Résolution Challenge SSTIC 2016

Vincent Fargues – vincent.fargues@thalesgroup.com
13/04/2016

Introduction

Ce document présente la résolution du challenge SSTIC 2016. A cause d'une mauvaise organisation temporelle de l'auteur, il n'est pas aussi détaillé que voulu. Toutefois il présente une des manières permettant de terminer le challenge

Etape 1 : Récupération du jeu

Le challenge commence avec un fichier *challenge.pcap*. Dans cette capture réseau se trouve un échange en HTTP. En utilisant la commande *tcpflow -r* il est possible d'extraire les paquets. En lançant ensuite *Binwalk* sur le les paquets extraits, on obtient un répertoire contenant un fichier index.html qui sera le vrai point de départ du challenge SSTIC 2016.

```
vincent@intru stage1 file challenge.pcap
challenge.pcap: tcpdump capture file (little-endian) - version 2.4 (Ethernet, capture length
32767)
vincent@intru stage1 tcpflow -r challenge.pcap
vincent@intru stage1 ls
010.069.016.064.40586-195.154.171.095.00080  195.154.171.095.00080-010.069.016.064.40586
challenge.pcap  report.xml
vincent@intru stage1 binwalk -re 195.154.171.095.00080-010.069.016.064.40586
vincent@intru stage1 cd _195.154.171.095.00080-010.069.016.064.40586.extracted
vincent@intru _195.154.171.095.00080-010.069.016.064.40586.extracted ls
Audio  fancybox  index.html  plugins  rpgjs  wood.png
```

On ouvre le fichier index.html dans un navigateur.

En parlant à différents PNJ, plusieurs niveaux peuvent être téléchargés. Pour le niveau 1, les fichiers suivants sont proposés :

```
vincent@intru niveau1 ls
calc.zip  radio.zip  SOS-Fant0me.zip
```

Dans cette solution, la résolution de CALC et de SOS-Fant0me sera détaillée.

Niveau 1

Niveau 1 - SOS-Fant0me

Ce niveau est une capture réseau au format PCAP. La même approche que pour l'extraction du jeu en Javascript est suivie.

Extraction des flux avec tcpflow

```
vincent@intru fantome tcpflow -r SOS-Fant0me.pcap
vincent@intru fantome ls
076.023.011.117.00080-129.020.126.116.56499  129.020.126.116.56499-076.023.011.117.00080
report.xml  SOS-Fant0me.pcap
```

Utilisation de Binwalk pour extraire les données compressées en zlib

```
vincent@intru fantome binwalk 129.020.126.116.56499-076.023.011.117.00080 -re
cd _129.020.126.116.56499-076.023.011.117.00080.extracted
vincent@intru _129.020.126.116.56499-076.023.011.117.00080.extracted ls
10B69  14B99  18BC9  1CBF4  1EC0C  20C24  240    259    275    291   2AC94  2C8    2E5
2F429  2F4CD  316  34B  37F  3B4  3E8  41E  456  48B  4AD9  517  545  57   58A  5B8  5E6  614
642  697   6F4  79A  833  8B09  AE9  CB39  EB51 12B81  16BB1  1ABE1  1D7    1EE    22C3C
24C54  26C69  28C81  2AC8  2AE    2CCAC  2EBC1  2F468  2FD    331  365  39B  3CE  401  439
472  4A5  500   52E  55C  573  5A1  5CF  5FD  62B  659  6AF1  71F  7CD  897  AB21  B44  D
```

Concaténation de toutes les données en un seul fichier.

```
vincent@intru _129.020.126.116.56499-076.023.011.117.00080.extracted cat * > condense
vincent@intru _129.020.126.116.56499-076.023.011.117.00080.extracted binwalk -re condense

DECIMAL       HEXADECIMAL     DESCRIPTION
--------------------------------------------------------------------------------
137099        0x2178B         Zip archive data, encrypted at least v1.0 to extract, compressed
size: 44, uncompressed size: 32, name: solution.txt
```

Une archive zip nécessitant un mot de passe est extraite. On cherche dans les strings du fichier reconstruit à la recherche d'un potentiel mot de passe.

```
vincent@intru _129.020.126.116.56499-076.023.011.117.00080.extracted strings condense| egrep
"\|"
|i la| clef p
94.|
9%`b|
#|3U
|r l|e sta|ge 1| ! L|e mo|t de p|ass|e de| l'a|rch|ive res|te ce|ui conv|enu
en|sem|ble.|[2016/02/27 - 23:15] sstic2016-stage1-solution.zip - Saisir mot de passe
k|=OS
ky|i
4|Tv
|C|y|b|3|r}
|S|S|T|I|C|_|2|0|1|6gC
Zu(|
oSU|
```

La string Cyb3rSSTIC_2016 semble être une bonne candidate.

```
vincent@intru _condense.extracted unzip -P Cyb3rSSTIC_2016 2178B.zip
Archive:  2178B.zip
 extracting: solution.txt
vincent@intru _condense.extracted cat solution.txt
```

```
368BE8C1CC7CC70C2245030934301C15%
```

Il est alors possible de valider le premier flag du challenge SSTIC

Niveau 1 – Calc

L'épreuve calc consiste en l'analyse d'un binaire pour calcultrice TI-83.

```
vincent@intru calc file SSTIC16.8xp
SSTIC16.8xp: TI-83+ Graphing Calculator (program)
```

Un outil existe pour décompiler les programmes pour TI83 :
https://github.com/thenaterhood/basically-ti-basic

A l'aide de ce programme on peut obtenir les instructions du langage spécifique à la TI-83 :

```
vincent@intru calc python3 /usr/bin/basically-ti-basic -d -i SSTIC16.8xp -o decom.txt
vincent@intru calc head decom.txt
ClrHome
Goto 1

Lbl 0
If S=5:Goto 5
If S=6:Goto 6
If S=8:Goto 8
If S=9:Goto 9
If S=10:Goto 10
If S=11:Goto 11
```

A l'aide du décompilé et de la documentation du langage étudié, il a été possible de réécrire un pseudo code python retraçant le flot d'exécution du programme.

```
def label2(a):
    Str1 = " "
    while a !=0:
        a = a/2.
        if(a-int(a) > 0):
            Str1= "1" +Str1
        else:
            Str1 = "0" + Str1
        a = int(a)
    Str1 = Str1[:-1]
    while len(Str1) < 32:
        Str1 = "0" + Str1
    return Str1

def xor(s_1,s_2):
            ret = ""
            for i in range(len(s_1)):
                ret += str(int(s_1[i])^int(s_2[i]))
            return ret


def label5(c):
    return label2(c)


def label3(s,a):
    return s[len(s)-(a+1)*8:len(s)-(a+1)*8+8]


def label4(str1,str2): #xor
    Str3 = xor(str1,str2)
    return Str3

def label7(str1): #bin2dec
    a = 0
    j=0
```

```
    for i in str1[::-1]:
        if i == "1":
            a+=2**j
        j=j+1
    return a
def label15(str1):
    return "00000000" + str1[:24]



Z = int(raw_input())
C = 4294967295
N = 0
l1 =
[0,1996959894,3993919788,2567524794,124634137,1886057615,3915621685,2657392035,249268274,20445
08324,3772115230,2547177864,162941995,2125561021,3887607047,2428444049,498536548,1789927666,40
89016648,2227061214,450548861,1843258603,4107580753,2211677639,325883990,1684777152,4251122042
,2321926636,335633487,1661365465,4195302755,2366115317,997073096,1281953886,3579855332,2724688
242,1006888145,1258607687,3524101629,2768942443,901097722,1119000684,3686517206,2898065728,853
044451,1172266101,3705015759,2882616665,651767980,1373503546,3369554304,3218104598,565507253,1
454621731,3485111705,3099436303,671266974,1594198024,3322730930,2970347812,795835527,148323022
5,3244367275,3060149565,1994146192,31158534,2563907772,4023717930,1907459465,112637215,2680153
253,3904427059,2013776290,251722036,2517215374,3775830040,2137656763,141376813,2439277719,3865
271297,1802195444,476864866,2238001368,4066508878,1812370925,453092731,2181625025,4111451223,1
706088902,314042704,2344532202,4240017532,1658658271,366619977,2362670323,4224994405,130353596
0,984961486,2747007092,3569037538,1256170817,1037604311,2765210733,3554079995,1131014506,87967
9996,2909243462,3663771856,1141124467,855842277,2852801631,3708648649,1342533948,654459306,318
8396048,3373015174,1466479909,544179635,3110523913,3462522015,1591671054,702138776,2966460450,
3352799412,1504918807,783551873,3082640443,3233442989,3988292384,2596254646,62317068,195781084
2,3939845945,2647816111,81470997,1943803523,3814918930,2489596804,225274430,2053790376,3826175
755,2466906013,167816743,2097651377,4027552580,2265490386,503444072,1762050814,4150417245,2154
129355,426522225,1852507879,4275313526,2312317920,282753626,1742555852,4189708143,2394877945,3
97917763,1622183637,3604390888,2714866558,953729732,1340076626,3518719985,2797360999,106882838
1,1219638859,3624741850,2936675148,906185462,1090812512,3747672003,2825379669,829329135,118133
5161,3412177804,3160834842,628085408,1382605366,3423369109,3138078467,570562233,1426400815,331
7316542,2998733608,733239954,1555261956,3268935591,3050360625,752459403,1541320221,2607071920,
3965973030,1969922972,40735498,2617837225,3943577151,1913087877,83908371,2512341634,3803740692
,2075208622,213261112,2463272603,3855990285,2094854071,198958881,2262029012,4057260610,1759359
992,534414190,2176718541,4139329115,1873836001,414664567,2282248934,4279200368,1711684554,2852
81116,2405801727,4167216745,1634467795,376229701,2685067896,3608007406,1308918612,956543938,28
08555105,3495958263,1231636301,1047427035,2932959818,3654703836,1088359270,936918000,284771489
9,3736837829,1202900863,817233897,3183342108,3401237130,1404277552,615818150,3134207493,345342
1203,1423857449,601450431,3009837614,3294710456,1567103746,711928724,3020668471,3272380065,151
0334235,755167117]
A = Z
S = 5

Str5 = label2(A) # bin
Str6 = label2(C) #bin
print Str5

S = 11
Str1 = Str6

while N<4:
    Str6 = Str1

    A = 0
    Str7 = label3(Str1,A) # extract word
    Str1 = label3(Str5,N) # extract word
    Str2 = Str7
    Str3 = label4(Str1,Str2) # xor
    Str1 = Str3
    A = label7(Str1) #bin2dec
    A = l1[A]

    Str8 = label2(A) #dec2bin
    Str1 = Str6
    Str1 = label15(Str1) # >>8
    Str2 = Str8
    Str3 = label4(Str1,Str2) # xor
    Str1 = Str3
    N=N+1
Str2 = "11111111111111111111111111111111"
Str3 = label4(Str1,Str2) # xor
Str1 = Str3
S = 19
```

```
A = label7(Str1) #bin2dec
print "A : %d"  % A
print label2(A)
print label2(3298472535)
if A == 3298472535:
    print "win"
else:
    print ":("
```

Ce code peut ensuite être simplifié pour obtenir le résultat suivant :

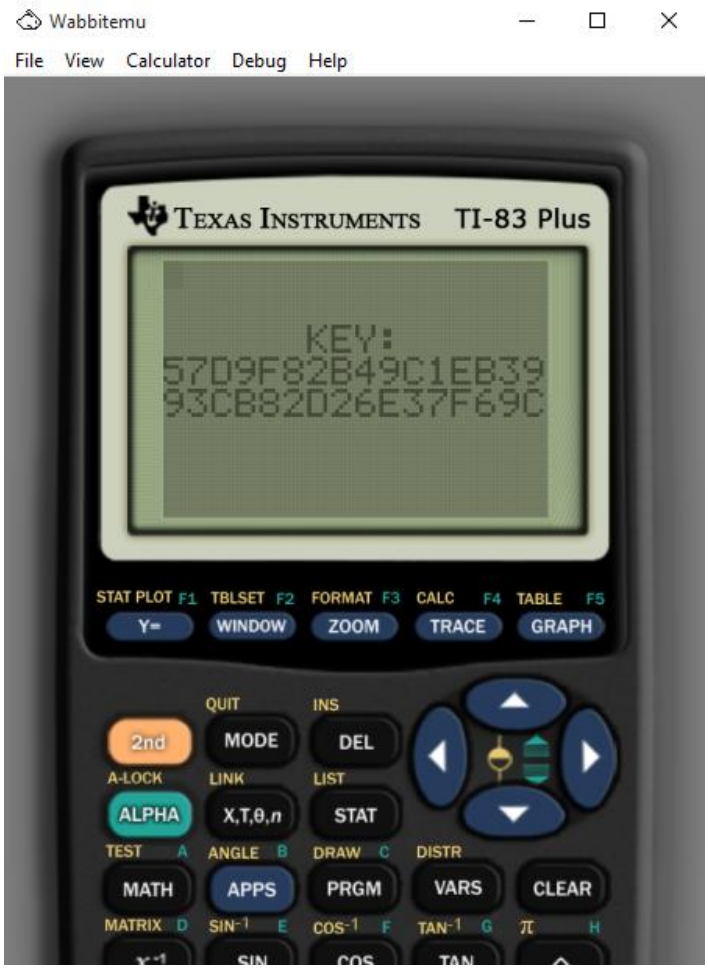
```
i = int(raw_input())
Str6 = 4294967295
l1 =
[0,1996959894,3993919788,2567524794,124634137,1886057615,3915621685,2657392035,249268274,20445
08324,3772115230,2547177864,162941995,2125561021,3887607047,2428444049,498536548,1789927666,40
89016648,2227061214,450548861,1843258603,4107580753,2211677639,325883990,1684777152,4251122042
,2321926636,335633487,1661365465,4195302755,2366115317,997073096,1281953886,3579855332,2724688
242,1006888145,1258607687,3524101629,2768942443,901097722,1119000684,3686517206,2898065728,853
044451,1172266101,3705015759,2882616665,651767980,1373503546,3369554304,3218104598,565507253,1
454621731,3485111705,3099436303,671266974,1594198024,3322730930,2970347812,795835527,148323022
5,3244367275,3060149565,1994146192,31158534,2563907772,4023717930,1907459465,112637215,2680153
253,3904427059,2013776290,251722036,2517215374,3775830040,2137656763,141376813,2439277719,3865
271297,1802195444,476864866,2238001368,4066508878,1812370925,453092731,2181625025,4111451223,1
706088902,314042704,2344532202,4240017532,1658658271,366619977,2362670323,4224994405,130353596
0,984961486,2747007092,3569037538,1256170817,1037604311,2765210733,3554079995,1131014506,87967
9996,2909243462,3663771856,1141124467,855842277,2852801631,3708648649,1342533948,654459306,318
8396048,3373015174,1466479909,544179635,3110523913,3462522015,1591671054,702138776,2966460450,
3352799412,1504918807,783551873,3082640443,3233442989,3988292384,2596254646,62317068,195781084
2,3939845945,2647816111,81470997,1943803523,3814918930,2489596804,225274430,2053790376,3826175
755,2466906013,167816743,2097651377,4027552580,2265490386,503444072,1762050814,4150417245,2154
129355,426522225,1852507879,4275313526,2312317920,282753626,1742555852,4189708143,2394877945,3
97917763,1622183637,3604390888,2714866558,953729732,1340076626,3518719985,2797360999,106882838
1,1219638859,3624741850,2936675148,906185462,1090812512,3747672003,2825379669,829329135,118133
5161,3412177804,3160834842,628085408,1382605366,3423369109,3138078467,570562233,1426400815,331
7316542,2998733608,733239954,1555261956,3268935591,3050360625,752459403,1541320221,2607071920,
3965973030,1969922972,40735498,2617837225,3943577151,1913087877,83908371,2512341634,3803740692
,2075208622,213261112,2463272603,3855990285,2094854071,198958881,2262029012,4057260610,1759359
992,534414190,2176718541,4139329115,1873836001,414664567,2282248934,4279200368,1711684554,2852
81116,2405801727,4167216745,1634467795,376229701,2685067896,3608007406,1308918612,956543938,28
08555105,3495958263,1231636301,1047427035,2932959818,3654703836,1088359270,936918000,284771489
9,3736837829,1202900863,817233897,3183342108,3401237130,1404277552,615818150,3134207493,345342
1203,1423857449,601450431,3009837614,3294710456,1567103746,711928724,3020668471,3272380065,151
0334235,755167117]
N=0
Str1 = Str6
while N<4:
    Str6 = Str1
    A = 0
    Str1 = ((i & (0xff<<N*8))>>(N*8)) ^(Str1 & 0xff)
    A = l1[Str1]
    Str1 = (Str6>>8)^A
    N=N+1

Str1 ^= 0xffffffff
if Str1 == 3298472535:
    print "Win !!!"
    print "Answer : %s" % i
```

Ce script est utilisé afin de bruteforcer la solution en quelques minutes :

```
vincent@intru calc python script4.py
89594902
Win !!!
Answer : 89594902
```

Un émulateur est ensuite utilisé afin de rentrer cette valeur et d'obtenir la solution de manière graphique.



Le deuxième flag du niveau 1 est ainsi validé et il est possible de passer au niveau 2.

Niveau 2

Le niveau 2 propose une nouvelle fois trois épreuves différentes :

```
vincent@intru niveau2 ls
foo  huge  loader
```

La résolution de foo et de loader sera présentée ici.

Niveau 2 – Foo

Le niveau Foo se compose d'un seul fichier :

```
vincent@intru new unzip foo.zip
Archive:  foo.zip
  inflating: foo.efi
```

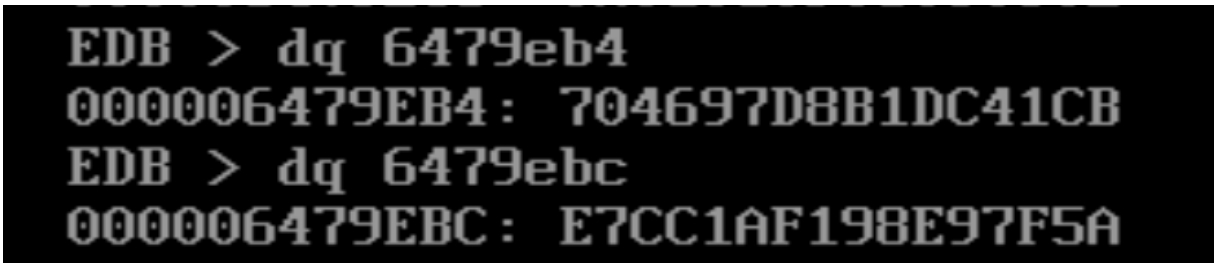
Ce niveau propose donc un crackme dans le langage EFI.

A l'aide de qemu il est possible de le lancer dans un environnement simulé. De plus il est possible d'utiliser le debugger EBC Debugger (http://www.uefi.org/node/550) afin d'avoir un debugger permettant de poser des breakpoints et d'afficher de la mémoire.

Durant la phase de reverse statique, une ligne attire particulièrement l'attention :

```
.text:10000954                  XOR32       R7, R5      ; Op1 ^= Op2
```

Sur cette ligne R5 contient une valeur calculée par le programme et R7 est une valeur dérivée de l'entrée utilisateur. Le reverse du traitement de l'entrée utilisateur ayant permis de réécrire un pseudo code python équivalent, il suffit de récupérer à l'aide du debugger les valeurs successives de R5 pour remonter à la clef attendue en entrée.



Le code python suivant permet alors de calculer la clef attendue en entrée.

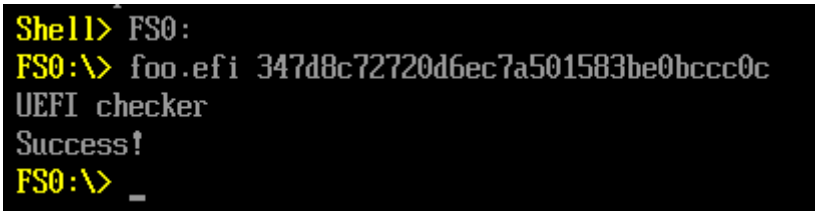
```
def sub400(text,len):
    l = len % 8
    var_a = (ord(text)>>l) & 0xff
    l2 = 8 - l
    var_b = (ord(text)<< l2) & 0xff
    return (var_a | var_b)^0xff


k=0
soluce =""
for j in "CB41DCB1D89746705A7FE998F11ACCE7".decode("hex"):
    for i in range(255):
        if j==chr(sub400(chr(i),k)):
            soluce += chr(i)
    k+=1

print "Solution : %s" % soluce.encode("hex")
```

```
vincent@intru foo python script2.py
Solution : 347d8c72720d6ec7a501583be0bccc0c
```

La validité de cette clef est vérifiée avec le binaire EFI :



Le premier flag du niveau 2 est ainsi validé

Niveau 2 – Loader

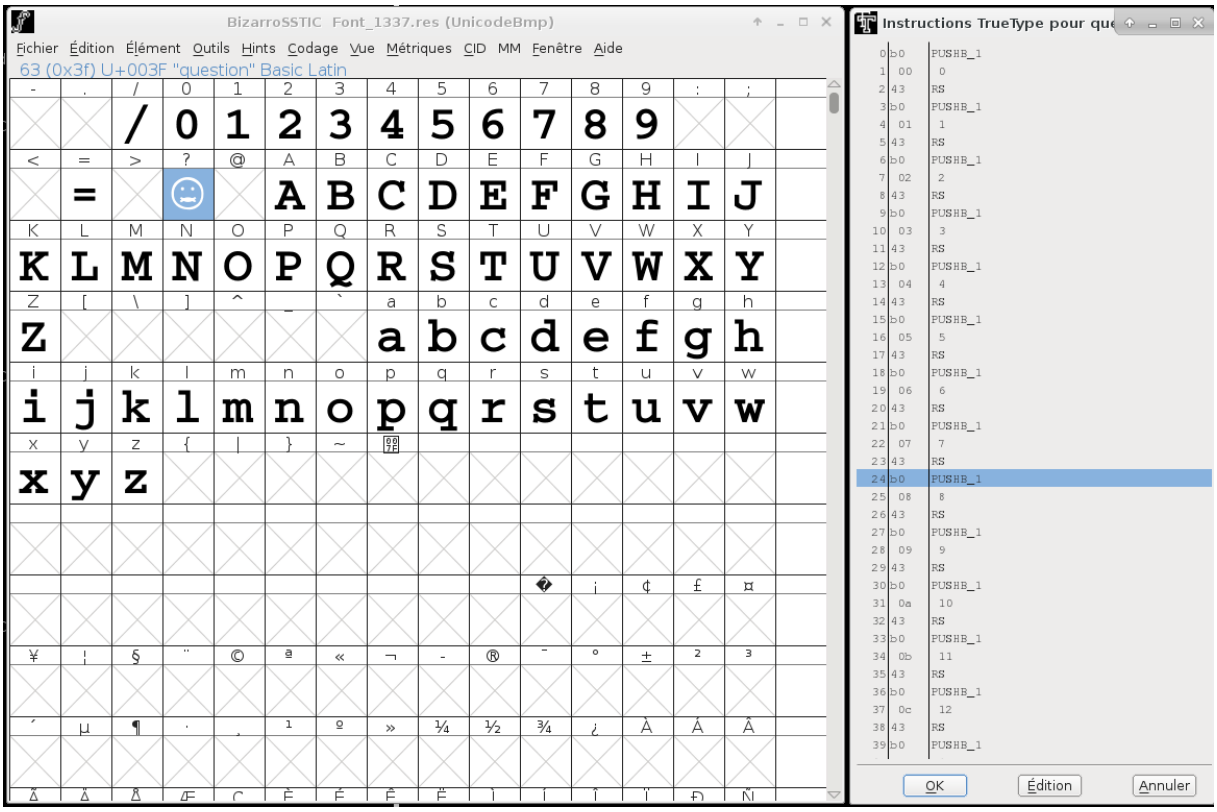
Le niveau loader est un exécutable Windows.

```
vincent@intru loader file loader.exe
loader.exe: PE32 executable (GUI) Intel 80386, for MS Windows
```

Ce binaire présente seulement un champ pour entrer du texte. La touche entrée n'a aucune influence sur le comportement. Cependant, le point d'interrogation est remplacé par un smiley « triste ». Le but

de ce niveau semble donc être d'obtenir un smiley « content ».

Après une analyse rapide du code de l'exécutable dans IDA, il apparait que ce comportement différent du smiley ne provient pas du code du binaire en lui-même. Il apparait alors intéressant de s'intéresser au code de la police de caractère utilisée. Cette ressource est extraite à l'aide de PE Explorer et est analysée à l'aide de Fontforge.



Chaque lettre a un code particulier dans le langage TrueType. Après une analyse de ce Bytecode il apparait le schéma suivant. Chaque lettre tapée est inscrite dans un tableau de 24 caractères dans l'ordre où elles ont été rentrées par l'utilisateur. Lorsque le caractère point d'interrogation est entré, des calculs sont effectués sur ce tableau afin d'afficher soit le smiley content, soit le smiley triste.

Un pseudo émulateur symbolique a été écrit afin de comprendre le fonctionnement du code de ce caractère.

```
def roll():
    mindex(3)

def mindex(v):
    v= int(v)
    last = stack[-v]
    for i in reversed(range(2,v+1)):
        stack[-i] = stack[-(i-1)]
    stack[-1]=last

def mul(v):
    v = int(v)
    v = v>>6
```

```
    stack[-1] = "(" + stack[-1] +")" + "*" + str(v)

def eq(v):
    global stack
    print stack[-1] + "==" + v
    stack = stack[:-1]

def add(v):
    stack[-1] = stack[-1] +"+"+ str(v)

def sub(v):
    stack[-1] = stack[-1] +"-"+ str(v)
def swap():
    mindex(2)


for i in range(22):
    stack +=["pos"+str(i)]
```

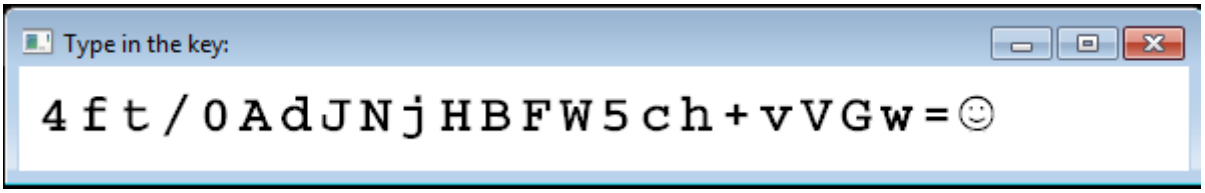
Les opérations sont ensuite lancées dans cet émulateur et les équations suivantes sont obtenues :

```
pos2==19
(pos12)*6==186
pos3==63
pos8==39
pos7==35
(pos13+6)*5-7==263
pos5==26
pos21+6==28
pos18==21
pos0-3==53
pos11-6==21
((pos19)*4+6)*3==582
(pos1)*4==20
pos15==2
((pos4)*4+1)*7==1463
((pos14)*5-2)*5==1415
pos17+2+4+2==70
pos10-1==32
pos16-2==5
((pos9+2+7)*1)*5==90
(pos6+5-1-3+5-6)*1==3
(((pos20-7-1)*4+3)*4-2)*5==1970
```

Après simplification un index dans l'alphabet base64 test obtenu pour chaque position :

```
pos2==19
pos12==31
pos3==63
pos8==39
pos7==35
pos13==48
pos5==26
pos21==22
pos18==21
pos0==56
pos11==27
pos19==47
pos1==5
pos15==2
pos4==52
pos14==57
pos17==62
pos10==33
pos16==7
pos9==9
pos6==3
pos20==32
```

Le résultat final est alors obtenu :



Le flag pour valider le niveau 2 peut être lu en décodant base64 :

```
echo -n "4ft/0AdJNjHBFW5ch+vVGw==" | base64 -d | xxd
00000000: e1fb 7fd0 0749 3631 c115 6e5c 87eb d51b  .....I61..n\....
```

Niveau 3

Le niveau 3 propose cette fois quatre épreuves différentes :

```
vincent@intru niveau3 ls
ring  strange  usb  vidéo
```

Les épreuves ring et strange valent 2 points chacune permettant de valider le niveau 3 avec une seule épreuve alors que usb et vidéo valent 1 point chacune et doivent être validées toutes les deux. La solution pour usb et vidéo sera présentée ici.

Niveau 3 – USB

Le niveau USB est composé de deux fichiers :

```
vincent@intru usb2 ls
img.bz2  userSSTIC.bin
vincent@intru usb2 file *
img.bz2:       bzip2 compressed data, block size = 900k
userSSTIC.bin: PE32+ executable (GUI) x86-64, for MS Windows
```

Le fichier userSSTIC.bin est un exécutable Windows qui installe un driver userSSTIC.sys.

Le fichier img.bz2 est une image d'une clef USB.

La rétro ingénierie du driver Windows met en évidence des routines de chiffrement et de déchiffrement. Le but de cette épreuve semble donc être de retrouver des fichiers chiffrés dans l'image de la clef USB fournie.

La rétro ingénierie montre l'utilisation d'un chiffrement de type RC6 avec une clef constante. Toute tentative d'utilisation des implémentations disponibles de RC6 ayant conduit à des résultats différents de ceux du driver, il a été décidé d'appeler directement les fonctions du driver.

Le fichier image présente 4 partitions, mais il apparait rapidement que les données intéressantes sont chiffrées et intercalées entre les partitions de la clef. Le but va donc être de reconstruire les données présentes entre les partitions, de les déchiffrer en utilisant rc6 et enfin d'appliquer les fonctions du binaire permettant de déchiffrer chaque fichier présent sur la clef.

Après le déchiffrement RC6, une structure apparait pour chaque fichier présent dans le filesystem caché. Un header annonce notamment la longueur du fichier et permet d'appeler les fonctions du driver avec les bons paramètres. Le code suivant permet d'extraire 5 fichiers dont le dernier contient la solution.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```
#include <fcntl.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <time.h>
#include <sys/syscall.h>

#include "offsets.h"

#define READ_SIZE 1024*1024*10


char * (*real_sub_11E14)(void *,void *,unsigned char *, unsigned char *);
char * (*real_sub_12284)(void *,void *,unsigned char *, unsigned char *);
char * (*real_sub_122F8)(void *,void *,unsigned char *, unsigned char *, unsigned char *,
unsigned int);
char * (*real_sub_12038)(void *,void *,unsigned char *, unsigned char *, unsigned char *);


char * sub_11E14(unsigned char * arg1) {
        return real_sub_11E14(NULL,NULL,NULL,arg1);
}
char * sub_12284(unsigned char * arg1, unsigned char * arg2) {
        return real_sub_12284(NULL,NULL,arg2,arg1);
}
char * sub_122F8(unsigned char * arg1, unsigned char * arg2, unsigned char * arg3, unsigned
int arg4) {
        return real_sub_122F8(NULL,NULL,arg2,arg1,arg3,arg4);
}
char * sub_12038(unsigned char * arg1, unsigned char * arg2, unsigned char * arg3) {
        return real_sub_12038(NULL,NULL,arg2,arg1,arg3);
}

int main(int argc, char ** argv) {
        int fd,i,j,k,m;
        struct stat statbuf;

        FILE *fp;
        char *dll;
        char *bytes;
        int out;

        /* mmap driver in memory */
        fd=open("../kernel_mod.bin",O_RDONLY);
        fstat(fd,&statbuf);
        if (( dll = mmap(0,statbuf.st_size, PROT_READ|PROT_WRITE|PROT_EXEC, MAP_PRIVATE, fd,
0)) == (caddr_t)-1 ) {
                printf("error in mmap\n");
                exit(1);
        }
        close(fd);

        /* get functions pointers */
        real_sub_12284 = (int (*)(unsigned char*, unsigned char *))dll+OFFSET_SUB_12284;
        real_sub_122F8 = (int (*)(unsigned char*, unsigned char *))dll+OFFSET_SUB_122F8;
        real_sub_11E14 = (int (*)(unsigned char*, unsigned char *))dll+OFFSET_SUB_11E14;
        real_sub_12038 = (int (*)(unsigned char*, unsigned char *))dll+OFFSET_SUB_12038;


        char * tmp =
"\x75\xab\x29\x21\xc8\x74\x53\x97\xac\xd5\xea\x5e\x2f\x31\x8b\x2c\x22\x13\x0a\xfd\x3c\x13\xd0\
x80\xc2\x49\xa8\x16\x4a\x4c\x06\x42\xf5\x77\xfe\x75\xd7\xf4\xda\x4d\x58\x14\x22\xe9\x25\x7a\xa
9\x46\x95\x44\xa7\xfe\x17\xd5\x19\xe1\x05\x26\x5f\x05\x81\x6c\x70\xc6\x22\x68\x96\x4d\x31\xe8\
xc3\xad\xdf\x8b\xc6\x68\xac\x7d\xb5\xfc\x01\x3f\xf3\x7d\x1f\x08\xb6\xef\xeb\x29\xeb\x98\xb7\x5
2\x83\x66\x5b\x54\xa1\x98\xcd\x64\x3e\x0a\x29\xa9\x16\x9b\xc4\xc1\x33\x22\x25\xec\x79\x78\x6a\
x46\xc4\x17\xea\x37\x0b\x6e\xb2\xeb\x30\xde\x5c\x09\xef\x01\x33\xdb\xfb\x35\xb7\x35\x7b\xa9\x2
c\x65\xbf\xdf\x98\x87\x66\xaf\x48\x65\x84\xb7\x6a\x70\xd8\xaf\xed\x6c\x34\x2d\xe3\xdf\x33\xbb\
xfc\xad\x79\xae\x46\x51\x11\xc3\x71\x14\xa5\x05\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x0
0\x00\x00\x00\x00\x00\x80\x1a\x07\x00\xe0\xff\xff\x00\x02\x00\x00\x00\x00\x00\x00\xf8\x2f\xf6\
x05\x00\xe0\xff\xff\x10\x41\xab\xee\x00\xf8\xff\xff";
        char *v17 = (char *)malloc(288*sizeof(char));
        memcpy(v17,tmp,288);


        char *plaintext = (char *)malloc(READ_SIZE*sizeof(char));
        char *ciphertext = (char *)malloc(READ_SIZE*sizeof(char));

        int offset_bloc2 = 0x40000601;
```

```
        // Read content
        char *DATA;
        fd=open("../img",O_RDONLY);
        lseek(fd,0x00000240,0);
        read(fd,ciphertext,961);
        lseek(fd,offset_bloc2,0);
        read(fd,ciphertext+961,0x60000);




        char *out_array=(char *)malloc(512*sizeof(char));
        char *rand_array=(char *)malloc(16*sizeof(char));


        memset(out_array,0x0,0x102);
        for(j=0;i<READ_SIZE;i+=16) {
                sub_12038(v17,ciphertext+i,plaintext+i);
        }

        // First File

        memcpy(rand_array,plaintext+4,16);
        out= sub_12284(out_array,rand_array);
        plaintext+=36;
        sub_122F8(out_array,plaintext,plaintext,0x39);
        fp = fopen("./extract/file1", "w+");
        fwrite(plaintext,0x39, 1, fp);
        fclose(fp);
        plaintext += 0x39;

        // Second File

        memcpy(rand_array,plaintext+4,16);
        out= sub_12284(out_array,rand_array);
        plaintext+=36;
        sub_122F8(out_array,plaintext,plaintext,0xC15);
        fp = fopen("./extract/file2", "w+");
        fwrite(plaintext,0xC15, 1, fp);
        fclose(fp);
        plaintext += 0xC15;

        //Third File

        memcpy(rand_array,plaintext+4,16);
        out= sub_12284(out_array,rand_array);
        plaintext+=36;
        sub_122F8(out_array,plaintext,plaintext,0x034631);
        fp = fopen("./extract/file3", "w+");
        fwrite(plaintext,0x034631, 1, fp);
        fclose(fp);
        plaintext += 0x034631;

        //4th File

        memcpy(rand_array,plaintext+4,16);
        out= sub_12284(out_array,rand_array);
        plaintext+=36;
        sub_122F8(out_array,plaintext,plaintext,0x1b234);
        fp = fopen("./extract/file4", "w+");
        fwrite(plaintext,0x1b234, 1, fp);
        fclose(fp);
        plaintext += 0x1b234;


        //5th File

        memcpy(rand_array,plaintext+4,16);
        out= sub_12284(out_array,rand_array);
        plaintext+=36;
        sub_122F8(out_array,plaintext,plaintext,0xfbe0);
        fp = fopen("./extract/file5", "w+");
        fwrite(plaintext,0xfbe0, 1, fp);
        fclose(fp);
        plaintext += 0xfbe0;
```

```
}
```

Les fichiers obtenus sont ensuite analysés :

```
ls
file1  file2  file3  file4  file5
vincent@intru extract file *
file1: ASCII text
file2: Non-ISO extended-ASCII text, with very long lines
file3: PDF document, version 1.5
file4: JPEG image data, Exif standard: [TIFF image data, big-endian, direntries=7,
orientation=upper-left, xresolution=98, yresolution=106, resolutionunit=2, software=Adobe
Photoshop CS5 Windows, datetime=2011:06:13 13:36:51], baseline, precision 8, 640x360, frames 3
file5: Zip archive data, at least v2.0 to extract
13:42:32 vincent@intru extract cat file1
password for the zip file : !WooYouAreSuchAnAwesomeGuy!
```

Ce qui est intéressant ici est le fichier file5 qui est un fichier zip et qui peut être extrait grâce au mot de passe présent dans le fichier 1.

```
vincent@intru extract unzip -P '!WooYouAreSuchAnAwesomeGuy!' file5
Archive:  file5
  inflating: 4.jpg
 extracting: key
vincent@intru extract xxd key
00000000: 0928 bde1 e3ed 8969 8632 dbff 4a23 1138  .(.....i.2..J#.8
```

Il est alors possible de valider le premier flag du niveau 3.

Niveau 3 - Vidéo

Le niveau vidéo se compose de plusieurs éléments

```
Archive:  video.zip
  inflating: Stage_anti_APT_chez_Airlhes/Airlhes_screensaver_setup.exe
  inflating: Stage_anti_APT_chez_Airlhes/Airlhes_CYBER_SECRET_possible_exfiltration.mp4
  inflating: Stage_anti_APT_chez_Airlhes/mission.txt
```

Le fichier .exe est destiné à installer un nouvel écran de veille dans un système Windows. Il ajoute aussi une clef de registre associée à l'écran de veille.

Le fichier .mp4 est une vidéo filmée dans le noir avec le reflet des couleurs d'un écran d'ordinateur sur le mur.

A partir de ces éléments et du message contenu dans mission.txt, il apparait évident qu'il va falloir comprendre comment la clef de registre influence le contenu des couleurs affichées par l'écran de veille. Dans un premier temps cela semble complexe notamment par rapport à la complexité de l'écran de veille. Cependant, il semble exister un mode dégradé de cet écran de veille dans lequel une seule couleur est affichée à l'écran en même temps. Ce comportement semble correspondre à la vidéo qui est fournie avec l'épreuve.

La partie rétro ingénierie du binaire s'est donc concentrée sur ce mode dégradé. Les éléments principaux suivants ont été compris :

- Un octet dans la clef de registre crée une séquence de 3 couleurs successives.
- Un tableau de 8 couleurs possibles est utilisé

- Une séquence d'initialisation et de fin permet de délimiter la séquence correspondant à la clef de registre.
- Il est possible de remonter à la clef de registre par une attaque en bruteforce traitant les octets de clefs un par un.

L'ordre de couleur suivant a été relevé :

```
Couleurs = ['turquoise', 'violet', 'turquoise', 'violet', 'bleu', 'violet', 'rouge', 'bleu',
'noir', 'rouge', 'bleu', 'turquoise', 'blanc', 'jaune', 'bleu', 'rouge', 'noir', 'violet',
'blanc', 'vert', 'jaune', 'noir', 'rouge', 'jaune', 'turquoise', 'blanc', 'noir', 'turquoise',
'noir', 'vert', 'blanc', 'jaune', 'bleu', 'jaune', 'bleu', 'violet', 'bleu', 'violet', 'noir',
'blanc', 'bleu', 'turquoise', 'bleu', 'vert', 'bleu', 'blanc', 'bleu', 'blanc', 'rouge',
'blanc', 'vert', 'turquoise', 'rouge', 'jaune', 'turquoise', 'vert', 'bleu', 'rouge', 'vert',
'turquoise', 'rouge', 'jaune', 'violet', 'blanc', 'vert', 'violet', 'turquoise', 'jaune',
'blanc', 'noir', 'rouge', 'turquoise', 'blanc', 'noir', 'turquoise', 'noir', 'bleu', 'rouge',
'blanc', 'violet', 'noir', 'vert', 'rouge', 'violet']
```

A partir de là un script a pu être écrit permettant de retrouver la clef de registre initiale :

```
vals = [0x4F488EB3, 0x4FB771B3, 0x4F4871B3, 0x4FB78E4C, 0x4F488E4C, 0x4FB7714C, 0x4F48714C,
0x4FB78EB3]

color = {0xff0000:"rouge",
0xff00:"vert",
0xffff00:"jaune",
0xff:"bleu",
0xff00ff:"violet",
0xffff:"turquoise",
0xffffff:"blanc",
0x0:"noir"}

time = ""
a = ["3FA43028","2304286D","DC322A36","4BA00BAD","641F20E8","C4F40A84","C08D8AC7"]
for b in a:
    time+=b.decode("hex")[::-1]


Couleurs = ['turquoise', 'violet', 'turquoise', 'violet', 'bleu', 'violet', 'rouge', 'bleu',
'noir', 'rouge', 'bleu', 'turquoise', 'blanc', 'jaune', 'bleu', 'rouge', 'noir', 'violet',
'blanc', 'vert', 'jaune', 'noir', 'rouge', 'jaune', 'turquoise', 'blanc', 'noir', 'turquoise',
'noir', 'vert', 'blanc', 'jaune', 'bleu', 'jaune', 'bleu', 'violet', 'bleu', 'violet', 'noir',
'blanc', 'bleu', 'turquoise', 'bleu', 'vert', 'bleu', 'blanc', 'bleu', 'blanc', 'rouge',
'blanc', 'vert', 'turquoise', 'rouge', 'jaune', 'turquoise', 'vert', 'bleu', 'rouge', 'vert',
'turquoise', 'rouge', 'jaune', 'violet', 'blanc', 'vert', 'violet', 'turquoise', 'jaune',
'blanc', 'noir', 'rouge', 'turquoise', 'blanc', 'noir', 'turquoise', 'noir', 'bleu', 'rouge',
'blanc', 'violet', 'noir', 'vert', 'rouge', 'violet']

current_couleurs=[]
res="\x00"*0x1c

for f in range(0x1c):
    for i in range(0,255,1):
        input_reg = res[:f] + chr(i) + res[f+1:]
        result = []
        l_input = 0x1c
        mark1=6
        mark2=0
        mark3=0



        case1_1=0
        case1_2=0
        stop = 0
        while case1_1 <l_input:
            if case1_2:
                mark3 = mark3/7
                mark2 = (mark3 % 7) +1
                mark4 = (mark1 + mark2) & 0x7
                mark1= mark4
                case1_2 = (case1_2+1)%3
                if(case1_2==0):
```

```
                    case1_1 = (case1_1+1) % l_input
                    if (case1_1==0):
                        stop=1
            else:
                case1_2 = 1
                mark3 = ord(input_reg[case1_1]) ^ ord(time[case1_1])
                mark2 = (mark3 % 7) +1
                mark4 = (mark1 + mark2) & 0x7
                mark1= mark4


            result +=[color[(vals[mark4] ^ 0x4FB78EB3)]]
            if stop==1:
                break
        if result[:(f+1)*3]==Couleurs[:(f+1)*3]:
            octet = i
            break
            print result
    current_couleurs= result
    res = res[:f] + chr(octet) + res[f+1:]
print res.encode("hex")
```

La clef de registre présente lors de l'enregistrement de la vidéo peut donc être retrouvée :

```
vincent@intru tryhard python script2.py
1800000078da0b1634172bdac558f9cb6e6257f3be7b5b003250077e
```

Le format de la clef ne correspond pas au format standard de 16 octets utilisé pendant tout le challenge mais un header zlib est présent : \x78\xda

```
zlib.decompress("78da0b1634172bdac558f9cb6e6257f3be7b5b003250077e".decode("hex")).encode("hex"
)
'5311371672ba0179fa3e918a83bedeb4'
```

La clef du niveau vidéo est validée et le challenge SSTIC 2016 est terminé, ou presque …

Final

En validant la dernière clef, le dernier challenge qui n'est pas des moindres est révélé:

```
vincent@intru sstic2016 cat final.txt
Coucou !

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

Grâce à un script python d'une complexité redoutable, il est possible d'obtenir enfin l'adresse mail tant désirée :

```
>>> "I01p1 y'4qe3553 z41y : 8Y6d5j9Vy88HUGHfGSKsJvqA@ffgvp.bet".decode("rot13")
u"V01c1 l'4dr3553 m41l : 8L6q5w9Il88UHTUsTFXfWidN@sstic.org"
```

Merci aux organisateurs pour ce challenge passionnant qui n'a pas encore livré tous ses secrets avec des niveaux qui n'ont pas encore été résolus.