Solution du challenge SSTIC 2016

Cécile Berillon (cecile.berillon@gmail.com)

12 mai 2016

Table des matières

1 Introduction

Ce document présente ma solution au challenge du SSTIC2016. Il consistait cette année en un jeu de rôle à trois étapes. A chacune de ces trois étapes du jeu, des personnages permettent de télécharger des fichiers zip contenant des challenges valant un ou deux points, sachant qu'il en faut deux pour valider une étape.

Le fichier initial, challenge.pcap, correspond à une capture réseau d'une requête HTTP pour télécharger le fichier challenge.zip. On l'ouvre à l'aide de *Wireshark*, on clique File, Export Objects, HTTP puis on télécharge le zip qui contient le jeu de rôle auquel on accède dans un navigateur Web.

2 Niveau 1

On recupère 4 fichiers :
— calc.zip
— crosswords.zip
— radio.zip
— SOS-Fant0me.zip

2.1 Martine retourne au lycée

En décompressant calc.zip, on récupère le fichier SSTIC16.8xp.

```
$ file SSTIC16.8xp
SSTIC16.8xp: TI-83+ Graphing Calculator (program)
```

La commande *file* indique qu'il s'agit d'un programme pour calculatrice graphique TI-83+. Le site *http ://www.cemetech.net/sc/* permet à partir du code compilé de récupérer le code source en TI-Basic.

Le programme prend au Label 1 un code en entrée, il effectue une série de calculs dessus puis vérifie au Label 19 si le code était le bon à l'aide du résultat de ces calculs. Dans ce cas, le générateur de nombres pseudo-aléatoire de la calculatrice est utilisé pour calculer la clé et l'imprimer à l'écran.

2.1.1 Brute-force du code entré

Pour brute-forcer le code à entrer, on retranscrit le code source TI-Basic en Python et on teste tous les entiers jusqu'à obtenir validation par le programme.

On peut remarquer que des chaînes de caractères de '1' et '0' sont utilisées pour effectuer des opérations bit à bit. Par exemple, le Label 4 :

```
Lbl 4
" "->Str3
For(I,1,length(Str1)
    If "1"=sub(Str1,length(Str1)-I+1,1) xor "1"=sub(Str2,length(Str1)-I+1,1):Then
        "1"+Str3->Str3
        Else
        "0"+Str3->Str3
    End
End
sub(Str3,1,length(Str3)-1)->Str3
Goto 0
```

fait un simple *xor* entre les entiers representés par Str1 et Str2 et le stocke dans Str3. Pour accélérer le brute-force du code entré, on peut retranscrire le programme en effectuant les calculs directement sur les entiers plutôt que sur les chaînes des caractères de leur transcription binaire.

Un résultat positif est obtenu pour le code d'entrée : **89594902**.

2.1.2 Obtention de la clé

Pour récupérer la clé, on utilise un émulateur de TI 83+ dans lequel on télécharge le .8xp, on tape le code trouve précedemment et on récupère la première clé !

FIGURE 1 – Exécution du programme dans l'émulateur

2.2 Martine contre les fantômes

On décompresse le deuxième challenge à un point : SOS-Fant0me.zip, qui contient une capture réseau. En l'ouvrant une nouvelle fois dans *Wireshark*, on remarque la présence des caractères **Gh0st** dans les paquets TCP.

Une recherche sur ce mot-clé permet de tomber sur le Remote Administration Tool Gh0st qui communique à l'aide du nombre magique *Gh0st* puis envoie les données sous forme compressée à l'aide de l'algorithme zlib.

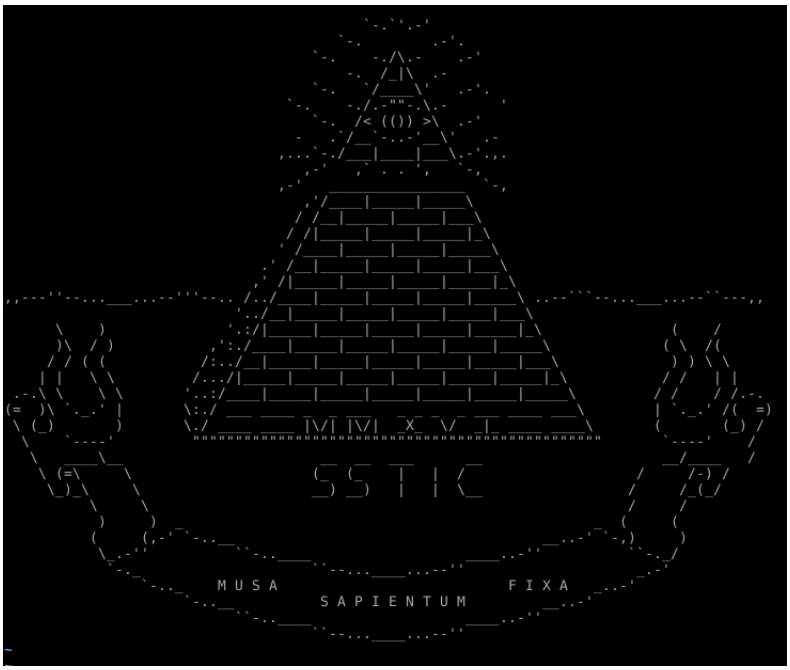
Le module gh0st_decode de l'outil *ChopShop* developpé par le MITRE permet de récupérer une succession de commandes/réponses de la capture réseau à l'aide de la commade :

```
$ chopshop -s extracts/ -F extracts/%N.txt -f SOS-Fant0me.pcap "gh0st_decode -s"
```

On obtient ainsi :

- Le message : *Salut ! Comme pis voici la clef pour le stage 1 ! Le mot de passe de l'archive reste celui convenu ensemble.*
- Le mot de passe tapé pour l'archive *sstic2016-stage1-solution.zip* : '**Cyb3rSSTIC_2016**'
- Un parcours progressif du système de fichier à partir de C :\
- Le téléchargement du fichier texte C :\Users\sstic\Documents\Challenge SSTIC 2016\how_to_rule_the_world.txt
- Le téléchargement du fichier video C :\Users\sstic\Documents\Challenge SSTIC 2016\visio_stage2.mp4
- Le téléchargement de l'archive C :\Users\sstic\Documents\Challenge SSTIC 2016\Stage 1\sstic2016-stage1-solution.zip

Le fichier texte how_to_rule_the_world.txt contient l'image :



La video visio_stage2.mp4 est un pur rickroll :)

En décompressant l'archive zip à l'aide du mot de passe trouvé dans les échanges, on obtient un fichier solution.txt qui contient la deuxieme clé : **368BE8C1CC7CC70C2245030934301C15** !

3 Niveau 2

On récupère 3 fichiers :
— foo.zip
— huge.zip
— loader.zip

On en profite pour récupérer les lunettes et le portable égarés dans la grotte et les rendre à son propriétaire en bon citoyen.

3.1 Martine rencontre un géant

En décompressant l'archive huge.zip, on récupère une autre archive huge.tar qui contient elle-même un fichier Huge de 128 To...

3.1.1 Mapping d'adresses entre l'archive et le fichier réel

En regardant de plus près ce que le fichier contient à l'aide de *ghex*, on peut voir qu'il s'agit d'un *Sparse file* et que Huge est en fait principalement composé d'octets nuls.

On trouve aussi les données suivantes qui correspondent au mapping entre les blocks de 4096 bytes de l'archive avec les blocks du fichier réel.

```
0
4096
1627791495168
4096
1627791519744
4096
11168083808256
4096
11168083820544
4096
25297854992384
4096
25297855041536
4096
27126397538304
4096
33981332525056
4096
33981332570112
8192
47446612774912
4096
47446612832256
4096
55346731216896
4096
64713413758976
4096
64713413775360
4096
65249501974528
4096
88943149977600
4096
88943150039040
4096
91853110185984
4096
91853110214656
4096
99931956908032
4096
99931956936704
4096
128019591467008
4096
128019591507968
4096
128574140710912
4096
```

3.1.2 Mapping d'adresses entre le fichier réel et les adresses virtuelles de l'ELF

On récupère la partie de l'archive située après les headers, elle commence par le nombre magique *ELF* ce qui laisse deviner la nature du fichier.

```
$ readelf --headers elf
ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x51466a42e705
  Start of program headers:          64 (bytes into file)
  Start of section headers:          0 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           56 (bytes)
  Number of program headers:         3
  Size of section headers:           0 (bytes)
  Number of section headers:         0
  Section header string table index: 0

There are no sections in this file.

Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000001000 0x00002b0000000000 0x00002b0000000000
                 0x00001ef000000000 0x00001ef000000000  R E    1000
  LOAD           0x00002affffe1000  0x000049f000000000 0x000049f000000000
                 0x0000161000000000 0x0000161000000000  R E    1000
  LOAD           0x000049effffe1000 0x0000000000020000 0x0000000000020000
                 0x00002affffe0000  0x00002affffe0000   R E    1000
```

Les *Program Headers* donnent la correspondance entre les adresses dans le Huge complet (incluant tous les zéros) et les adresses virtuelles utilisées dans l'exécutable.

Avec toutes ces infos, on peut coder un script python permettent d'obtenir la correspondance entre les adresses virtuelles utilisées dans l'ELF et les adresses dans le Huge privé de ses zéros, ce qui va nous permettre de le désassembler.

3.1.3 Désassemblage de l'ELF

Dans la plupart des cas, sauf un developpé après, les octets nuls du fichier ne servent que de *NOP sled* permettant d'atteindre la partie suivante du code assembleur.

On récupère dans les headers de la section précèdente l'entrypoint : *0x51466a42e705* et on commence a désassembler.

```
    and     rsp,0xfffffffffffffc00
    mov     rbp,rsp
    xor     rax,rax
    inc     rax
    xor     rdi,rdi
    inc     rdi
    movabs  rsi,0x5a48156509b7
    mov     edx,0x1b
    syscall
```

Un premier appel système pour imprimer sur stdout la chaîne de caractère : "*Please enter the password :*"

```
    xor     rax,rax
    xor     rdi,rdi
    mov     rsi,rbp
    mov     edx,0x400
    syscall
```

Un deuxième appel système pour récupérer le mot de passe entré.

```
    mov     rax,rsp
    mov     rdi,rsp
    mov     rsi,rsp
    movabs  r15,0x10f338cf000c
    call    r15
```

Un appel à la fonction suivante est fait :

```
movzx   ebx,BYTE PTR [rsi]
sub     bl,0x30
jb      0x5f
cmp     bl,0xa
jb      0x24
sub     bl,0x11
jb      0x5f
cmp     bl,0x6
jb      0x21
sub     bl,0x20
jb      0x5f
cmp     bl,0x6
jae     0x5f
add     bl,0xa
inc     rsi
shl     ebx,0x8
mov     bl,BYTE PTR [rsi]
sub     bl,0x30
jb      0x5f
cmp     bl,0xa
jb      0x4d
sub     bl,0x11
jb      0x5f
cmp     bl,0x6
jb      0x4a
sub     bl,0x20
jb      0x5f
cmp     bl,0x6
jae     0x5f
add     bl,0xa
inc     rsi
shl     bl,0x4
shr     bx,0x4
mov     BYTE PTR [rdi],bl
inc     rdi
loop    0x0
ret
```

Cette fonction prend les caractères du mot de passe entré deux par deux, vérifie individuellement qu'il s'agit de caractères hexadecimaux (majuscules ou minuscules) puis les transforme en octet. '4B' devient ainsi '\x4B', ce qui équivaut à un textsf.decode('hex') en Python.

```
movabs rbx,0x43abdb4a000c
jmp    rbx
```

Hop, hop, fonction suivante..

```
cmp    BYTE PTR [rsp],0x29
jne    0x1b
movabs rbx,0x4a170682000c
jmp    rbx
```

.. vérification que le premier octet est 0x29 ..

```
cmp    WORD PTR [rsp+0x2],0xd17e
movabs rbx,0x6f4b0e0000c
movabs r14,0x4a170682ee02
cmovne rbx,r14
jmp    rbx
```

.. vérification que les troisième et quatrième octets sont 0x7e et 0xd1 ..

```
cmp    BYTE PTR [rsp+0xb],0x8c
movabs rbx,0x6f4b0e0f38f
movabs r14,0x49e7e541000c
cmove  rbx,r14
jmp    rbx
```

.. vérification que le douxième octet est 0x8c ..

```
push   rax
lea    rax,[rsp+0x10b]
movzx  rbx,BYTE PTR [rsp+0x9]
mov    BYTE PTR [rax],0x0
lea    rcx,[rip+0x6]
lea    rcx,[rcx+rbx*2]
jmp    rcx

{{  add rax, al) }}

cmp    BYTE PTR [rax],0x65
movabs rbx,0x352845ab000c
movabs r14,0x49e7e541c056
cmovne rbx,r14
pop    rax
jmp    rbx
```

.. ici les octets nuls ne sont pas transparents puisque rax = 0x7fffffffd103 et donc al n'est pas nul. En fonction rbx, le nombre d'addition de al n'est pas le même. Cela donne le deuxieme octet qui vaut 0x89 ..

```
lea     rbx,[rip+0x0]
xor     ebx,DWORD PTR [rsp+0xc]
cmp     ebx,0xa9b00f5c
movabs  rbx,0x352845ab3bf9
movabs  r14,0x59cb440c000c
cmove   rbx,r14
jmp     rbx
```

.. rbx vaut 0x352845ab3bd5, on effectue un *xor* entre ebx et 0xa9b00f5c pour obtenir les quatre derniers octets : 0x89, 0x34, 0x1b et 0xec ..

```
movabs  rbx,0x59cbc8cc0b83
mov     rcx,QWORD PTR [rip+0x19]
push    rax
mov     eax,DWORD PTR [rsp+0x10]
xor     rax,rbx
movabs  rbx,0x2a7ee24a000c
cmpxchg rcx,rbx
pop     rax
jmp     rcx
```

.. encore un xor ici entre ecx et ebx avec rbx=0x59cbc8cc0b83 et ecx=0x59cb440c4556 pour obtenir que les octets neuf a douze valent 0xd5, 0x4e, 0xc0 et 0x8c (dernier octet déjà obtenu précedemment par ailleurs) ..

```
push    rax
lea     rdi,[rsp+0x18]
lea     rsi,[rip+0x42]
mov     ecx,0xa
rep movs BYTE PTR es:[rdi],BYTE PTR ds:[rsi]
mov     ebx,DWORD PTR [rsp+0xc]
xor     DWORD PTR [rsp+0x1c],ebx
fclex
fld     TBYTE PTR [rsp+0x18]
fld     st(0)
fcos
fcompp
fstsw   ax
and     ax,0xffdf
cmp     ax,0x4000
movabs  rbx,0x99a3805000c
movabs  r14,0x2a7ee24aae7e
cmovne  rbx,r14
pop     rax
jmp     rbx
```

.. et enfin les 4 octets manquants, présents dans ebx, sont déterminés tels que le float, composé a partir d'une constante donnée dont une partie a subi un xor avec ces octets, vérifie l'équation cos(x) = x.

On a ainsi obtenu le password validant : **29897ed1d4d68c99d54ec08c89341bec**

3.1.4 "Réparation" de l'ELF

Au lieu de finir de désassembler l'ELF pour obtenir la clé voulue, on peut le 'réparer'. On modifie directement l'ELF dans *ghex*.

La première étape est de modifier les *Program headers* de l'ELF pour mapper correctement la mémoire telle qu'elle est dans le fichier sans les zéros. En écrivant un script python qui fait l'inverse de celui précedent, on peut mapper les pages du fichier aux adresses virtuelles du fichier originel. On obtient ainsi les *Program headers* suivants :

```
Program Headers:
  Type           Offset             VirtAddr           PhysAddr
                 FileSiz            MemSiz              Flags  Align
  LOAD           0x0000000000001000 0x00002c7affef0000 0x00002c7affef0000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000002000 0x00002c7affef6000 0x00002c7affef6000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000003000 0x0000352845ab0000 0x0000352845ab0000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000004000 0x0000352845ab3000 0x0000352845ab3000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000005000 0x000042021da90000 0x000042021da90000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000006000 0x000042021da9c000 0x000042021da9c000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000007000 0x000043abdb4a0000 0x000043abdb4a0000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000008000 0x000049e7e5410000 0x000049e7e5410000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000009000 0x000049e7e541b000 0x000049e7e541b000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x000000000000a000 0x000049e7e541c000 0x000049e7e541c000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x000000000000b000 0x00004a1706820000 0x00004a1706820000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x000000000000c000 0x00004a170682e000 0x00004a170682e000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x000000000000d000 0x000051466a42e000 0x000051466a42e000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x000000000000e000 0x000059cb440c0000 0x000059cb440c0000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x000000000000f000 0x000059cb440c4000 0x000059cb440c4000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000010000 0x00005a4815650000 0x00005a4815650000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000011000 0x000006f4b0e00000 0x000006f4b0e00000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000012000 0x000006f4b0e0f000 0x000006f4b0e0f000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000013000 0x0000099a38050000 0x0000099a38050000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000014000 0x0000099a38057000 0x0000099a38057000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000015000 0x000010f338cf0000 0x000010f338cf0000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000016000 0x000010f338cf7000 0x000010f338cf7000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000017000 0x00002a7ee24a0000 0x00002a7ee24a0000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000018000 0x00002a7ee24aa000 0x00002a7ee24aa000
                 0x0000000000001000 0x0000000000001000  R E    1000
  LOAD           0x0000000000019000 0x00002affffff000 0x00002affffff000
                 0x0000000000001000 0x0000000000001000  R E    1000
```

On rajoute aussi des jmp pour compenser les NOP sled et éviter des Segmentation Fault et on modifie le cmp pour la partie où le NOP sled n'en est pas un et où la comparaison sera fausse pour que l'exécution du programme continue.

En entrant le mot de passe trouvé au-dessus, on obtient :

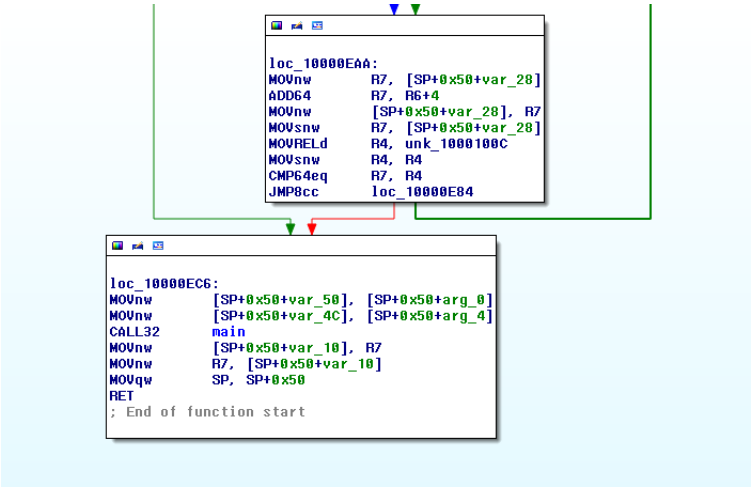
"The key is : **E574B514667F6AB2D83047BB871A54F5**"

3.2 Martine démarre

On décompresse foo.zip pour obtenir foo.efi.

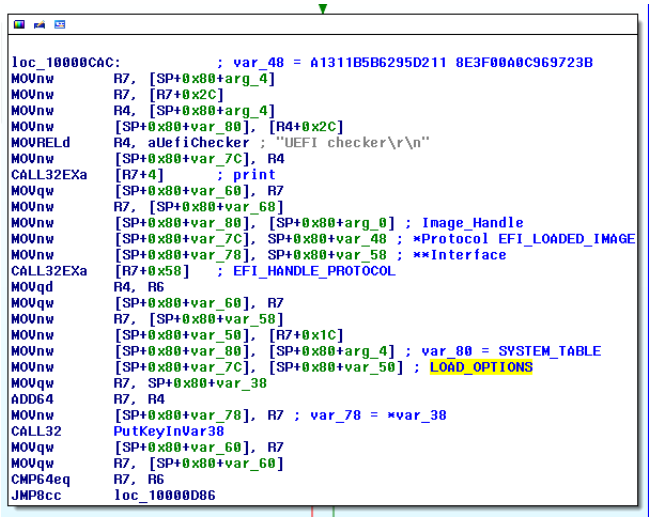
```
$ file foo.efi
foo.efi: PE32 executable (DLL) (EFI application) EFI byte code, for MS Windows
```

Il s'agit donc d'une application en EFI byte code que l'on charge dans IDA.



On peut voir sur la capture d'écran la fonction main qui prend deux arguments en entrée.

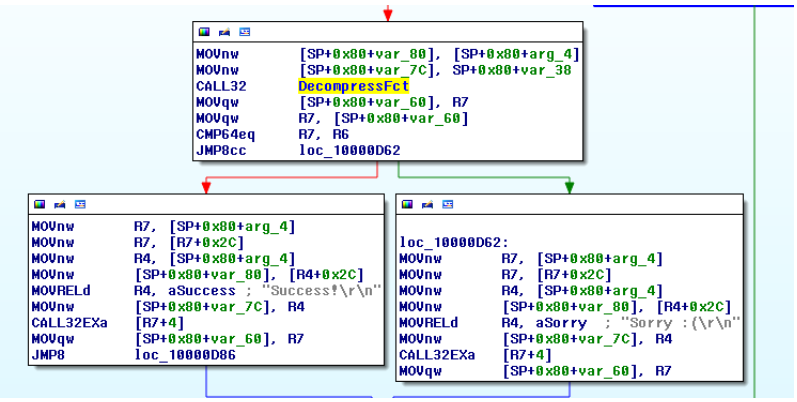
La fonction efi_main en C prend en argument le couple (EFI_HANDLE ImageHandle, EFI_SYSTEM_TABLE *SystemTable), qui correspond donc aux arguments ci-dessus.



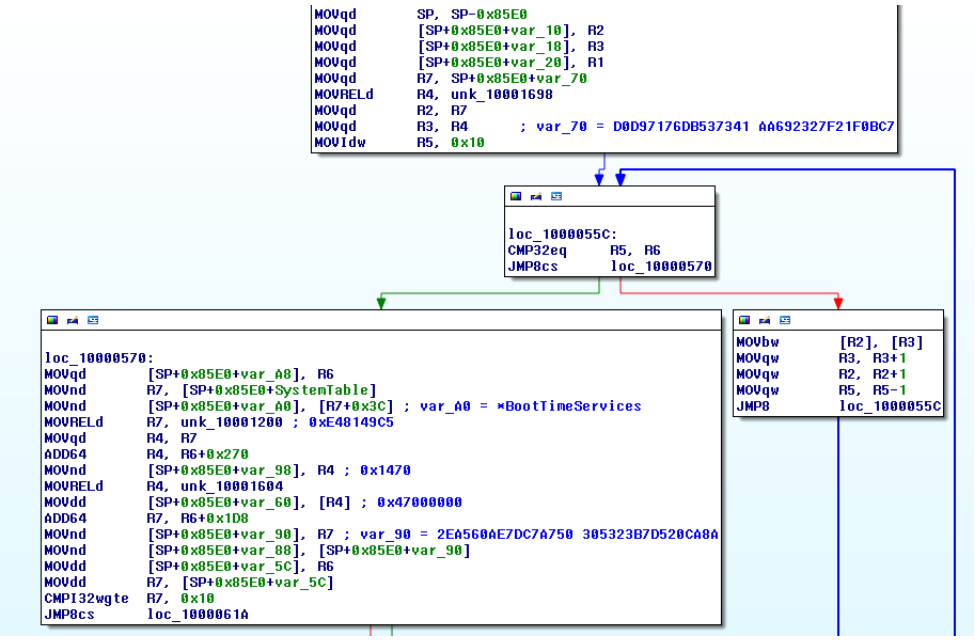
L'application charge ensuite l'image donnée en entrée a l'aide d'EFI_HANDLE_PROTOCOL et du GUID A1311B5B6295D211 8E3F00A0C969723B pointé par la variable 48.

La fonction PutKeyInVar38 prend ensuite une clé dans les options de cette image. Elle vérifie sa présence, qu'elle fait bien 32 caractères de long et qu'elle ne contient que des caractères hexadecimaux (en majuscule ou en minuscule). Elle la transforme ensuite de chaîne de caractères à une suite d'octets, comme pour *Huge*,

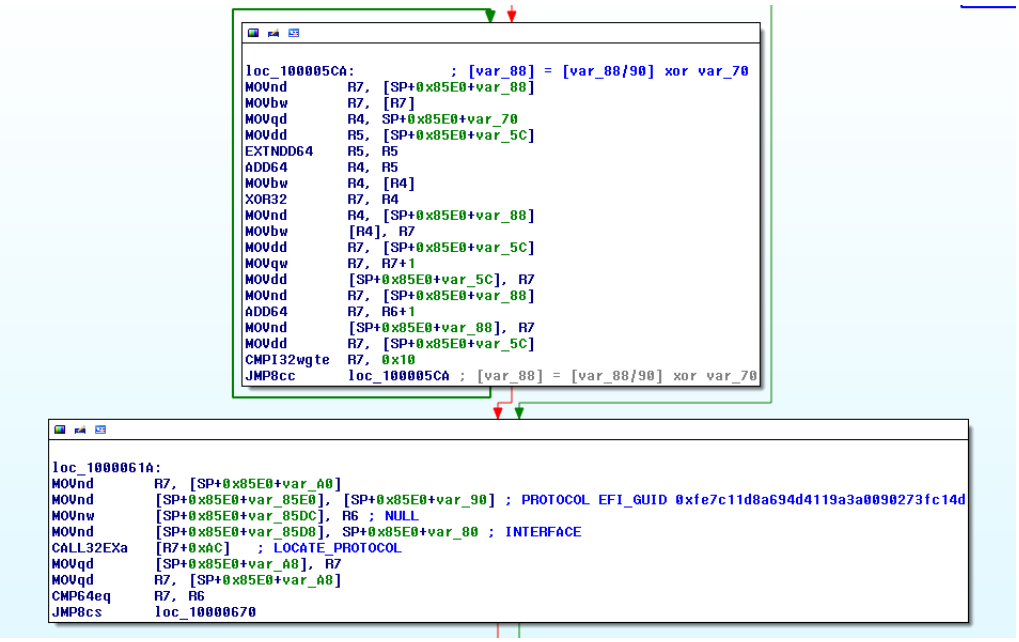
puis la stocke à l'aide de la variable 38.



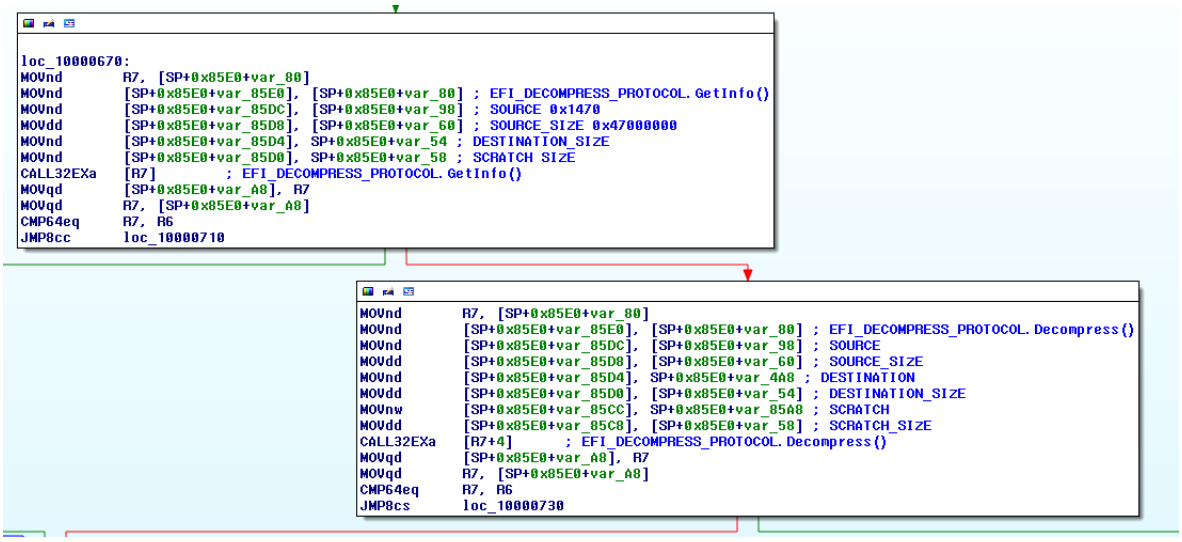
Une dernière fonction est appelée, qui, suivant la valeur retournée affiche *Success !* ou *Sorry :(*



L'application récupère deux valeurs : var_70 -> D0D97176DB537341AA692327F21F0BC7 et var_90/88 -> 2EA560AE7DC7A750305323B7D520CA8A



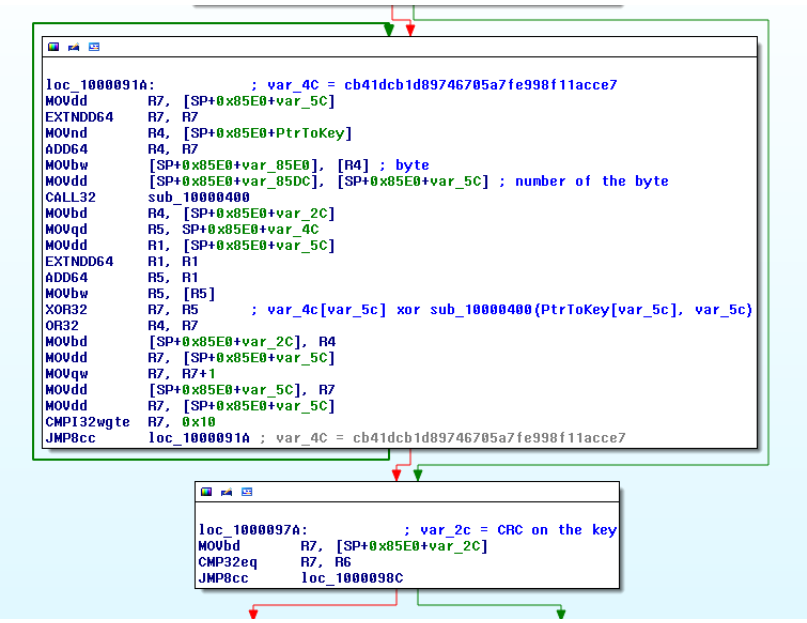
Le protocole localisé a pour GUID fe7c11d8a694d4119a3a0090273fc14d qui est le xor des deux valeurs précédentes, et correspond au protocole de décompression.



Les 0x47 octets situés en 0x1470 sont décompressés. Pour les récupérer, on utilise un petit script python et on obtient les données suivantes :

```
secret data: cb41dcb1d89746705a7fe998f11acce7
```

L'application vérifie qu'il s'agit bien des caractères hexadécimaux et les transforme une nouvelle fois sous forme d'octets.



Enfin, une fonction est appliquée sur chaque octet de la clé initale, l'octet résultant doit être égal à la donnée secrète trouvée juste avant.

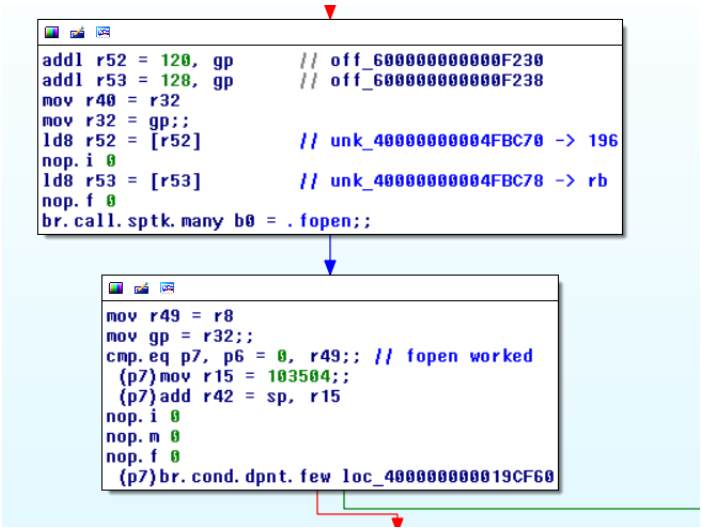
Cela permet, en inversant la fonction, d'obtenir la clé : **347d8c72720d6ec7a501583be0bccc0c**

4 Niveau 3

On récupère 4 fichiers :
— ring.zip
— strange.zip
— usb.zip
— video.zip

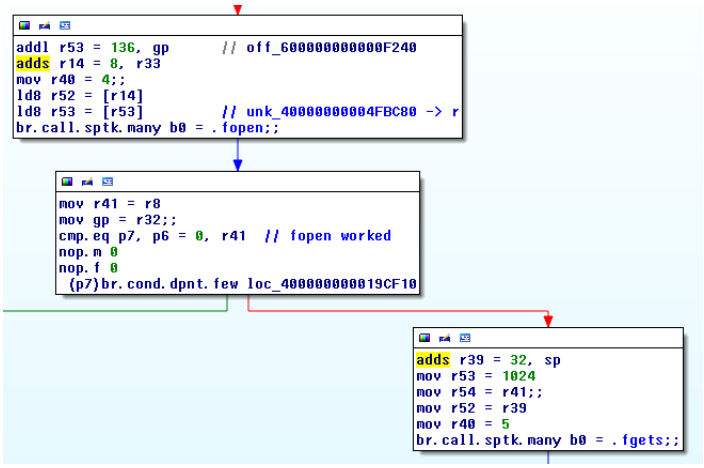
4.1 Martine apprend a lire

L'archive strange.zip renferme deux fichiers :
— Un fichier de données 196
— Un exécutable en Itanium 64 a.out
On commence par regarder a.out. Le programme en lui-même commence en 0x400000000019C700

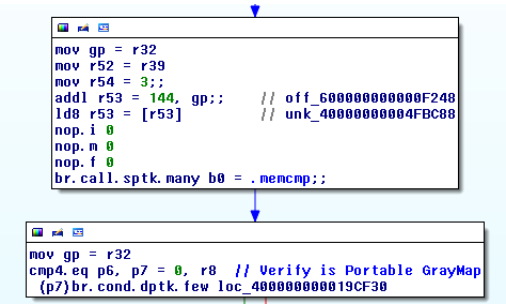


Tout d'abord, le fichier '196' est ouvert en mode 'rb', et le bon déroulement de l'opération est vérifié.

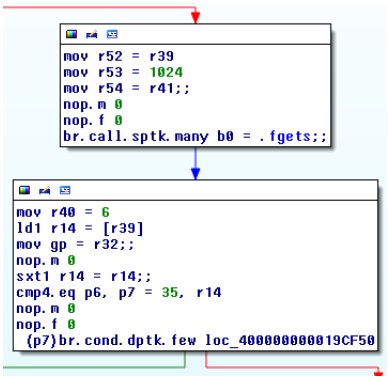
4.1.1 Chargement de l'header l'image



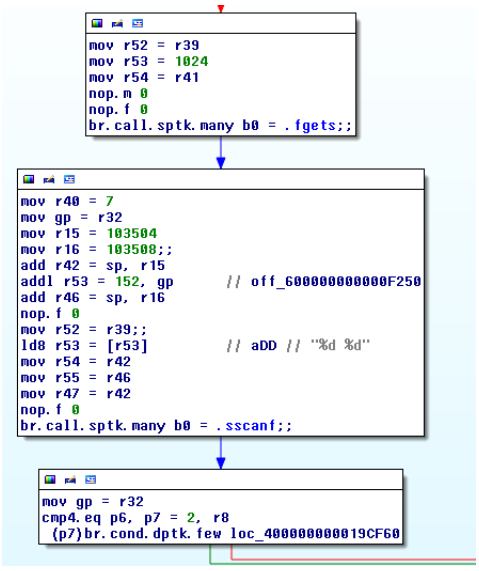
Le nom de fichier donné en argument est ouvert en mode 'r' et le bon déroulement de l'opération est vérifié. Ensuite, la première ligne (ou les 1024 premiers caractères) sont lus.



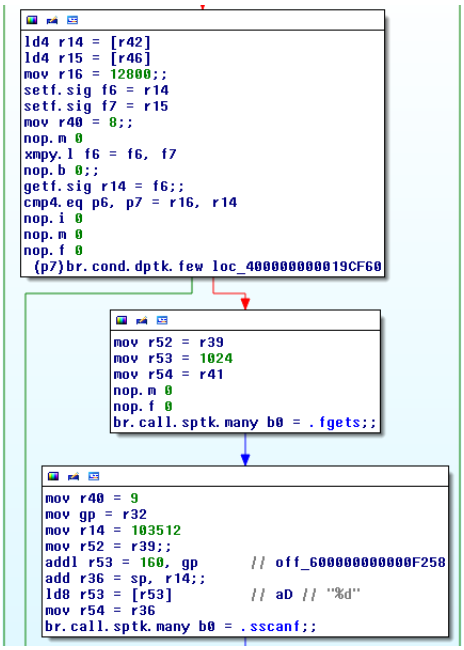
On verifie que les premiers correspondent a 'P2\xA'. Ces caractères sont associés au format Portable GrayMap, qui permet d'encoder une image pixel par pixel en niveaux de gris.



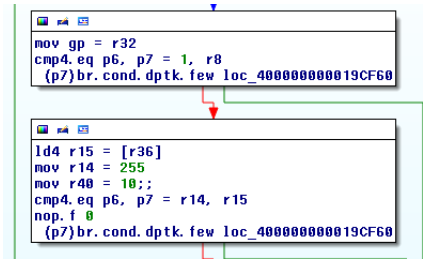
La ligne suivante est lue et il est verifié qu'elle commence par un #, donc qu'il s'agit d'une ligne de commentaire.



Sur la ligne suivante, sont extraits deux entiers, puis il est vérifié qu'ils sont bien extraits. Selon le format PGM, ils correspondent aux nombres de colonnes et de lignes dans l'image.

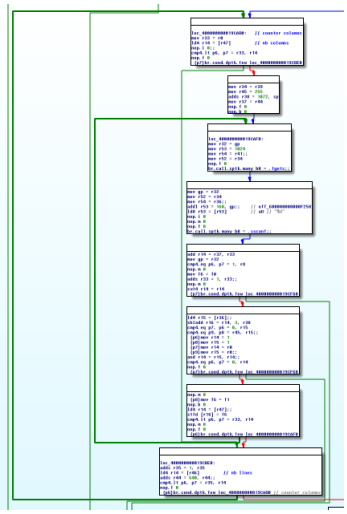


Le produit du nombre de lignes et de colonnes vaut 12800 et un entier est extrait de la ligne suivante.

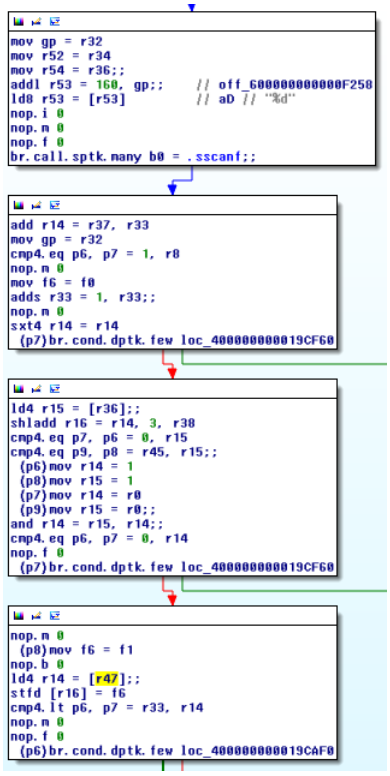


L'entier a bien été extrait et vaut 255. C'est le nombre qui correspondra au blanc sur l'image alors que 0 correspondra au noir.

4.1.2 Chargement de l'image



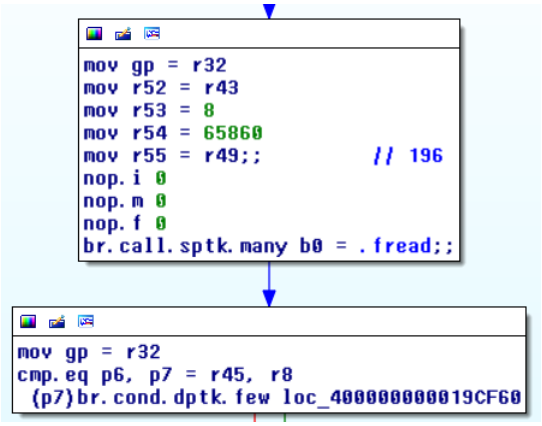
Il y a deux boucles : la boucle externe itère sur les lignes, la boucle interne itère sur les colonnes. A chaque boucle externe, 640 est ajouté à la variable d'offset, l'image a donc 640 colonnes et 20 lignes.



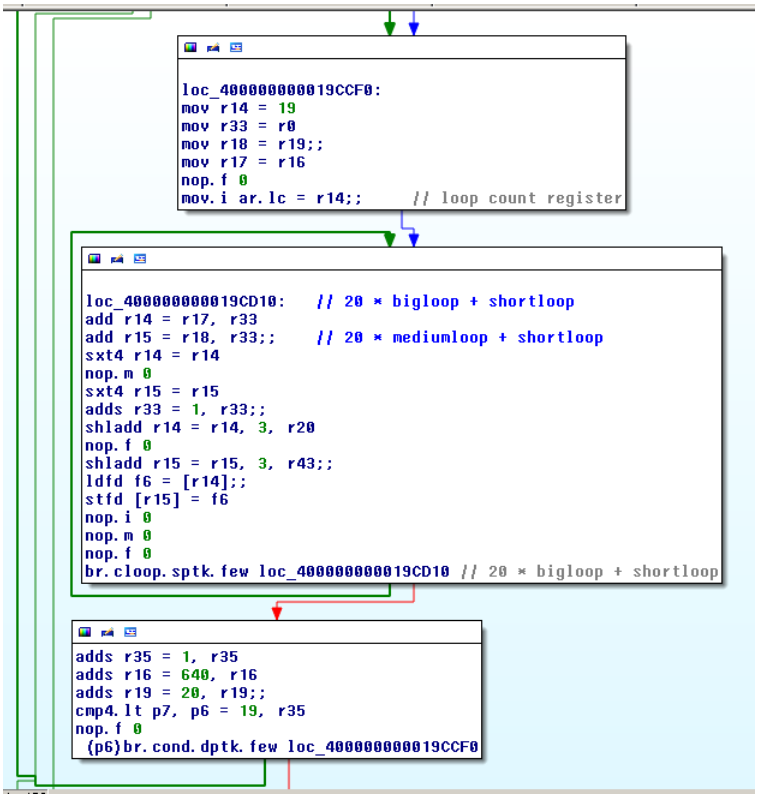
Pour chaque pixel, une nouvelle ligne est lue, un entier est extrait et le bon deroulement de l'extraction est vérifié. Ensuite, il est vérifié que cet entier vaut soit 255 soit 0. Si c'est bien le cas, un double float 1.0 si le pixel vaut 0, ou, 0.0 si le pixel vaut 255 est écrit en mémoire.

4.1.3 Chargement des caractères

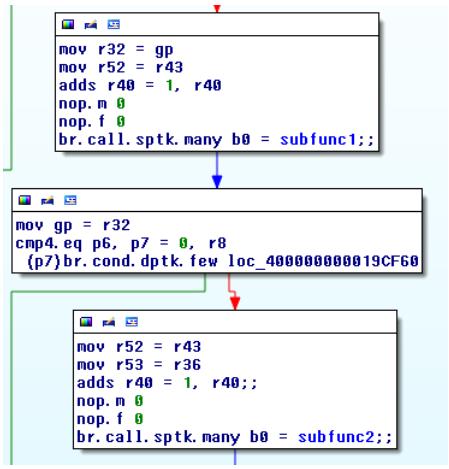
On retrouve ensuite une grande boucle externe, sur 32 itérations. Il s'agit du nombre de caractères hexadécimaux contenus dans la clé recherchée. On peut donc supposer qu'à chaque itération un caractère est verifié.



65860 * 8 octets sont chargés du fichier '196'.



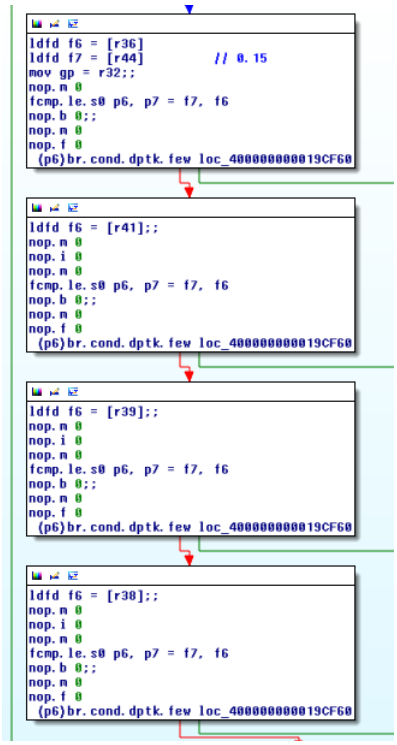
A chaque itération, un bloc carré de 20x20 est récupéré puis recopié sur le début des octets chargés du fichier '196'. L'image initiale est donc la concaténation de 32 blocs de 20x20, ce qui donne bien une image de taille 20x640.



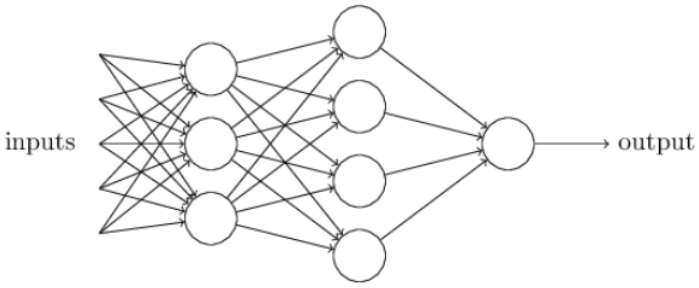
Deux sous-fonctions effectuent ensuite des tests sur ces caractères. La première vérifie notamment que les premières et dernières lignes et colonnes sont blanches.

4.1.4 Vérification des caractères

La deuxième sous-fonction correspond à un réseau de neurones convolutif à deux niveaux qui a pour fonction la reconnaissance de caractère sur une image. Le premier niveau prend en input les 400 pixels de l'image d'un caractère, leur applique un poids différent et en retire 160 valeurs. Cela correspond aux 160 premières sous-sous-fonctions. Puis en effectuant le même traitement avec des poids differents, de ces 160 valeurs le second niveau en retire 4, ce qui correspond à la dernière sous-sous-fonction.



Les 4 valeurs sortantes doivent être inférieures à 0.15 pour que le caractère de l'image entrée soit le bon.



Le modèle peut se résumer sur le schema ci-dessus, avec de gauche à droite :
— Les 'inputs' sont les 400 double float 0.0 ou 1.0 des pixels
— La première couche de neurones qui compte en réalité 160 neurones, après traitement des inputs
— La deuxième couche de 4 neurones, après traitement des neurones de la couche précédente
— Le neurone final qui correspond à la comparaison avec 0.15 et délivre l'output : *Était-ce le bon caractère sur l'image ?*

Pour calculer deux premiers niveaux, la même fonction est utilisée, avec des constantes différentes, une fonction sigmoide, de la forme :

$$\frac{1}{1+\exp(-\sum_j w_j x_j - b)}.$$

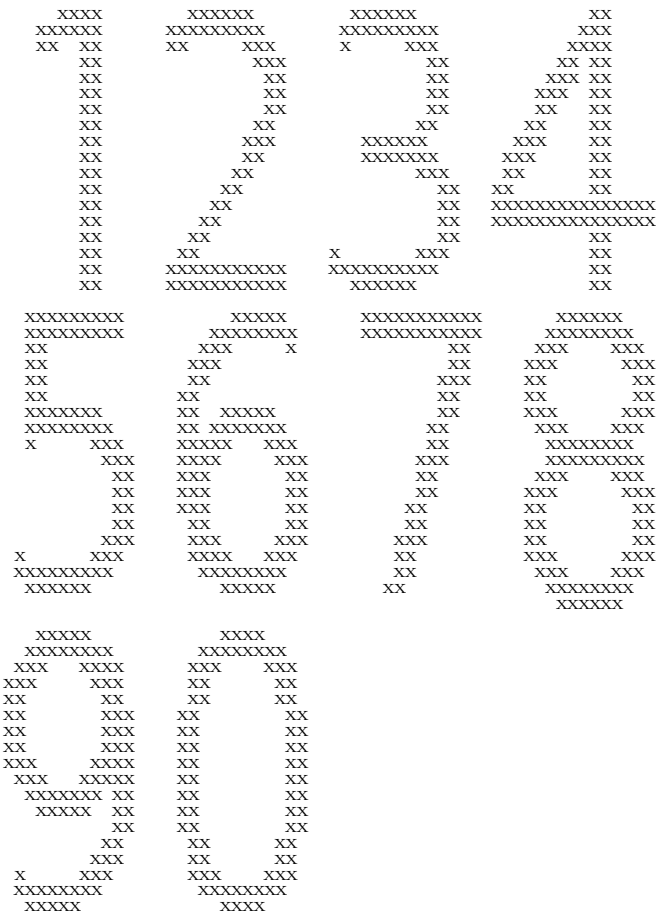
où les b et les w_j correspondent à l'offset et aux poids, et où les x_j correpondent aux entrées (soit les pixels, soit le premier niveau de neurones).

Les constantes utilisées sont issues des `65860 * 8` octets chargés de '`196`' à chaque itération de la boucle.

4.1.5 Brute-force de la clé

Pour récupérer la clé, il ne reste plus qu'à trouver des échantillons d'images à tester. En remarquant que les 3200 premiers octets chargés de '196' sont toujours effacés, on peut se demander ce qu'ils contiennent...

Il s'agit en fait de double float tous proches de 0 ou de 1. En les imprimant sous format 20x20, on obtient les 10 chiffres mais aucune lettre !



On retranscrit le réseau de neurone en Python, et on bruteforce la clé avec ces chiffres pour finalement obtenir : **234250384725082873357720855440 35**, clé valant deux points !

5 Niveau Final

On récupère finalement un fichier final.txt qui contient :

```
Coucou !

Tu as presque reussi le challenge !
I01p1 y'4qe3553 z41y : 8Y6d5j9Vy88HUGHfGSKsJvqA@ffgvp.bet
```

On peut se douter que l'adresse est toujours en sstic.org. En appliquant une subtitution monoalphabetique translatant le f sur le s et en laissant les chiffres intacts, correspondant au ROT13 :). On obtient finalement :

```
V01c1 l'4dr3553 m41l : 8L6q5w9Il88UHTUsTFXfWidN@sstic.org
```

qui annonce la fin du challenge !

Pour conclure, je tenais à remercier les concepteurs et organisateurs de ce challenge, ainsi que toutes les personnes qui ont dû me supporter pendant que je rageais dessus ;)

A Calc.zip - Brute-force du code entré

```
def zero():
    global S
    if S == 5:
        return five()
    elif S == 6:
        return six()
    elif S == 8:
        return eight()
    elif S == 9:
        return nine()
    elif S == 10:
        return ten()
    elif S == 11:
        return eleven()
    elif S == 13:
        return thirteen()
    elif S == 14:
        return fourteen()
    elif S == 16:
        return sixteen()
    elif S == 17:
        return seventeen()
    elif S == 18:
        return eighteen()
    elif S == 19:
        return nineteen()
    elif S == 21:
        return twentyone()
    elif S == 23:
        return twentythree()
    else:
        raise Exception()

def three():
    global A, Str1
    start = A * 8
    Str1 = rshift( Str1 & (int("11111111", 2) << (start)) , start)
    return zero()

def two():
    global A, Str1
    Str1 = A
    A = 0
    return zero()

def four():
    global Str3, Str1, Str2
    Str3 = Str1 ^ Str2
    return zero()

def seven():
    global Str1, Str2, A
    A = Str1
    return zero()
```

```
def rshift(val, n):
    s = 0x80000000 ^ 0xFFFFFFFF
    for i in range(0,n):
        val >>= 1
        val &= s
    return val

def fifteen():
    global Str1
    Str1 = rshift(Str1, 8)
    return zero()

def one(code):
    global Z, C, N, L1, A, S
    Z = code
    C = 4294967295
    N = 0
    L1 =
        [0,1996959894,3993919788,2567524794,124634137,1886057615,3915621685,2657392035,\

    249268274,2044508324,3772115230,2547177864,162941995,2125561021,3887607047,\
    2428444049,498536548,1789927666,4089016648,2227061214,450548861,1843258603,\
    4107580753,2211677639,325883990,1684777152,4251122042,2321926636,335633487,\
    1661365465,4195302755,2366115317,997073096,1281953886,3579855332,2724688242,\
    1006888145,1258607687,3524101629,2768942443,901097722,1119000684,3686517206,\
    2898065728,853044451,1172266101,3705015759,2882616665,651767980,1373503546,\
    3369554304,3218104598,565507253,1454621731,3485111705,3099436303,671266974,\
    1594198024,3322730930,2970347812,795835527,1483230225,3244367275,3060149565,\
    1994146192,31158534,2563907772,4023717930,1907459465,112637215,2680153253,\
    3904427059,2013776290,251722036,2517215374,3775830040,2137656763,141376813,\
    2439277719,3865271297,1802195444,476864866,2238001368,4066508878,1812370925,\
    453092731,2181625025,4111451223,1706088902,314042704,2344532202,4240017532,\
    1658658271,366619977,2362670323,4224994405,1303535960,984961486,2747007092,\
    3569037538,1256170817,1037604311,2765210733,3554079995,1131014506,879679996,\
    2909243462,3663771856,1141124467,855842277,2852801631,3708648649,1342533948,\
    654459306,3188396048,3373015174,1466479909,544179635,3110523913,3462522015,\
    1591671054,702138776,2966460450,3352799412,1504918807,783551873,3082640443,\
    3233442989,3988292384,2596254646,62317068,1957810842,3939845945,2647816111,\
    81470997,1943803523,3814918930,2489596804,225274430,2053790376,3826175755,\
    2466906013,167816743,2097651377,4027552580,2265490386,503444072,1762050814,\
    4150417245,2154129355,426522225,1852507879,4275313526,2312317920,282753626,\
    1742555852,4189708143,2394877945,397917763,1622183637,3604390888,2714866558,\
    953729732,1340076626,3518719985,2797360999,1068828381,1219638859,3624741850,\
    2936675148,906185462,1090812512,3747672003,2825379669,829329135,1181335161,\
    3412177804,3160834842,628085408,1382605366,3423369109,3138078467,570562233,\
    1426400815,3317316542,2998733608,733239954,1555261956,3268935591,3050360625,\
    752459403,1541320221,2607071920,3965973030,1969922972,40735498,2617837225,\
    3943577151,1913087877,83908371,2512341634,3803740692,2075208622,213261112,\
    2463272603,3855990285,2094854071,198958881,2262029012,4057260610,1759359992,\
    534414190,2176718541,4139329115,1873836001,414664567,2282248934,4279200368,\
    1711684554,285281116,2405801727,4167216745,1634467795,376229701,2685067896,\
    3608007406,1308918612,956543938,2808555105,3495958263,1231636301,1047427035,\
    2932959818,3654703836,1088359270,936918000,2847714899,3736837829,1202900863,\
    817233897,3183342108,3401237130,1404277552,615818150,3134207493,3453421203,\
    1423857449,601450431,3009837614,3294710456,1567103746,711928724,3020668471,\
    3272380065,1510334235,755167117]
    A = Z
```

```
    S = 5
    return two()

def five():
    global Str5, Str1, A, C, S
    Str5 = Str1
    A = C
    S = 11
    return two()

def eleven():
    global Str1, Str6
    Str6 = Str1
    return twelve()

def twelve():
    global Str1, Str6, A, S
    Str1 = Str6
    A = 0
    S = 13
    return three()

def thirteen():
    global Str7, Str1, Str5, A, N, S
    Str7 = Str1
    Str1 = Str5
    A = N
    S = 14
    return three()

def fourteen():
    global Str1, Str7, Str2, S
    Str1 = Str1
    Str2 = Str7
    S = 6
    return four()

def six():
    global Str1, Str3, S
    Str1 = Str3
    S = 8
    return seven()

def eight():
    global L1, A, S
    A = L1[A+1-1]
    S = 9
    return two()

def nine():
    global Str8, Str1, Str6, S
    Str8 = Str1
    Str1 = Str6
    S = 10
    return fifteen()

def ten():
    global Str2, Str8, S
```

```
    Str2 = Str8
    S = 16
    return four()

def sixteen():
    global Str1, Str3, S, N
    Str1 = Str3
    S = 11
    N += 1
    if N == 4:
        return seventeen()
    else:
        return zero()

def seventeen():
    global Str2, S
    Str2 = int("11111111111111111111111111111111", 2)
    S = 18
    return four()

def eighteen():
    global Str1, Str3, S
    Str1 = Str3
    S = 19
    return seven()

def nineteen():
    global A
    if A == 3298472535:
        return "success"
    else:
        return "fail"

def twentyone():
    return

def twentythree():
    global Str1
    print Str1
    return twentyone()

i = 0
while True:
    if (i % 100000 == 0):
        print i
    if one(i) != "fail":
        print "Code: " + str(i)
        break;
    i += 1
```

B S0S-Fantomes.zip - Commandes extraites

```
TOKEN: LOGIN: SSTIC-PC1: Windows 7 No service pack - Build: 7600 - Clock: 2500 Mhz
    - IP: 10.100.96.21 Webcam: yes
COMMAND: ACTIVED
COMMAND: SYSTEM
TOKEN: PSLIST
PID EXE      PROC NAME
272 smss.exe          \SystemRoot\System32\smss.exe
344 csrss.exe         \SystemRoot\System32\csrss.exe
352 SearchIndexer.exe        C:\Windows\system32\SearchIndexer.exe
392 winit.exe         winit.exe
404 winlogon.exe          winlogon.exe
492 services.exe          C:\Windows\system32\services.exe
500 lsass.exe         C:\Windows\system32\lsass.exe
508 lsm.exe       C:\Windows\system32\lsm.exe
608 svchost.exe       C:\Windows\system32\svchost.exe -k DcomLaunch
724 svchost.exe       C:\Windows\system32\svchost.exe -k RPCSS
816 svchost.exe       C:\Windows\system32\svchost.exe -k
    LocalServiceNetworkRestricted
860 svchost.exe       C:\Windows\system32\svchost.exe -k
    LocalSystemNetworkRestricted
884 svchost.exe       C:\Windows\system32\svchost.exe -k netsvcs
1048    svchost.exe       C:\Windows\system32\svchost.exe -k LocalService
1152    svchost.exe       C:\Windows\system32\svchost.exe -k NetworkService
1296    spoolsv.exe       spoolsv.exe
1536    dwm.exe       C:\Windows\system32\Dwm.exe
1552    explorer.exe          C:\Windows\Explorer.EXE
1641    conhost.exe       C:\Windows\system32\conhost.exe
1664    brasseur.exe          C:\tmp\brasseur.exe
1702    vlc.exe       C:\Program Files\vlc\vlc.exe
1778    iexplore.exe          iexplore.exe
1785    thunderbird.exe       C:\Program Files\Mozilla\Thunderbird\thunderbird.exe
1852    notepad.exe       notepad.exe
COMMAND: KEYBOARD
TOKEN: KEYBOARD START (OFFLINE)
COMMAND: NEXT
TOKEN: KEYBOARD DATA
[2016/02/27 - 23:14] New message: [SSTIC 2016/Challenge] Stage 1
TOKEN: KEYBOARD DATA
Sal
TOKEN: KEYBOARD DATA
ut ! C
TOKEN: KEYBOARD DATA
omme p
TOKEN: KEYBOARD DATA
is voic
TOKEN: KEYBOARD DATA
i la
TOKEN: KEYBOARD DATA
 clef p
TOKEN: KEYBOARD DATA
ou
TOKEN: KEYBOARD DATA
r l
TOKEN: KEYBOARD DATA
e sta
```

```
TOKEN: KEYBOARD DATA
ge 1
TOKEN: KEYBOARD DATA
 ! L
TOKEN: KEYBOARD DATA
e mo
TOKEN: KEYBOARD DATA
t de p
TOKEN: KEYBOARD DATA
ass
TOKEN: KEYBOARD DATA
e de
TOKEN: KEYBOARD DATA
 l'a
TOKEN: KEYBOARD DATA
rch
TOKEN: KEYBOARD DATA
ive res
TOKEN: KEYBOARD DATA
te ce
TOKEN: KEYBOARD DATA
ui conv
TOKEN: KEYBOARD DATA
enu en
TOKEN: KEYBOARD DATA
sem
TOKEN: KEYBOARD DATA
ble.
TOKEN: KEYBOARD DATA
[2016/02/27 - 23:15] sstic2016-stage1-solution.zip - Saisir mot de passe
TOKEN: KEYBOARD DATA
C
TOKEN: KEYBOARD DATA
y
TOKEN: KEYBOARD DATA
b
TOKEN: KEYBOARD DATA
3
TOKEN: KEYBOARD DATA
r
TOKEN: KEYBOARD DATA
S
TOKEN: KEYBOARD DATA
S
TOKEN: KEYBOARD DATA
T
TOKEN: KEYBOARD DATA
I
TOKEN: KEYBOARD DATA
C
TOKEN: KEYBOARD DATA
_
TOKEN: KEYBOARD DATA
2
TOKEN: KEYBOARD DATA
0
TOKEN: KEYBOARD DATA
1
```

```
TOKEN: KEYBOARD DATA
6
COMMAND: LIST DRIVE
TOKEN: DRIVE LIST
DRIVE   TOTAL   FREE    FILESYSTEM   DESCRIPTION
C   10228   8171    NTFS    Local Disk
D   0   0       CD Drive
COMMAND: LIST FILES (C:\)
TOKEN: FILE LIST
TYPE    NAME    SIZE    WRITE TIME
FILE    autoexec.bat    0   1438353600
FILE    config.sys  0   1438353600
DIR PerfLogs    0   1438353600
DIR Program Files   0   1438353600
DIR Users   0   1438353600
DIR Windows 0   1438353600
COMMAND: LIST FILES (C:\Users\)
TOKEN: FILE LIST
TYPE    NAME    SIZE    WRITE TIME
DIR Public  0   1438353600
DIR sstic   0   1438353600
COMMAND: LIST FILES (C:\Users\sstic\)
TOKEN: FILE LIST
TYPE    NAME    SIZE    WRITE TIME
FILE    Contacts    0   1438353600
FILE    Desktop 0   1438353600
FILE    Documents   0   1438353600
FILE    Downloads   0   1438353600
FILE    Favorites   0   1438353600
FILE    Links   0   1438353600
FILE    Music   0   1438353600
DIR Pictures    0   1438353600
DIR Saved Games 0   1438353600
DIR Searches    0   1438353600
DIR Videos  0   1438353600
COMMAND: LIST FILES (C:\Users\sstic\Documents\)
TOKEN: FILE LIST
TYPE    NAME    SIZE    WRITE TIME
FILE    Challenge SSTIC 2016    0   1438353600
COMMAND: LIST FILES (C:\Users\sstic\Documents\Challenge SSTIC 2016\)
TOKEN: FILE LIST
TYPE    NAME    SIZE    WRITE TIME
FILE    how_to_rule_the_world.txt   2759    1451606340
DIR Stage 1 0   1453797600
DIR visio_stage2.mp4    1374020 1454960220
COMMAND: DOWN FILES (C:\Users\sstic\Documents\Challenge SSTIC 2016\
    how_to_rule_the_world.txt)
C__Users_sstic_Documents_Challenge SSTIC 2016_how_to_rule_the_world.txt
TOKEN: FILE SIZE (C:\Users\sstic\Documents\Challenge SSTIC 2016\
    how_to_rule_the_world.txt: 2759)
COMMAND: CONTINUE
COMMAND: FILE DATA (2759)
Wrote 2759 of 2759 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _how_to_rule_the_world.txt
COMMAND: CONTINUE
COMMAND: DOWN FILES (C:\Users\sstic\Documents\Challenge SSTIC 2016\visio_stage2.
    mp4)
C__Users_sstic_Documents_Challenge SSTIC 2016_visio_stage2.mp4
```

```
TOKEN: FILE SIZE (C:\Users\sstic\Documents\Challenge SSTIC 2016\visio_stage2.mp4:
    193968)
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 8183 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016_visio_stage2
    .mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 16366 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 24549 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 32732 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 40915 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 49098 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 57281 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 65464 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 73647 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 81830 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 90013 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 98196 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 106379 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 114562 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
```

```
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 122745 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 130928 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 139111 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 147294 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 155477 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 163660 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 171843 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 180026 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (8183)
Wrote 188209 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: FILE DATA (5759)
Wrote 193968 of 193968 to C__Users_sstic_Documents_Challenge SSTIC 2016
    _visio_stage2.mp4
COMMAND: CONTINUE
COMMAND: LIST FILES (C:\Users\sstic\Documents\Challenge SSTIC 2016\Stage 1\)
TOKEN: FILE LIST
TYPE    NAME    SIZE    WRITE TIME
DIR sstic2016-stage1-solution.zip    207 1456614900
COMMAND: DOWN FILES (C:\Users\sstic\Documents\Challenge SSTIC 2016\Stage 1\
    sstic2016-stage1-solution.zip)
C__Users_sstic_Documents_Challenge SSTIC 2016_Stage 1_sstic2016-stage1-solution.
    zip
TOKEN: FILE SIZE (C:\Users\sstic\Documents\Challenge SSTIC 2016\Stage 1\sstic2016-
    stage1-solution.zip: 234)
COMMAND: CONTINUE
COMMAND: FILE DATA (234)
Wrote 234 of 234 to C__Users_sstic_Documents_Challenge SSTIC 2016_Stage 1
    _sstic2016-stage1-solution.zip
COMMAND: CONTINUE
COMMAND: LIST DRIVE
```

C Huge.zip - Mappings entre fichier compressé et les adresses virtuelles

```
#!/bin/python

import sys

# Sparse File header
offsets = [ 0, 1627791495168, 1627791519744, 11168083808256, 11168083820544,
    25297854992384, 25297855041536, 27126397538304, 33981332525056,
    33981332570112, 33981332570112 + 4096,47446612774912, 47446612832256,
    55346731216896, 64713413758976, 64713413775360, 65249501974528,
    88943149977600, 88943150039040, 91853110185984, 91853110214656,
    99931956908032, 99931956936704, 128019591467008, 128019591507968,
    128574140710912 ]

# Program headers
offset1 = int(0x0000000000001000)
virtaddr1 = int(0x00002b0000000000)
size1 = int(0x00001ef000000000)

offset2 = int(0x00002affffe1000)
virtaddr2 = int(0x000049f000000000)
size2 = int(0x0000161000000000)

offset3 = int(0x000049effffe1000)
virtaddr3 = int(0x0000000000020000)
size3 = int(0x00002affffe0000)

def mapVirtAddress(address):
    if address >= offset1 and address < offset1 + size1:
        return address - offset1 + virtaddr1
    if address >= offset2 and address < offset2 + size2:
        return address - offset2 + virtaddr2
    if address >= offset3 and address < offset3 + size3:
        return address - offset3 + virtaddr3
    raise Exception("Invalid Address")

def mapSparsefile(address):
    i = address / 4096
    if i >= len(offsets):
        return Exception("Invalid Address")
    return (address - i * 4096) + offsets[i]



if __name__ == "__main__":
    address = int(sys.argv[1], 16)
    elfaddress = mapSparsefile(address)
    virtaddress = mapVirtAddress(elfaddress)
    print virtaddress
    print hex(virtaddress)
```

```
#!/bin/python

import sys

offsets = [ 0, 1627791495168, 1627791519744, 11168083808256, 11168083820544,
    25297854992384, 25297855041536, 27126397538304, 33981332525056,
    33981332570112, 33981332570112 + 4096,47446612774912, 47446612832256,
    55346731216896, 64713413758976, 64713413775360, 65249501974528,
    88943149977600, 88943150039040, 91853110185984, 91853110214656,
    99931956908032, 99931956936704, 128019591467008, 128019591507968,
    128574140710912 ]


def findPage(address):
    for i in range(len(offsets)):
        offset = offsets[i]
        if offset <= address and address < offset + 4096:
            return i * 4096 + (address - offset)
    raise Exception("Invalid Address")

def mapVirtAddress(hexaddress):
    if hexaddress >= int(0x00002b0000000000) and hexaddress < int(0
        x00002b0000000000 + 0x00001ef000000000):
        return hexaddress - int(0x00002b0000000000) + int(0x0000000000001000)

    elif hexaddress >= int(0x000049f000000000) and hexaddress < int(0
        x000049f000000000 + 0x0000161000000000):
        return hexaddress - int(0x000049f000000000) + int(0x00002affffe1000)
    elif hexaddress >= int(0x0000000000020000) and hexaddress < int(0
        x0000000000020000 + 0x00002affffe0000):
        return hexaddress - int(0x000000000020000) + int(0x000049effffe1000)
    else:
        raise Exception("Invalid Address")

if __name__ == "__main__":
    i = int(sys.argv[1], 16)
    res = findPage(mapVirtAddress(i))
    print i
    print hex(i)
    print res
    print hex(res
```

D Foo.zip

D.1 Décompression des données

```
from uefi_firmware.efi_compressor import EfiDecompress

compressed = "\x3F\x00\x00\x00\x5C\x00\x00\x00\x00\x3C\x4C\x8D\x82\x33\x02\xED\x64
    \x00\xB7\x17\x30\x7D\xA8\x12\xAF\x1B\x02\x1D\xFA\xB8\x61\x24\xFE\x3B\x24\xF5\
    x1F\xCC\xCE\x8B\xA7\x73\x85\x72\x72\x6A\x51\x8F\x09\xC1\xD4\xB1\x0C\x7B\xA3\
    xA1\xB3\xCF\x07\x8F\xCD\x9D\x55\x34\x75\xDE\xD9\x63\x83\x20\x00"

result = EfiDecompress(compressed, 0x47)
print result
```

D.2 Calcul de la clé

```
var_4C = "cb41dcb1d89746705a7fe998f11acce7"
key = ""
count = 0

# Sous-fonction modifiant les bytes de la cle
def sub_10000400(char, charoffset):
    var_10 = char
    var_C = 8
    charoffset = charoffset % var_C
    char = char >> charoffset
    leftoffset = var_C - charoffset
    var_10 = var_10 << leftoffset
    return ~(var_10 | char) & 0xFF


# Fonction inverse
def rev_sub_10000400(char, charoffset):
    var_10 = char
    var_C = 8
    charoffset = charoffset % var_C
    char = char << charoffset
    leftoffset = var_C - charoffset
    var_10 = var_10 >> leftoffset
    return ~(var_10 | char) & 0xFF

# Application de la fonction inverse sur la chaine cible
for var_5C in range(0x10):
    key += hex( rev_sub_10000400( int(var_4C[var_5C*2: var_5C*2+2], 16), var_5C) )
        [2:].zfill(2)
print('␣key:␣' + key)


# Verification de la cle
for var_5C in range(0x10):
    char_1 = sub_10000400( int(key[var_5C*2: var_5C*2+2], 16), var_5C)
    char_2 = int( var_4C[var_5C*2: var_5C*2+2], 16 )
    newchar = char_1 ^ char_2
    count |= newchar
print count
```

E Strange.zip

E.1 Récupération des images échantillons

```
from struct import unpack

def hex_to_float(fl):
    return unpack('d', fl)

o = open('196', 'rb')

for loop in range(10):
    block = o.read(65860 * 8)

    for i in range(20):
        line = ''
        for j in range(20):
            k = 20*i + j
            pixfloat = block[8*k: 8*(k+1)]
            pixdata = round(hex_to_float(pixfloat)[0])
            if pixdata:
                line += 'X'
            else:
                line += '␣'
        print line
o.close()
```

E.2 Brute-Force de la clé

```
from struct import unpack
from math import exp

def hex_to_float(fl):
    return unpack('d', fl) [0]


def subsubfunc(data, number):
    # Level 1
    list1 = []
    offset = 3216
    for i in range(160):
        r = hex_to_float( data[offset: offset + 8] )
        for k in range(400):
            r += number[k] * hex_to_float( data[offset + 8*(k+1): offset + 8*(k+2)
                ] )
        list1.append( subsubsubfunc(r) )
        offset += 3240

    # Level 2
    list2 = []
    for i in range(4):
        r = hex_to_float( data[offset: offset + 8] )
        for k in range(160):
            r += list1[k] * hex_to_float( data[offset + 8*(k+1): offset + 8*(k+2)]
                )
        list2.append( subsubsubfunc(r) )
        offset += 1320
```

```
    return list2


def subsubsubfunc(f8):
    return 1.0 / ( 1 + exp(-f8) )


def loaddigits():
    # Load the digits in order 1,2..,9,0
    r = []
    fd = open('digits')
    for digit in range(10):
        data = []
        for i in range(20):
            line = fd.readline()
            for j in range(20):
                if line[j] == 'X':
                    data.append(1.0)
                else:
                    data.append(0.0)
        r.append(data)
    fd.close()
    return r


o = open('196', 'rb')
digits = loaddigits()

password = ''
for loop in range(32):
    block = o.read(65860 * 8)
    for digit in range(10):
        r = subsubfunc(block, digits[digit])
        if r[0]<0.15 and r[1]<0.15 and r[2]<0.15 and r[3]<0.15:
            password += str( (digit + 1) % 10 )

print password
o.close()
```