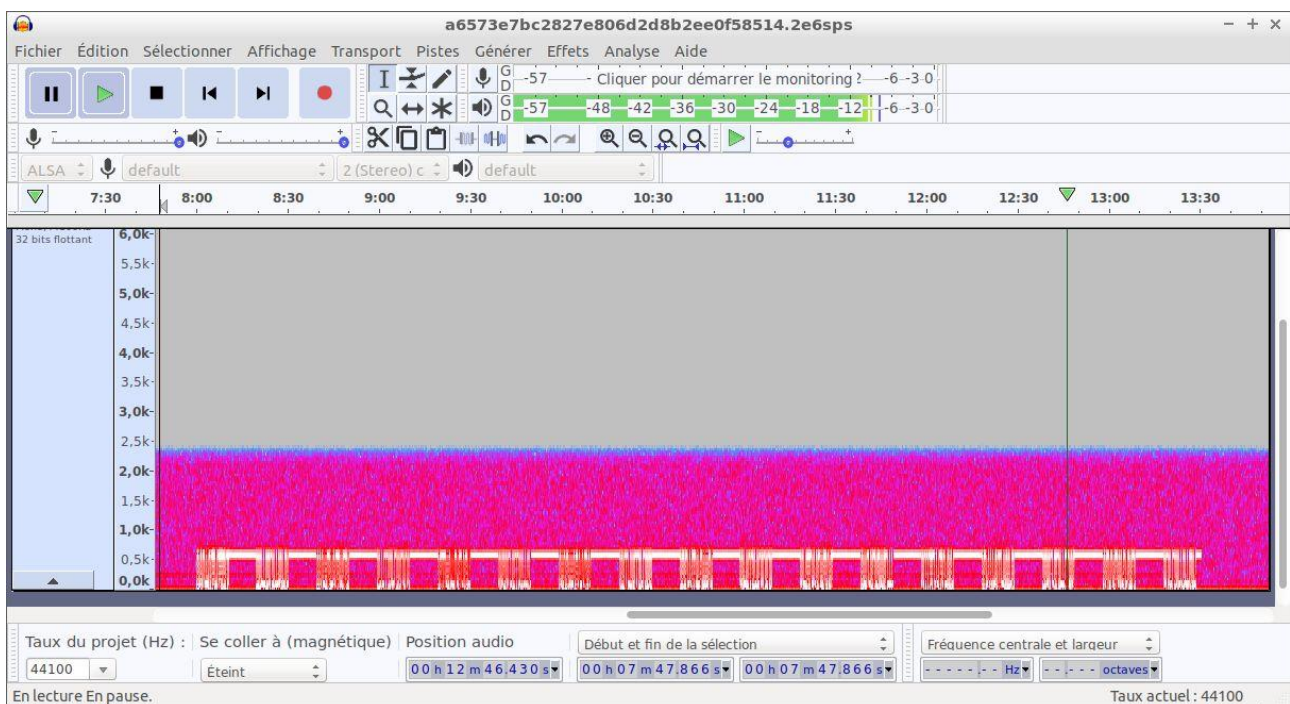
Step 4 : Software Defined Radio.

On part de : <https://t.co/NvnUnJre3V.a>

Le fichier obtenu (a6573e7bc2827e806d2d8b2ee0f58514.2e6sps.cfile) via un lien obtenu par mail est de type inconnu d'après Binwalk.

Une recherche sur les termes **2e6sps** et **cfile** dans google semble montrer qu'il s'agit d'un enregistrement radio pour l'équipement de SDR (software defined radio) USRP à 2 millions de samples par secondes.

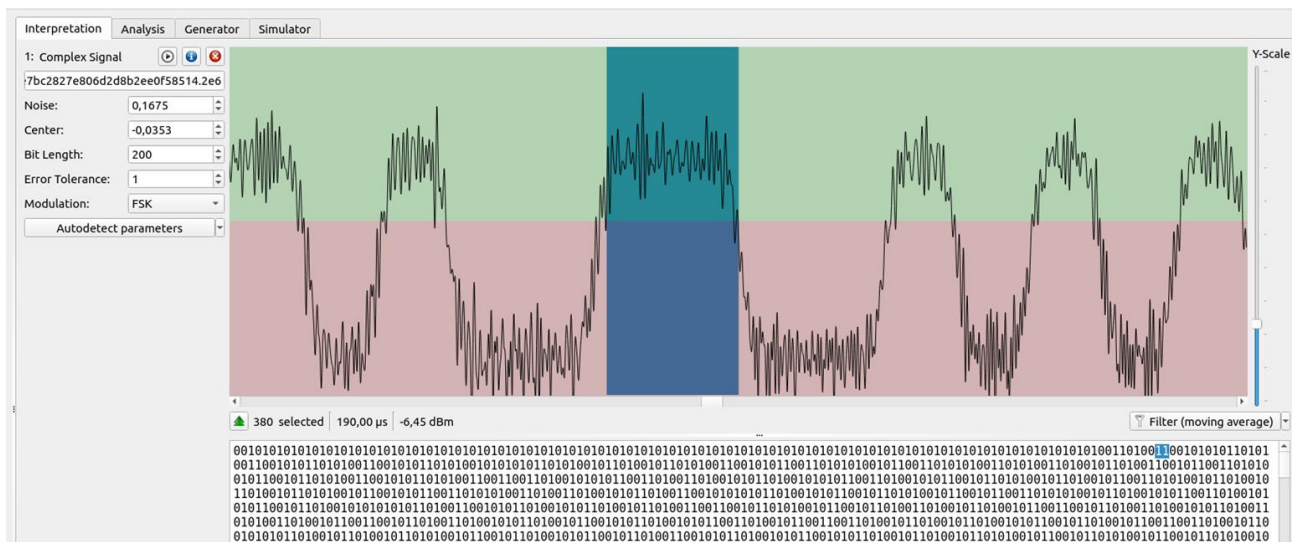
Pour en avoir le cœur net le fichier est ouvert dans audacity en mode Raw Mono affichage spectrogramme.



On distingue un signal carré répété en 17 rafales, il s'agit bien d'un fichier radio.

L'outil d'analyse Universal Radio Hacker est particulièrement adapté pour ce genre d'analyse radio, car il s'agit d'un message numérique à retrouver dans un signal radio.

Le plus compliqué étant de trouver les bons paramètres pour que les états haut et bas correspondent correctement à des 1 et des 0.



Il faut dans un premier temps régler les seuils de détection des états hauts et bas (zone verte et rose sur l'image).

Dans un second temps il faut régler la valeur « Bit Length » afin de définir combien de temps dure un bit.

Pour cela, on identifie visuellement que les états haut et bas sont soit courts soit longs et les temps longs correspondent maximum à 2 temps courts.

On ajuste Bit Length jusqu'à ce que la sélection de 11 dans le champ texte sélectionne bien 2 états hauts successif sur le graph.

Ce signal carré ressemble fortement à du code Manchester utilisé fréquemment en transmission de signaux numériques.

```
01100101101010011001010110101001100110011010010101100110100110100101011001101001010
1100101101010010110100101100110101001011010010110100101101010010110010101100110101010011010011010
01010110100110010101011010010101100101101010010110010110011010101001011010010101100110100101010
1100101101001010101010110100110010101101001010
```

En copiant le code manchester d'une des 17 rafale dans un décodeur en ligne (<https://eleif.net/manchester.html>), on obtient le code binaire suivant :

```
0110100001110100011101000111000001110011001110100010111100101110111011001101001011100100111010001
110101011000010110110001100001011000100111001100101110011001110010001011101101100011010000011
0001001110010010111100110001011000010011000000110100011000110011010100111001001101100110010100110
1100011011101100101001101100011001000110001011001010110011001100010011001010110010000110011001110
0100110001001101000110001000110011001110010011100100111000011001100011000100111001001011100110100
0011101000110110101101100
```

Ce qui converti en ASCII donne l'URL suivante:

<https://virtualabs.fr/lh19/1a04c596e67e621efbed3914b3998f19.html>

Step 5 : Stalking et Social engineering

L'énoncé de cette étape :

Already there when I started to hack, it is nowadays gone and will never come back.
This platform was once among the bests, providing an early hacking contest and amazing tasks to solve without a rest.

Show me your abilities by telling me what this platform is.
But don't yell it here, just whisper it to my ear,
show me what it did look like, show me what I did look
like at this time, and the black badge will no longer be mine.

Virtualabs

Première étape retrouver la plateforme disparue.

Il fallait lire la section « A propos » sur la page de Virtu : <https://virtualabs.fr/author/damien-cauquil.html>

La plateforme mentionnée plus haut était « Espionet.com » ancêtre de root-me.org.

« il découvrit aussi le site Espionet (le root-me de l'époque). »

Un dump du site est disponible sur archive.org



Pour la seconde partie ça se corse, car il s'agit de retrouver une photo de Virtu datant de cette époque soit à peu près entre 2002 et 2004.

Premier réflexe, regarder le Viadeo de Virtu (<http://fr.viadeo.com/fr/profile/damien.cauquil>) pour voir son parcours scolaire et chercher des photos de classe sur copains d'avant mais sans succès.

Se rabattre ensuite sur archive.org pour retrouver des reliques d'infos sur Hackerz Voice et autres sites et assos par lesquelles Virtu est passé.

Au fil des pérégrinations, on peut retrouver des vieux albums photo des Nuits Du Hack passées.

<https://get.google.com/albumarchive/106467546667993439289>

La plus vieille image que trouvée dans ces albums date de la NDH 2006.



Et voilà !!!! challenge résolu mais quelqu'un l'a déjà résolu avant moi.

Merci à Virtu pour ce challenge très intéressant avec des étapes très variées.

Merci à Gatewatcher pour m'avoir fourni un Micro:bit et du temps pour ce challenge.