

# Solution du challenge SSTIC 2016

Julien Lenoir

22 mai 2016

## **Introduction**

Voici ma solution pour le challenge. Elle est un peu bâclée par manque de temps j'en ai bien conscience. Je ne concours pas pour gagner le classement qualité, je sais qu'il y aura toujours un adepte des ordiphones et autres tubes nommés produisant une solution de 120 pages bien meilleure que la mienne.

# Niveau 1

## Calculatrice

Le premier challenge est un programme pour TI-83/TI 83+. Après en avoir extrait le code TiBasic on peut commencer son analyse. Le programme demande à l'utilisateur un nombre en entrée :

```
1 Lbl 1
2 Input "Entrez le code : ",Z
3 4294967295->C
4 0->N
5 {0,1996959894,3993919788,2567524794,124634137,1886057615,3915621685,...
6 Z->A
7 5->S
8 Goto 2
```

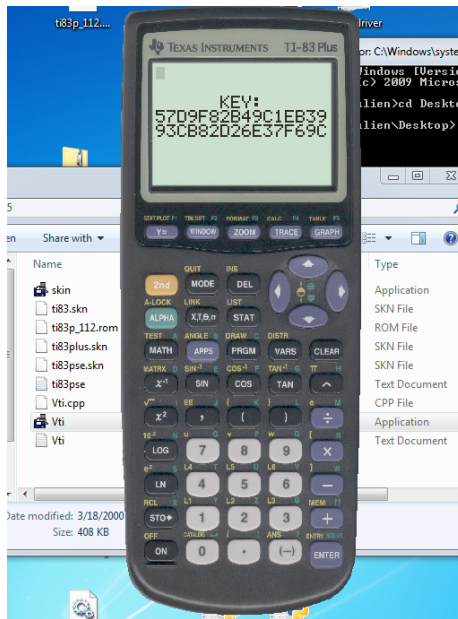
Puis effectue une boucle de calcul avec en paramètre le nombre entré par l'utilisateur. Si le résultat de ce calcul vaut 3298472535 le programme génère la clé du challenge grâce au générateur aléatoire de la calculatrice. L'entrée utilisateur sert alors de graine au générateur aléatoire :

```
1 Lbl 19
2 ClrHome
3 If A=3298472535:Then
4   Z->rand
5   21->S
6   1->X
7   4->Y
8   Output(3,7,"KEY:")
9   Goto 21
10  Else
11   Output(3,6,"PERDU")
12 End
13 Goto X
```

On peut donc implémenter l'algorithme en python pour finalement se rendre compte qu'il s'agit d'un CRC32...

```
1 Arr = [0,1996959894,3993919788,2567524794,124634137,1886057615,3915621685,...
2
3 user_input = 89594902
4
5 state = 0xFFFFFFFF
6
7 for i in range(4):
8
9     byte1 = state & 0xFF
10    byte2 = (user_input >> (8 * i)) & 0xFF
11
12    byte3 = byte2 ^ byte1
13
14    X = Arr[byte3]
15
16    state = state >> 8
17
18    state = state ^ X
19
20 state = state ^ 0xFFFFFFFF
21
22 if state == 3298472535:
23     print "Woot!"
24     quit()
```

N'ayant pas une TI83 sous la main j'ai utilisé un émulateur, en l'occurrence l'outil *Virtual TI*. Il suffit de donner au programme le nombre 89594902 :

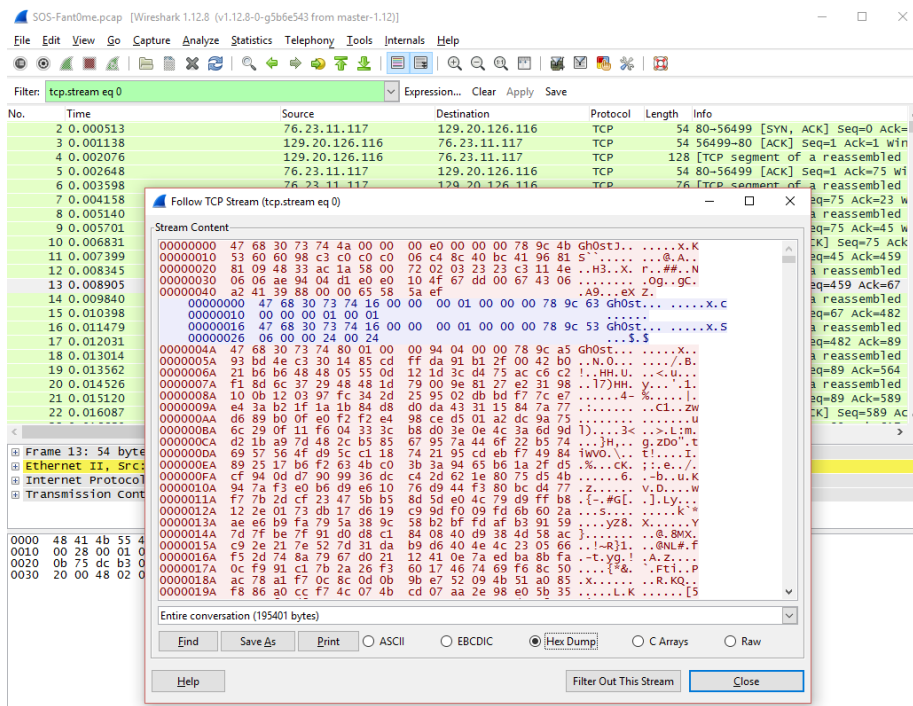


Pour obtenir la clé : **57D9F82B49C1EB3993CB82D26E37F69C**.

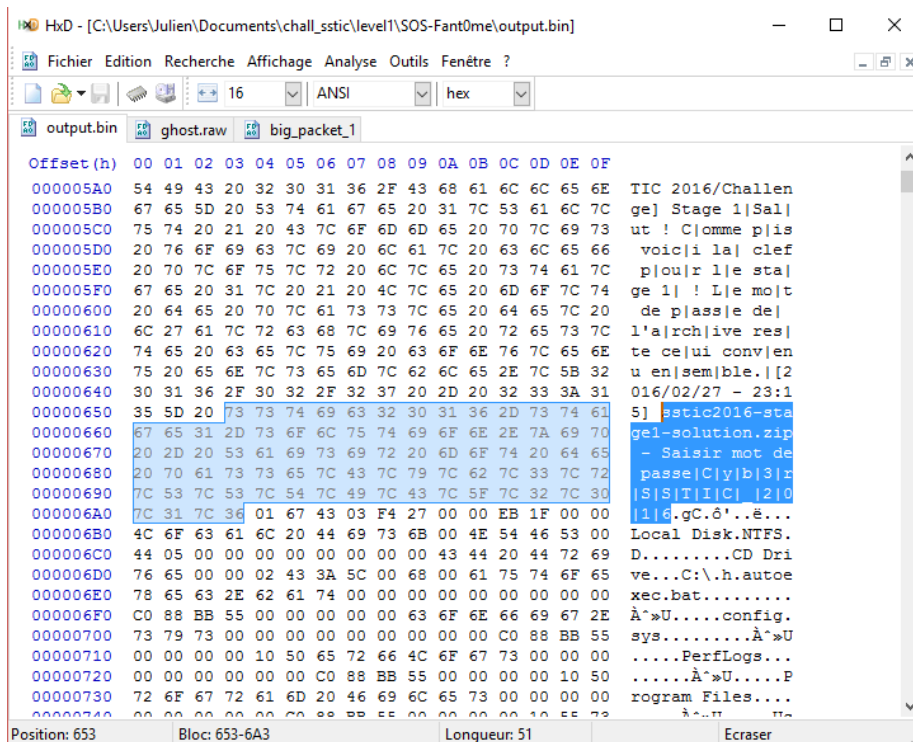
## SOS-Fant0me

Le challenge SOS-Fant0me se présente sous la forme d'un pcap. On s'aperçoit rapidement, grâce aux "magic" de chaque paquet qu'il s'agit d'un trafic réseau généré par *Gh0st RAT*<sup>1</sup>.

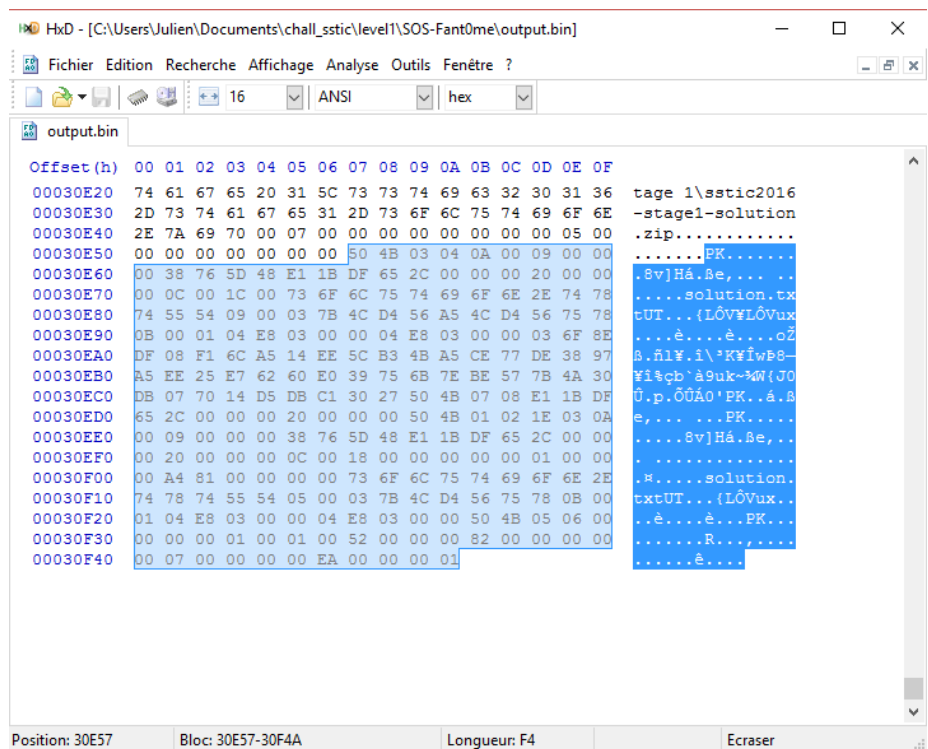
1. <http://www.cert.ssi.gouv.fr/site/CERTA-2014-ACT-002/CERTA-2014-ACT-002.html>



En se documentant sur le protocole on peut extraire les blocs compressés dans un fichier. L'analyse laisse apparaître la transmission d'un mot de passe pour une archive :



On cherche alors une archive dans ce même échange. Cette archive se trouve à la fin de l'échange :



Il suffit d'extraire cette archive avec le mot de passe transmis pour obtenir la clé suivante : **368BE8C1CC7CC70C2245030934301C15**

## Niveau 2

### foo

Dans cette épreuve il faut analyser un binaire UEFI. Il semble exister des debuggers pour ce genre de choses mais ça a l'air vraiment fatiguant à installer et à paramétrer donc j'ai privilégié l'approche statique. La documentation *Unified Extensible Firmware Interface Specification 2.5* est d'une aide précieuse. Une fois compris la convention d'appel et le byte code qui est relativement simple, l'analyse se fait plutôt bien.

Le programme prend en argument une chaîne hexadécimale de 16 octets. Rapidement, un GUID est généré en "xorand" deux buffers en dur dans le binaire :

```
1  loc_100005CA:                ; CODE XREF:
2      MOVnd     R7, [SP+0x85E0+hc_buf2_1D8_]
3      MOVbw     R7, [R7]
4      MOVqd     R4, SP+0x85E0+buf1_copy
5      MOVdd     R5, [SP+0x85E0+counter_loop_1]
6      EXTND64   R5, R5
7      ADD64     R4, R5
8      MOVbw     R4, [R4]
9      XOR32     R7, R4
10     MOVnd     R4, [SP+0x85E0+hc_buf2_1D8_] ;
11     MOVbw     [R4], R7
12     MOVdd     R7, [SP+0x85E0+counter_loop_1]
13     MOVqw     R7, R7+1
14     MOVdd     [SP+0x85E0+counter_loop_1]
15     MOVnd     R7, [SP+0x85E0+hc_buf2_1D8_]
16     ADD64     R7, R6+1
17     MOVnd     [SP+0x85E0+hc_buf2_1D8_], R7
18     MOVdd     R7, [SP+0x85E0+counter_loop_1]
19     CMPI32wgte R7, 0x10
20     JMP8cc    loc_100005CA
```

Les deux buffers en question sont : D0D97176DB537341AA692327F21F0BC7 et 2EA560AE7DC7A750305323B7D520CA8A83.

Le résultat est : **fe7c11d8a694d4119a3a0090273fc14d**. C'est le GUID du protocole *EFI\_DECOMPRESS\_PROTOCOL\_GUID* :

```
1 #define EFI_DECOMPRESS_PROTOCOL_GUID \
2 {0xd8117cfe,0x94a6,0x11d4,\
3 {0x9a,0x3a,0x00,0x90,0x27,0x3f,0xc1,0x4d}}
```

Dont l'interface est la suivante :

```
1 typedef struct _EFI_DECOMPRESS_PROTOCOL {
2     EFI_DECOMPRESS_GET_INFO GetInfo;
3     EFI_DECOMPRESS_DECOMPRESS Decompress;
4 } EFI_DECOMPRESS_PROTOCOL;
```

Les données compressées se trouvent dans une structure située à l'offset 0x10001470 :

```
1 .data:10001470                db 0x3F
2 .data:10001471                db 0
```

```

3  .data:10001472          db    0
4  .data:10001473          db    0
5  .data:10001474          db  0x5C
6  .data:10001475          db    0
7  .data:10001476          db    0
8  .data:10001477          db    0
9  .data:10001478          db    0
10 .data:10001479          db  0x3C
11 .data:1000147A          db  0x4C
12 .data:1000147B          db  0x8D
13 .data:1000147C          db  0x82
14 .data:1000147D          db  0x33

```

Pour décompresser ces données j'ai utilisé une implémentation en python de l'algorithme TIANO :

```

1  #! /usr/bin/env python2
2
3  #- python implementation of tianocore decompress
4  #- see EDK-files BaseUefiDecompressLib.c and PeiLib/Decompress.c
5  #- date: 03/2012
6  #- author: Jakob Heinemann, jakob@jakobheinemann.de
7  #- license: GPL
8
9  #- usage:
10 #-     tiano = TIANO_DECOMP()
11 #-     uncombuf = tiano.Decompress(combuf)

```

Le buffer décompressé est le suivant :

```

1  secret data: cb41dcb1d89746705a7fe998f11acce7

```

Le programme UEFI convertit la chaîne depuis l'ascii vers binaire. Chaque caractère de cette chaîne est "swapé". Le résultat est alors comparé à la chaîne passée en argument au programme :

```

1  loc_1000091A:          ; CODE XREF: check_key+448
2  MOVdd     R7, [SP+0x85E0+counter_loop_1]
3  EXTND64   R7, R7
4  MOVnd     R4, [SP+0x85E0+buf_param]
5  ADD64     R4, R7      ; Op1 += Op2
6  MOVbw     [SP+0x85E0+p1], [R4]
7  MOVdd     [SP+0x85E0+p2], [SP+0x85E0+counter_loop_1]
8  CALL32    swap_char
9  MOVbd     R4, [SP+0x85E0+or_sum]
10 MOVqd     R5, SP+0x85E0+convertedstring
11 MOVdd     R1, [SP+0x85E0+counter_loop_1]
12 EXTND64   R1, R1      ; Sign-extend a dword value
13 ADD64     R5, R1      ; Op1 += Op2
14 MOVbw     R5, [R5]    ; Move byte
15 XOR32     R7, R5      ; Op1 ^= Op2
16 OR32      R4, R7      ; Op1 |= Op2
17 MOVbd     [SP+0x85E0+or_sum], R4 ; Move byte
18 MOVdd     R7, [SP+0x85E0+counter_loop_1]
19 MOVqw     R7, R7+1
20 MOVdd     [SP+0x85E0+counter_loop_1], R7
21 MOVdd     R7, [SP+0x85E0+counter_loop_1]
22 CMPI32wgte R7, 0x10
23 JMP8cc    loc_1000091A

```



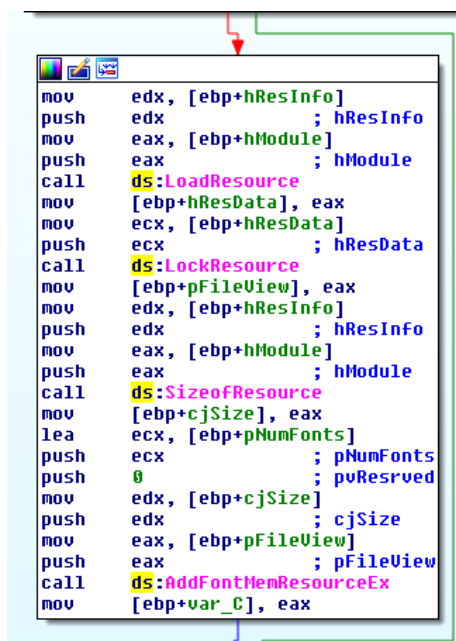
Il suffit d'exécuter le script python suivant :

```
1 stored_key = "cb41dcb1d89746705a7fe998f11acce7".decode("hex")
2
3 def swap_char(input_char, counter):
4     shift = counter % 8
5     c = (input_char << shift) | (input_char >> (8 - shift))
6     c = (~c) & 0xFF
7     return c
8
9 s = ""
10
11 for i in range(16):
12     x = swap_char(ord(stored_key[i]), i)
13     s += chr(x)
14
15 print "key is : %s" % s.encode("hex")
```

La clé solution est : **347d8c72720d6ec7a501583be0bccc0c**.

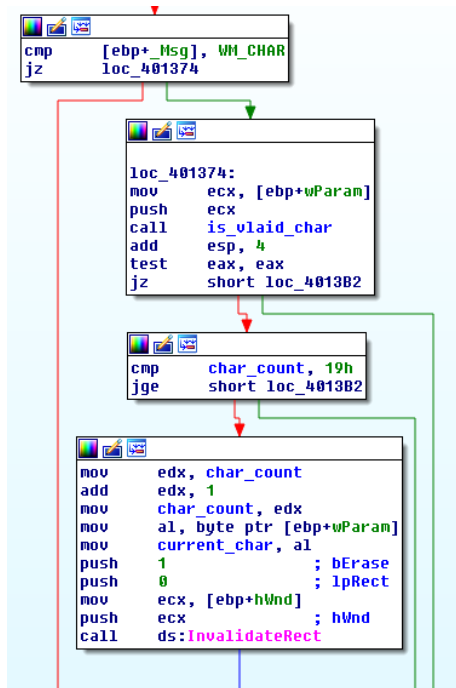
## loader

Cette fois il s'agit d'un programme Windows qui charge une fonte nommée *BizarroSSTIC* se trouvant dans ses ressources :



```
mov     edx, [ebp+hResInfo]
push   edx           ; hResInfo
mov     eax, [ebp+hModule]
push   eax           ; hModule
call   ds:LoadResource
mov     [ebp+hResData], eax
mov     ecx, [ebp+hResData]
push   ecx           ; hResData
call   ds:LockResource
mov     [ebp+pFileView], eax
mov     edx, [ebp+hResInfo]
push   edx           ; hResInfo
mov     eax, [ebp+hModule]
push   eax           ; hModule
call   ds:SizeofResource
mov     [ebp+cjSize], eax
lea    ecx, [ebp+pNumFonts]
push   ecx           ; pNumFonts
push   0             ; pvResrved
mov     edx, [ebp+cjSize]
push   edx           ; cjSize
mov     eax, [ebp+pFileView]
push   eax           ; pFileView
call   ds:AddFontMemResourceEx
mov     [ebp+var_C], eax
```

Puis affiche une fenêtre. La fonction de gestion des messages de la fenêtre effectue un filtrage sur les caractères. Elle n'autorise que les caractères alphanumériques ainsi que `"/=+?"` :



Si le nombre de caractères maximal n'est pas atteint et que le caractère est autorisé il est alors affiché, donc envoyé à la fonte chargée par le noyau.

Il faut analyser la fonte pour comprendre ce qui s'y passe. Pour cela on utilise un décompilateur de fontes (ou un stagiaire). A chaque caractère est associé un bytecode exécuté par la fonte, comme le montre l'exemple suivant :

```

1 <TTGlyph name="L" xMin="42" yMin="0" xMax="557" yMax="574">
2   <contour>
3     <pt x="234" y="488" on="1"/>
4     <pt x="234" y="86" on="1"/>
5     <pt x="467" y="86" on="1"/>
6     <pt x="467" y="277" on="1"/>
7     <pt x="557" y="277" on="1"/>
8     <pt x="557" y="0" on="1"/>
9     <pt x="42" y="0" on="1"/>
10    <pt x="42" y="86" on="1"/>
11    <pt x="140" y="86" on="1"/>
12    <pt x="140" y="488" on="1"/>
13    <pt x="42" y="488" on="1"/>
14    <pt x="42" y="574" on="1"/>
15    <pt x="353" y="574" on="1"/>
16    <pt x="353" y="488" on="1"/>
17  </contour>
18  <instructions><assembly>
19    PUSHB [ ] /* 2 values pushed */
20    99 99
21    RS [ ] /* ReadStore */
22    DUP [ ] /* DuplicateTopStack */
23    PUSHB [ ] /* 1 value pushed */
24    37
25    WS [ ] /* WriteStore */
26    PUSHB [ ] /* 1 value pushed */
27    1
28    ADD [ ] /* Add */
29    WS [ ] /* WriteStore */
30  </assembly></instructions>
31 </TTGlyph>

```

A chacun des caractères du charset base64 est associé une valeur unique. Le code exécuté pour ces caractères stocke cette valeur dans le *store* du moteur de fonte. Seul le caractère "?" a un programme beaucoup plus grand. Il s'agit d'une calculatrice qui récupère chacune des valeurs stockées précédemment dans

le *store* pour les mettre dans la pile puis effectuer des opérations dessus. J'ai décidé d'implémenter les instructions de la machine virtuelle pour exécuter le programme. J'ai initialisé la pile avec des inconnues et arrêté le programme chaque fois que je rencontrais une égalité non satisfaite : lorsqu'une équation n'était pas résolue. Il suffisait alors de résoudre l'équation à la main, d'injecter la valeur de la variable dans mon script et de l'exécuter à nouveau. Voici ce qu'affiche le script en cas d'équation non résolue :

```

Invite de commandes
Executing : MINDEX[ ]
['PUSHB[ ]', '2', 'ADD[ ]', 'PUSHB[ ]', '7', 'MINDEX[ ]', 'PUSHB[ ]', '1', 'SUB[ ]', 'PUSHB[ ]']
Executing : PUSHB[ ]
pushing 2
['ADD[ ]', 'PUSHB[ ]', '7', 'MINDEX[ ]', 'PUSHB[ ]', '1', 'SUB[ ]', 'PUSHB[ ]', '6', 'MINDEX[ ]']
Executing : ADD[ ]
['PUSHB[ ]', '7', 'MINDEX[ ]', 'PUSHB[ ]', '1', 'SUB[ ]', 'PUSHB[ ]', '6', 'MINDEX[ ]', 'PUSHW[ ]']
Executing : PUSHB[ ]
pushing 7
['MINDEX[ ]', 'PUSHB[ ]', '1', 'SUB[ ]', 'PUSHB[ ]', '6', 'MINDEX[ ]', 'PUSHW[ ]', '1463', 'EQ[ ]']
Executing : MINDEX[ ]
['PUSHB[ ]', '1', 'SUB[ ]', 'PUSHB[ ]', '6', 'MINDEX[ ]', 'PUSHW[ ]', '1463', 'EQ[ ]', 'PUSHB[ ]']
Executing : PUSHB[ ]
pushing 1
['SUB[ ]', 'PUSHB[ ]', '6', 'MINDEX[ ]', 'PUSHW[ ]', '1463', 'EQ[ ]', 'PUSHB[ ]', '99', 'RS[ ]']
Executing : SUB[ ]
['PUSHB[ ]', '6', 'MINDEX[ ]', 'PUSHW[ ]', '1463', 'EQ[ ]', 'PUSHB[ ]', '99', 'RS[ ]', 'AND[ ]']
Executing : PUSHB[ ]
pushing 6
['MINDEX[ ]', 'PUSHW[ ]', '1463', 'EQ[ ]', 'PUSHB[ ]', '99', 'RS[ ]', 'AND[ ]', 'PUSHB[ ]', '99']
Executing : MINDEX[ ]
['PUSHW[ ]', '1463', 'EQ[ ]', 'PUSHB[ ]', '99', 'RS[ ]', 'AND[ ]', 'PUSHB[ ]', '99', 'SWAP[ ]']
Executing : PUSHW[ ]
pushing 1463
['EQ[ ]', 'PUSHB[ ]', '99', 'RS[ ]', 'AND[ ]', 'PUSHB[ ]', '99', 'SWAP[ ]', 'WS[ ]', 'PUSHB[ ]']
Executing : EQ[ ]
Equal constraint, integer:1463 == expression:(integer:448 * expression:(integer:1 + expression:(integer:256 * expression:14)))
C:\Users\Julien\Documents\chall_sstic\level2\loader>

```

Voici le script qui a généré la sortie précédente :

```

1 import sys
2 import math
3
4 CELL_TYPE_INTEGER = 0
5 CELL_TYPE_EXPRESSION = 1
6
7 rs_99 = 1
8
9 def F26Dot6(N):
10
11     frac = 0.015625
12     big_part = N >> 6
13
14     small_part = 0
15
16     for i in range(6):
17         print N & 1
18         if (N & 1) == 1:
19             small_part += frac
20             N = N >> 1
21         print "frac", frac
22         frac = frac*2
23
24     return big_part + small_part
25
26 class Cell():
27
28     def __init__(self, type, content):
29         self.type = type
30         self.content = content
31
32     def __str__(self):
33         if ( self.type == CELL_TYPE_INTEGER):
34             return "integer:%d" % self.content
35         else:
36             return "expression:%s" % self.content
37
38 class Stack():
39
40     def __init__(self):
41         self.storage = []
42
43     def push(self, element):
44         self.storage = [element] + self.storage
45
46     def pop(self):
47         element = self.storage[0]
48         self.storage = self.storage[1:]

```

```

49     return element
50
51
52 def get_ith_element(self, i):
53     if (i == 0) or (i > len(self.storage)):
54         print "get_ith_element error : invalid index %d" % i
55         quit()
56
57     if (i == 1):
58         return self.pop()
59
60     if (i == len(self.storage)):
61         element = self.storage[-1]
62         self.storage = self.storage[:-1]
63     else:
64         i = i - 1
65         element = self.storage[i]
66         self.storage = self.storage[:i] + self.storage[i+1:]
67     return element
68
69 def __str__(self):
70     s = ""
71     for element in self.storage:
72         print "\t%s" % element
73     return s
74
75 def push(i_stream, stack):
76     elt = i_stream[1]
77     print "pushing %s" % elt
78     stack.push(Cell(0,int(elt)))
79
80     return 2
81
82 def mul(i_stream, stack):
83     elt1 = stack.pop()
84     elt2 = stack.pop()
85
86     if (elt1.type == elt2.type == CELL_TYPE_INTEGER):
87
88         elt1 = float(elt1.content)
89         elt2 = float(elt2.content)
90
91         (part, whole) = math.modf(elt1 * elt2 / float(64))
92
93         result = Cell(CELL_TYPE_INTEGER, int(whole))
94     else:
95         result = Cell(CELL_TYPE_EXPRESSION, "%s * %s" % (elt1, elt2))
96
97     stack.push(result)
98
99     return 1
100
101 def add(i_stream, stack):
102     elt1 = stack.pop()
103     elt2 = stack.pop()
104
105     if (elt1.type == elt2.type == CELL_TYPE_INTEGER):
106         result = Cell(CELL_TYPE_INTEGER, elt1.content + elt2.content)
107     else:
108         result = Cell(CELL_TYPE_EXPRESSION, "%s + %s" % (elt1, elt2))
109
110     stack.push(result)
111
112     return 1
113
114 def sub(i_stream, stack):
115     elt1 = stack.pop()
116     elt2 = stack.pop()
117
118     if (elt1.type == elt2.type == CELL_TYPE_INTEGER):
119         result = Cell(CELL_TYPE_INTEGER, elt2.content - elt1.content)
120     else:
121         result = Cell(CELL_TYPE_EXPRESSION, "%s - %s" % (elt2, elt1))
122
123     stack.push(result)
124
125     return 1
126
127
128 def roll(i_stream, stack):
129     elt = stack.get_ith_element(3)
130     print elt
131     stack.push(elt)
132     return 1
133
134 def minindex(i_stream, stack):
135     elt = stack.pop()
136
137     if (elt.type == CELL_TYPE_EXPRESSION):
138         print "Cannot minindex on expression"
139         exit()
140
141     index = elt.content
142
143     elt = stack.get_ith_element(index)
144     stack.push(elt)
145     return 1
146
147 def eq(instructions, stack):

```

```

148
149     elt1 = stack.pop()
150     elt2 = stack.pop()
151
152     if ( elt1.type != elt2.type ):
153         print "Equal constraint, %s == %s" % (elt1, elt2)
154         quit()
155     else:
156         if (elt1.content != elt2.content ):
157             print "Unsatisfied constraint : %s == %s" % (elt1,elt2)
158             stack.push(Cell(CELL_TYPE_INTEGER,0))
159             quit()
160         else:
161             stack.push(Cell(CELL_TYPE_INTEGER,1))
162
163     return 1
164
165 def rs(instructions, stack):
166     index = stack.pop()
167     if (index.type != CELL_TYPE_INTEGER):
168         print "RS invalid param : %s" % index
169         quit()
170
171     if ( index.content != 99):
172         print "RS unexpected index : %d" % index
173         quit()
174
175     stack.push(Cell(CELL_TYPE_INTEGER,rs_99))
176
177     return 1
178
179 def ws(instructions, stack):
180     value = stack.pop()
181     index = stack.pop()
182
183     if (value.type != CELL_TYPE_INTEGER):
184         print "RS invalid param : %s" % index
185         quit()
186
187
188     if (index.type != CELL_TYPE_INTEGER):
189         print "RS invalid param : %s" % index
190         quit()
191
192     if ( index.content != 99):
193         print "RS unexpected index : %d" % index
194         quit()
195
196     rs_99 = value.content
197
198
199     return 1
200
201
202 def _and(instructions,stack):
203     elt1 = stack.pop()
204     elt2 = stack.pop()
205
206
207     if ( not(elt1.type == elt2.type == CELL_TYPE_INTEGER) ):
208         print "And : error on input elements"
209         exit()
210
211     stack.push( Cell(CELL_TYPE_INTEGER, elt1.content & elt2.content) )
212
213     return 1
214
215 def swap(instructions, stack):
216
217     elt1 = stack.pop()
218     elt2 = stack.pop()
219
220     stack.push(elt1)
221     stack.push(elt2)
222
223
224     return 1
225
226 def execute(handlers, instructions, stack):
227
228     while (1):
229
230         print instructions[:10]
231
232         inst = instructions[0]
233
234         if (inst not in handlers.keys()):
235             print "Error : unknown instruction %s" % inst
236             quit()
237
238         func = handlers[inst]
239
240         print "Executing : %s" % inst
241
242         move = func(instructions, stack)
243
244         instructions = instructions[move:]
245
246

```

```

247 handlers = {}
248 handlers["PUSHB[ ]"] = push
249 handlers["PUSHW[ ]"] = push
250 handlers["MINDEX[ ]"] = minindex
251 handlers["ROLL[ ]"] = roll
252 handlers["MUL[ ]"] = mul
253 handlers["ROLL[ ]"] = roll
254 handlers["EQ[ ]"] = eq
255 handlers["RS[ ]"] = rs
256 handlers["AND[ ]"] = _and
257 handlers["SWAP[ ]"] = swap
258 handlers["WS[ ]"] = ws
259 handlers["ADD[ ]"] = add
260 handlers["SUB[ ]"] = sub
261
262 instructions = []
263 values = {}
264 values[0] = 56
265 values[1] = 5
266 values[2] = 19
267 values[3] = 63
268 values[5] = 26
269 values[7] = 35
270 values[8] = 39
271 values[11] = 27
272 values[12] = 31
273 values[13] = 48
274 values[15] = 2
275 values[18] = 21
276 values[19] = 47
277 values[21] = 22
278
279 #Ajout des valeurs sur la stack initiale
280 #Si la valeur est connue on pousse un nombre, dans le cas contraire une
281 #expression.
282 stack = Stack()
283 for i in range(22):
284     if (i in values.keys()):
285         stack.push( Cell(CELL_TYPE_INTEGER, values[i]) )
286     else:
287         stack.push( Cell(CELL_TYPE_EXPRESSION, "%i%d" % i) )
288
289 instructions.append("ROLL[ ]")
290 instructions.append("PUSHW[ ]")
291 instructions.append("256")
292 instructions.append("MUL[ ]")
293 instructions.append("PUSHB[ ]")
294 instructions.append("19")
295 instructions.append("MINDEX[ ]")
296 instructions.append("PUSHW[ ]")
297 instructions.append("256")
298 instructions.append("MUL[ ]")
299 instructions.append("PUSHB[ ]")
300 instructions.append("20")
301 instructions.append("MINDEX[ ]")
302 instructions.append("PUSHW[ ]")
303 instructions.append("19")
304 instructions.append("EQ[ ]")
305 instructions.append("PUSHB[ ]")
306 instructions.append("99")
307 instructions.append("RS[ ]")
308 instructions.append("AND[ ]")
309 instructions.append("PUSHB[ ]")
310 instructions.append("99")
311 instructions.append("SWAP[ ]")
312 instructions.append("WS[ ]")
313 instructions.append("PUSHB[ ]")
314 instructions.append("10")
315 instructions.append("MINDEX[ ]")
316 instructions.append("PUSHB[ ]")
317 instructions.append("6")
318 instructions.append("ADD[ ]")
319 instructions.append("PUSHB[ ]")
320 instructions.append("4")
321 instructions.append("MINDEX[ ]")
322 instructions.append("PUSHB[ ]")
323 instructions.append("6")
324 instructions.append("ADD[ ]")
325 instructions.append("PUSHB[ ]")
326 instructions.append("10")
327 instructions.append("MINDEX[ ]")
328 instructions.append("PUSHW[ ]")
329 instructions.append("320")
330 instructions.append("MUL[ ]")
331 instructions.append("PUSHB[ ]")
332 instructions.append("11")
333 instructions.append("MINDEX[ ]")
334 instructions.append("PUSHW[ ]")
335 instructions.append("384")
336 instructions.append("MUL[ ]")
337 instructions.append("PUSHW[ ]")
338 instructions.append("186")
339 instructions.append("EQ[ ]")
340 instructions.append("PUSHB[ ]")
341 instructions.append("99")
342 instructions.append("RS[ ]")
343 instructions.append("AND[ ]")
344 instructions.append("PUSHB[ ]")
345

```

```

346 instructions.append("99")
347 instructions.append("SWAP[ ]")
348 instructions.append("WS[ ]")
349 instructions.append("PUSHB[ ]")
350 instructions.append("18")
351 instructions.append("MINDEX[ ]")
352 instructions.append("PUSHW[ ]")
353 instructions.append("63")
354 instructions.append("EQ[ ]")
355 instructions.append("PUSHB[ ]")
356 instructions.append("99")
357 instructions.append("RS[ ]")
358 instructions.append("AND[ ]")
359 instructions.append("PUSHB[ ]")
360 instructions.append("99")
361 instructions.append("SWAP[ ]")
362 instructions.append("WS[ ]")
363 instructions.append("ROLL[ ]")
364 instructions.append("PUSHW[ ]")
365 instructions.append("320")
366 instructions.append("MUL[ ]")
367 instructions.append("PUSHB[ ]")
368 instructions.append("14")
369 instructions.append("MINDEX[ ]")
370 instructions.append("PUSHW[ ]")
371 instructions.append("39")
372 instructions.append("EQ[ ]")
373 instructions.append("PUSHB[ ]")
374 instructions.append("99")
375 instructions.append("RS[ ]")
376 instructions.append("AND[ ]")
377 instructions.append("PUSHB[ ]")
378 instructions.append("99")
379 instructions.append("SWAP[ ]")
380 instructions.append("WS[ ]")
381 instructions.append("PUSHB[ ]")
382 instructions.append("7")
383 instructions.append("SUB[ ]")
384 instructions.append("PUSHB[ ]")
385 instructions.append("14")
386 instructions.append("MINDEX[ ]")
387 instructions.append("PUSHW[ ]")
388 instructions.append("35")
389 instructions.append("EQ[ ]")
390 instructions.append("PUSHB[ ]")
391 instructions.append("99")
392 instructions.append("RS[ ]")
393 instructions.append("AND[ ]")
394 instructions.append("PUSHB[ ]")
395 instructions.append("99")
396 instructions.append("SWAP[ ]")
397 instructions.append("WS[ ]")
398 instructions.append("PUSHB[ ]")
399 instructions.append("13")
400 instructions.append("MINDEX[ ]")
401 instructions.append("PUSHB[ ]")
402 instructions.append("2")
403 instructions.append("ADD[ ]")
404 instructions.append("PUSHB[ ]")
405 instructions.append("6")
406 instructions.append("MINDEX[ ]")
407 instructions.append("PUSHB[ ]")
408 instructions.append("6")
409 instructions.append("ADD[ ]")
410 instructions.append("PUSHB[ ]")
411 instructions.append("16")
412 instructions.append("MINDEX[ ]")
413 instructions.append("PUSHW[ ]")
414 instructions.append("256")
415 instructions.append("MUL[ ]")
416 instructions.append("PUSHB[ ]")
417 instructions.append("17")
418 instructions.append("MINDEX[ ]")
419 instructions.append("PUSHB[ ]")
420 instructions.append("3")
421 instructions.append("SUB[ ]")
422 instructions.append("PUSHB[ ]")
423 instructions.append("8")
424 instructions.append("MINDEX[ ]")
425 instructions.append("PUSHB[ ]")
426 instructions.append("1")
427 instructions.append("ADD[ ]")
428 instructions.append("PUSHB[ ]")
429 instructions.append("11")
430 instructions.append("MINDEX[ ]")
431 instructions.append("PUSHB[ ]")
432 instructions.append("2")
433 instructions.append("ADD[ ]")
434 instructions.append("PUSHB[ ]")
435 instructions.append("10")
436 instructions.append("MINDEX[ ]")
437 instructions.append("PUSHB[ ]")
438 instructions.append("7")
439 instructions.append("SUB[ ]")
440 instructions.append("SWAP[ ]")
441 instructions.append("PUSHB[ ]")
442 instructions.append("4")
443 instructions.append("ADD[ ]")
444 instructions.append("PUSHB[ ]")

```

```

445 instructions.append("8")
446 instructions.append("MINDEX [ ]")
447 instructions.append("PUSHW [ ]")
448 instructions.append("263")
449 instructions.append("EQ [ ]")
450 instructions.append("PUSHB [ ]")
451 instructions.append("99")
452 instructions.append("RS [ ]")
453 instructions.append("AND [ ]")
454 instructions.append("PUSHB [ ]")
455 instructions.append("99")
456 instructions.append("SWAP [ ]")
457 instructions.append("WS [ ]")
458 instructions.append("PUSHB [ ]")
459 instructions.append("16")
460 instructions.append("MINDEX [ ]")
461 instructions.append("PUSHW [ ]")
462 instructions.append("26")
463 instructions.append("EQ [ ]")
464 instructions.append("PUSHB [ ]")
465 instructions.append("99")
466 instructions.append("RS [ ]")
467 instructions.append("AND [ ]")
468 instructions.append("PUSHB [ ]")
469 instructions.append("99")
470 instructions.append("SWAP [ ]")
471 instructions.append("WS [ ]")
472 instructions.append("PUSHB [ ]")
473 instructions.append("7")
474 instructions.append("MINDEX [ ]")
475 instructions.append("PUSHB [ ]")
476 instructions.append("7")
477 instructions.append("ADD [ ]")
478 instructions.append("PUSHB [ ]")
479 instructions.append("9")
480 instructions.append("MINDEX [ ]")
481 instructions.append("PUSHW [ ]")
482 instructions.append("28")
483 instructions.append("EQ [ ]")
484 instructions.append("PUSHB [ ]")
485 instructions.append("99")
486 instructions.append("RS [ ]")
487 instructions.append("AND [ ]")
488 instructions.append("PUSHB [ ]")
489 instructions.append("99")
490 instructions.append("SWAP [ ]")
491 instructions.append("WS [ ]")
492 instructions.append("PUSHB [ ]")
493 instructions.append("7")
494 instructions.append("MINDEX [ ]")
495 instructions.append("PUSHW [ ]")
496 instructions.append("192")
497 instructions.append("MUL [ ]")
498 instructions.append("PUSHB [ ]")
499 instructions.append("8")
500 instructions.append("MINDEX [ ]")
501 instructions.append("PUSHB [ ]")
502 instructions.append("2")
503 instructions.append("SUB [ ]")
504 instructions.append("PUSHB [ ]")
505 instructions.append("12")
506 instructions.append("MINDEX [ ]")
507 instructions.append("PUSHB [ ]")
508 instructions.append("6")
509 instructions.append("SUB [ ]")
510 instructions.append("PUSHB [ ]")
511 instructions.append("10")
512 instructions.append("MINDEX [ ]")
513 instructions.append("PUSHW [ ]")
514 instructions.append("21")
515 instructions.append("EQ [ ]")
516 instructions.append("PUSHB [ ]")
517 instructions.append("99")
518 instructions.append("RS [ ]")
519 instructions.append("AND [ ]")
520 instructions.append("PUSHB [ ]")
521 instructions.append("99")
522 instructions.append("SWAP [ ]")
523 instructions.append("WS [ ]")
524 instructions.append("PUSHB [ ]")
525 instructions.append("13")
526 instructions.append("MINDEX [ ]")
527 instructions.append("PUSHB [ ]")
528 instructions.append("5")
529 instructions.append("ADD [ ]")
530 instructions.append("PUSHB [ ]")
531 instructions.append("9")
532 instructions.append("MINDEX [ ]")
533 instructions.append("PUSHW [ ]")
534 instructions.append("53")
535 instructions.append("EQ [ ]")
536 instructions.append("PUSHB [ ]")
537 instructions.append("99")
538 instructions.append("RS [ ]")
539 instructions.append("AND [ ]")
540 instructions.append("PUSHB [ ]")
541 instructions.append("99")
542 instructions.append("SWAP [ ]")
543 instructions.append("WS [ ]")

```



```

544 instructions.append("SWAP [ ]")
545 instructions.append("PUSHW [ ]")
546 instructions.append("21")
547 instructions.append("EQ [ ]")
548 instructions.append("PUSHB [ ]")
549 instructions.append("99")
550 instructions.append("RS [ ]")
551 instructions.append("AND [ ]")
552 instructions.append("PUSHB [ ]")
553 instructions.append("99")
554 instructions.append("SWAP [ ]")
555 instructions.append("WS [ ]")
556 instructions.append("PUSHB [ ]")
557 instructions.append("7")
558 instructions.append("MINDEX [ ]")
559 instructions.append("PUSHW [ ]")
560 instructions.append("448")
561 instructions.append("MUL [ ]")
562 instructions.append("PUSHB [ ]")
563 instructions.append("11")
564 instructions.append("MINDEX [ ]")
565 instructions.append("PUSHB [ ]")
566 instructions.append("1")
567 instructions.append("SUB [ ]")
568 instructions.append("PUSHB [ ]")
569 instructions.append("4")
570 instructions.append("MINDEX [ ]")
571 instructions.append("PUSHW [ ]")
572 instructions.append("320")
573 instructions.append("MUL [ ]")
574 instructions.append("PUSHB [ ]")
575 instructions.append("6")
576 instructions.append("MINDEX [ ]")
577 instructions.append("PUSHW [ ]")
578 instructions.append("64")
579 instructions.append("MUL [ ]")
580 instructions.append("PUSHB [ ]")
581 instructions.append("6")
582 instructions.append("MINDEX [ ]")
583 instructions.append("PUSHW [ ]")
584 instructions.append("582")
585 instructions.append("EQ [ ]")
586 instructions.append("PUSHB [ ]")
587 instructions.append("99")
588 instructions.append("RS [ ]")
589 instructions.append("AND [ ]")
590 instructions.append("PUSHB [ ]")
591 instructions.append("99")
592 instructions.append("SWAP [ ]")
593 instructions.append("WS [ ]")
594 instructions.append("PUSHB [ ]")
595 instructions.append("8")
596 instructions.append("MINDEX [ ]")
597 instructions.append("PUSHW [ ]")
598 instructions.append("20")
599 instructions.append("EQ [ ]")
600 instructions.append("PUSHB [ ]")
601 instructions.append("99")
602 instructions.append("RS [ ]")
603 instructions.append("AND [ ]")
604 instructions.append("PUSHB [ ]")
605 instructions.append("99")
606 instructions.append("SWAP [ ]")
607 instructions.append("WS [ ]")
608 instructions.append("PUSHB [ ]")
609 instructions.append("9")
610 instructions.append("MINDEX [ ]")
611 instructions.append("PUSHW [ ]")
612 instructions.append("2")
613 instructions.append("EQ [ ]")
614 instructions.append("PUSHB [ ]")
615 instructions.append("99")
616 instructions.append("RS [ ]")
617 instructions.append("AND [ ]")
618 instructions.append("PUSHB [ ]")
619 instructions.append("99")
620 instructions.append("SWAP [ ]")
621 instructions.append("WS [ ]")
622 instructions.append("PUSHW [ ]")
623 instructions.append("320")
624 instructions.append("MUL [ ]")
625 instructions.append("PUSHB [ ]")
626 instructions.append("6")
627 instructions.append("MINDEX [ ]")
628 instructions.append("PUSHB [ ]")
629 instructions.append("2")
630 instructions.append("ADD [ ]")
631 instructions.append("PUSHB [ ]")
632 instructions.append("7")
633 instructions.append("MINDEX [ ]")
634 instructions.append("PUSHB [ ]")
635 instructions.append("1")
636 instructions.append("SUB [ ]")
637 instructions.append("PUSHB [ ]")
638 instructions.append("6")
639 instructions.append("MINDEX [ ]")
640 instructions.append("PUSHW [ ]")
641 instructions.append("1463")
642 instructions.append("EQ [ ]")

```

```

643 instructions.append("PUSHB [ ]")
644 instructions.append("99")
645 instructions.append("RS [ ]")
646 instructions.append("AND [ ]")
647 instructions.append("PUSHB [ ]")
648 instructions.append("99")
649 instructions.append("SWAP [ ]")
650 instructions.append("WS [ ]")
651 instructions.append("PUSHB [ ]")
652 instructions.append("4")
653 instructions.append("MINDEX [ ]")
654 instructions.append("PUSHW [ ]")
655 instructions.append("1415")
656 instructions.append("EQ [ ]")
657 instructions.append("PUSHB [ ]")
658 instructions.append("99")
659 instructions.append("RS [ ]")
660 instructions.append("AND [ ]")
661 instructions.append("PUSHB [ ]")
662 instructions.append("99")
663 instructions.append("SWAP [ ]")
664 instructions.append("WS [ ]")
665 instructions.append("PUSHB [ ]")
666 instructions.append("5")
667 instructions.append("MINDEX [ ]")
668 instructions.append("PUSHB [ ]")
669 instructions.append("1")
670 instructions.append("SUB [ ]")
671 instructions.append("ROLL [ ]")
672 instructions.append("PUSHW [ ]")
673 instructions.append("70")
674 instructions.append("EQ [ ]")
675 instructions.append("PUSHB [ ]")
676 instructions.append("99")
677 instructions.append("RS [ ]")
678 instructions.append("AND [ ]")
679 instructions.append("PUSHB [ ]")
680 instructions.append("99")
681 instructions.append("SWAP [ ]")
682 instructions.append("WS [ ]")
683 instructions.append("PUSHB [ ]")
684 instructions.append("3")
685 instructions.append("SUB [ ]")
686 instructions.append("SWAP [ ]")
687 instructions.append("PUSHW [ ]")
688 instructions.append("256")
689 instructions.append("MUL [ ]")
690 instructions.append("PUSHB [ ]")
691 instructions.append("4")
692 instructions.append("MINDEX [ ]")
693 instructions.append("PUSHW [ ]")
694 instructions.append("32")
695 instructions.append("EQ [ ]")
696 instructions.append("PUSHB [ ]")
697 instructions.append("99")
698 instructions.append("RS [ ]")
699 instructions.append("AND [ ]")
700 instructions.append("PUSHB [ ]")
701 instructions.append("99")
702 instructions.append("SWAP [ ]")
703 instructions.append("WS [ ]")
704 instructions.append("PUSHB [ ]")
705 instructions.append("4")
706 instructions.append("MINDEX [ ]")
707 instructions.append("PUSHB [ ]")
708 instructions.append("2")
709 instructions.append("SUB [ ]")
710 instructions.append("PUSHW [ ]")
711 instructions.append("5")
712 instructions.append("EQ [ ]")
713 instructions.append("PUSHB [ ]")
714 instructions.append("99")
715 instructions.append("RS [ ]")
716 instructions.append("AND [ ]")
717 instructions.append("PUSHB [ ]")
718 instructions.append("99")
719 instructions.append("SWAP [ ]")
720 instructions.append("WS [ ]")
721 instructions.append("SWAP [ ]")
722 instructions.append("PUSHB [ ]")
723 instructions.append("5")
724 instructions.append("ADD [ ]")
725 instructions.append("PUSHB [ ]")
726 instructions.append("6")
727 instructions.append("SUB [ ]")
728 instructions.append("ROLL [ ]")
729 instructions.append("PUSHW [ ]")
730 instructions.append("90")
731 instructions.append("EQ [ ]")
732 instructions.append("PUSHB [ ]")
733 instructions.append("99")
734 instructions.append("RS [ ]")
735 instructions.append("AND [ ]")
736 instructions.append("PUSHB [ ]")
737 instructions.append("99")
738 instructions.append("SWAP [ ]")
739 instructions.append("WS [ ]")
740 instructions.append("PUSHW [ ]")
741 instructions.append("64")

```

```

742 instructions.append("MUL[ ]")
743 instructions.append("SWAP[ ]")
744 instructions.append("PUSHB[ ]")
745 instructions.append("3")
746 instructions.append("ADD[ ]")
747 instructions.append("SWAP[ ]")
748 instructions.append("PUSHW[ ]")
749 instructions.append("3")
750 instructions.append("EQ[ ]")
751 instructions.append("PUSHB[ ]")
752 instructions.append("99")
753 instructions.append("RS[ ]")
754 instructions.append("AND[ ]")
755 instructions.append("PUSHB[ ]")
756 instructions.append("99")
757 instructions.append("SWAP[ ]")
758 instructions.append("WS[ ]")
759 instructions.append("PUSHW[ ]")
760 instructions.append("256")
761 instructions.append("MUL[ ]")
762 instructions.append("PUSHB[ ]")
763 instructions.append("2")
764 instructions.append("SUB[ ]")
765 instructions.append("PUSHW[ ]")
766 instructions.append("320")
767 instructions.append("MUL[ ]")
768 instructions.append("PUSHW[ ]")
769 instructions.append("1970")
770 instructions.append("EQ[ ]")
771 instructions.append("PUSHB[ ]")
772 instructions.append("99")
773 instructions.append("RS[ ]")
774 instructions.append("AND[ ]")
775 instructions.append("PUSHB[ ]")
776 instructions.append("99")
777 instructions.append("SWAP[ ]")
778 instructions.append("WS[ ]")
779 instructions.append("PUSHB[ ]")
780 instructions.append("99")
781 instructions.append("RS[ ]")
782 instructions.append("IP[ ]")
783 instructions.append("PUSHB[ ] /* 5 values pushed */")
784 instructions.append("0 1 2 3 4")
785 instructions.append("PUSHB[ ] /* 5 values pushed */")
786 instructions.append("5 6 7 8 9")
787 instructions.append("ELSE[ ] /* Else */")
788 instructions.append("PUSHB[ ] /* 5 values pushed */")
789 instructions.append("60 61 62 63 64")
790 instructions.append("PUSHB[ ] /* 5 values pushed */")
791 instructions.append("65 66 67 68 69")
792 instructions.append("EIF[ ] /* EndIf */")
793 instructions.append("SVTCA[0] /* SetFPVectorToAxis */")
794 instructions.append("PUSHB[ ]")
795 instructions.append("0")
796 instructions.append("SCFS[ ] /* SetCoordFromStackFP */")
797 instructions.append("PUSHB[ ]")
798 instructions.append("0")
799 instructions.append("SCFS[ ] /* SetCoordFromStackFP */")
800 instructions.append("PUSHB[ ]")
801 instructions.append("0")
802 instructions.append("SCFS[ ] /* SetCoordFromStackFP */")
803 instructions.append("PUSHB[ ]")
804 instructions.append("0")
805 instructions.append("SCFS[ ] /* SetCoordFromStackFP */")
806 instructions.append("PUSHB[ ]")
807 instructions.append("0")
808 instructions.append("SCFS[ ] /* SetCoordFromStackFP */")
809 instructions.append("PUSHB[ ]")
810 instructions.append("0")
811 instructions.append("SCFS[ ] /* SetCoordFromStackFP */")
812 instructions.append("PUSHB[ ]")
813 instructions.append("0")
814 instructions.append("SCFS[ ] /* SetCoordFromStackFP */")
815 instructions.append("PUSHB[ ]")
816 instructions.append("0")
817 instructions.append("SCFS[ ] /* SetCoordFromStackFP */")
818 instructions.append("PUSHB[ ]")
819 instructions.append("0")
820 instructions.append("SCFS[ ] /* SetCoordFromStackFP */")
821 instructions.append("PUSHB[ ]")
822 instructions.append("0")
823 instructions.append("SCFS[ ] /* SetCoordFromStackFP */")
824
825 execute(handlers, instructions, stack)

```

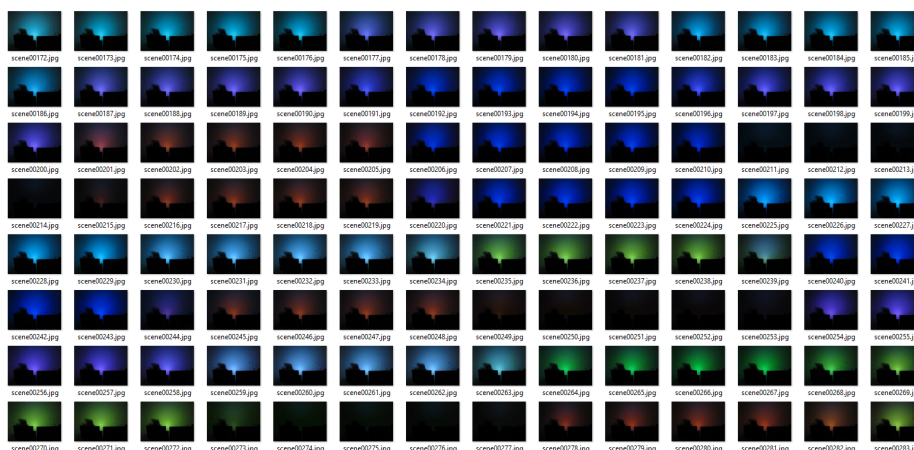
On obtient finalement la clé encodée en base64 : "4ft/0AdJNjHBFW5ch+vVGw".  
 Un fois le bourrage ajouté et la chaîne décodée on obtient la clé : **e1fb7fd007493631c1156e5c87ebd51b**.  
 On peut enfin passer au niveau 3 et rencontrer les gens qui ont popularisé le hacking en France (ou pas).

## Niveau 3

### Video

Cette épreuve semble tout droit sortie du *Laboratoire Cool et Vraiment Opérationnel*. Un économiseur d'écran qui exfiltre des données à une certaine heure de la nuit (oui aussi j'ai mis du temps à m'en rendre compte). Ce programme (l'économiseur d'écran donc), lit une clé dans la base de registre et l'exfiltre en faisant changer l'écran de couleur. Après avoir reversé (et pressé F5) j'ai compris que la clé de registre était "xorée" avec une chaîne en dur dans le binaire. Puis chaque octet est converti en base 6. Chaque chiffre du résultat de cette conversion est transformé en une couleur. Ce après y avoir ajouté le résultat de la conversion précédente, modulo 8. Certainement afin d'éviter que deux couleurs identiques ne se suivent dans la séquence.

J'ai ensuite échantillonné frame par frame la vidéo fournie grâce à VLC :



Et finalement j'ai exécuté mon script python pour retrouver la clé, après avoir perdu quelques dixièmes.

```
1 import zlib
2 import struct
3
4 index_to_color = {}
5 index_to_color[0] = "rouge"
6 index_to_color[1] = "vert"
7 index_to_color[2] = "jaune"
8 index_to_color[3] = "bleu"
9 index_to_color[4] = "violet"
10 index_to_color[5] = "cyan"
11 index_to_color[6] = "blanc"
12 index_to_color[7] = "noir"
13
14 color_to_index = {}
15 color_to_index["rouge"] = 0
16 color_to_index["vert"] = 1
17 color_to_index["jaune"] = 2
18 color_to_index["bleu"] = 3
19 color_to_index["violet"] = 4
20 color_to_index["cyan"] = 5
21 color_to_index["blanc"] = 6
22 color_to_index["noir"] = 7
23
24 #Chaîne hardcodée du binaire
25 xor_string = "2830
26             a43f6d280423362a32dcad0ba04be8201f64840af4c4c78a8dc0a2c44019a143823814fd6c90e07e2a40dfd3f23e".decode(
27             "hex")
28
29 def generate_byte_string(video_colors):
30     len_array = len(video_colors)
31     if (len_array % 3) != 0:
```

```

32     print "da fuck ?"
33     return None
34
35     out_buf = ""
36
37     previous_color = 6
38     i = 0
39     j = 0
40
41     while(i < len_array):
42
43         #On prend un triplet de couleurs pour decode un byte
44         triplet = video_colors[:3]
45         video_colors = video_colors[3:]
46
47         parts = []
48
49         for color in triplet:
50
51             encoded_data = (color_to_index[color] - previous_color) % 8
52             previous_color = color_to_index[color]
53
54             x = (encoded_data - 1) % 7
55
56             parts.append(x)
57
58         parts.reverse()
59
60         #Conversion en base 6
61         byte = 0
62         for part in parts:
63             byte += byte*6 + part
64
65         #Finalement on xor avec un octet de la chaine hardcodee
66         byte = byte ^ ord( xor_string[j] )
67
68         i += 3
69         j += 1
70
71         out_buf += chr(byte)
72
73     return out_buf
74
75 #Video notees patiemment sans perdre (top) de dixiemes
76 video_colors = ["cyan", "violet", "cyan"]
77 video_colors += ["violet", "bleu", "violet"]
78 video_colors += ["rouge", "bleu", "noir"]
79 video_colors += ["rouge", "bleu", "cyan"]
80 video_colors += ["blanc", "jaune", "bleu"]
81 video_colors += ["rouge", "noir", "violet"]
82 video_colors += ["blanc", "vert", "jaune"]
83 video_colors += ["noir", "rouge", "jaune"]
84 video_colors += ["cyan", "blanc", "noir"]
85 video_colors += ["cyan", "noir", "vert"]
86 video_colors += ["blanc", "jaune", "bleu"]
87 video_colors += ["jaune", "bleu", "violet"]
88 video_colors += ["bleu", "violet", "noir"]
89 video_colors += ["blanc", "bleu", "cyan"]
90 video_colors += ["bleu", "vert", "bleu"]
91 video_colors += ["blanc", "bleu", "blanc"]
92 video_colors += ["rouge", "blanc", "vert"]
93 video_colors += ["cyan", "rouge", "jaune"]
94 video_colors += ["cyan", "vert", "bleu"]
95 video_colors += ["rouge", "vert", "cyan"]
96 video_colors += ["rouge", "jaune", "violet"]
97 video_colors += ["blanc", "vert", "violet"]
98 video_colors += ["cyan", "jaune", "blanc"]
99 video_colors += ["noir", "rouge", "cyan"]
100 video_colors += ["blanc", "noir", "cyan"]
101 video_colors += ["noir", "bleu", "rouge"]
102 video_colors += ["blanc", "violet", "noir"]
103 video_colors += ["vert", "rouge", "violet"]
104
105
106 out_buf = generate_byte_string(video_colors)
107 #On ignore les 4 octets de taille
108 out_buf = out_buf[4:]
109
110 s = zlib.decompress(out_buf)
111 print "key is : %s" % s.encode("hex")

```

La solution est : **5311371672ba0179fa3e918a83bedeb4**

## Usb

Pour cette épreuve nous sont fournies : l'image d'une clé USB ainsi qu'un binaire. Ce binaire embarque dans ses ressources un driver qui est écrit sur le disque puis chargé.

Le programme *userland* est chargé transmettre les fichiers du répertoire *C:\SSTIC* au driver. Pour cela, le programme *userland* communique en remplaçant le champ à l'offset 0x2C8 de sa structure *KUSER\_SHARE\_DATA*, partagée avec le noyau, par un pointeur vers une structure où ce dernier stocke les informations relatives au fichier transmis :

```

1 typedef struct FileStruct {
2     Event ReadyToHashFile
3     Event FileHashed
4     Event Event3
5     Event Event4
6     DWORD FileSize
7     char Md5Sum[16]
8     PVOID FileBuffer
9 } FileStruct;

```

Le programme *userland* est chargé de détecter le branchement d'un clé USB. Lorsque celle-ci est branché, un message de type *DBT\_DEVICEARRIVAL* est envoyé à la fonction de gestion de la fenêtre du programme qui transmet alors le nom du volume monté au noyau grâce à la fonction *ZwSetInformationProcess*.

De son côté le noyau récupère les fichiers un par un et les chiffre en RC4 avec une clé générée aléatoirement pour chaque fichier. Les fichiers chiffrés sont ensuite stockés dans une liste chaînée.

```

E8 61 25 00 00      call     memset
48 8B 35 22 40 20 00  mov     rsi, cs:MappedStructure
48 8D 9C 24 30 01 00 00  lea    rbx, [rsp+158h+RC4_key]
BF 04 00 00 00      mov     edi, 4

loc_11173:
48 8D 0D B2 3F 00 00  lea    rcx, seed
FF 15 F0 2E 00 00      call   cs:RtlRandomEx
89 03                mov    [rbx], eax
48 83 C3 04          add    rbx, 4
48 83 EF 01          sub    rdi, 1
75 E7                jnz   short loc_11173

BF 10 00 00 00      mov    edi, 10h
48 8D 94 24 30 01 00 00  lea    rdx, [rsp+158h+RC4_key]
48 8D 4C 24 20        lea    rcx, [rsp+158h+RC4_context]
44 8B C7             mov    r8d, edi
E8 DE 10 00 00      call   RC4_set_key
44 8B AC 20          mov    r9d, [rsi+20h] ; FileSize
4C 8D A6 34          lea    r8, [rsi+34h] ; FileBuffer
48 8D 56 34          lea    rdx, [rsi+34h] ; FileBuffer
48 8D AC 24 20        lea    rcx, [rsp+158h+RC4_context]
E8 3C 11 00 00      call   RC4_crypt
83 3D 65 3F 00 00 1E  cmp    cs:Counter, 1Eh
73 5A                jnb   short loc_1121F

```

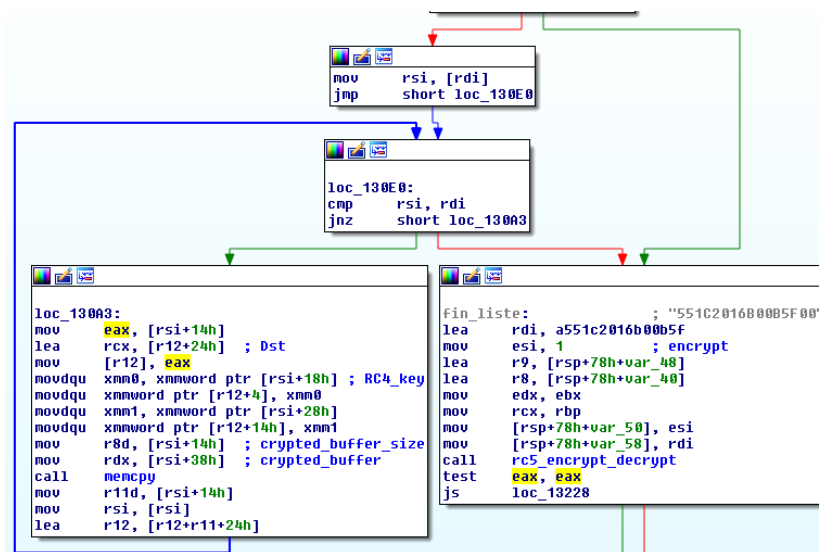
Une fois ce processus terminé, le driver alloue un grand buffer où il copie chaque fichier chiffré, précédé de l'en-tête suivante :

```

1 typedef struct CryptedFileHeader {
2     DWORD FileSize
3     char Rc4Key[16]
4     char Md5Sum[16]
5 } CryptedFileHeader;

```

Comme le montre la capture suivante :



Ce gros buffer est ensuite chiffré en RC5 avec la clé *551C2016B00B5F00* puis réparti entre les différentes partitions FAT de la clé USB branchée précédemment. L’algorithme RC5 présent ici semble utiliser 20 rondes. Après une tentative infructueuse d’utiliser les bibliothèques de troubadours cryptopathes, j’ai décidé de riper le code binaire de l’algorithme.

Il suffit alors de prendre chacun des blocs chiffrés dans l’image de la clé et de les rassembler dans un fichier. Ces blocs se trouvent aux offsets :

- 0x240
- 0x40000600
- 0x79B00000

On peut ensuite les extraire simplement avec le script python suivant :

```

1 if __name__ == '__main__':
2
3     data = open("RawDecryptedBuffer.bin", "rb").read()
4
5     offset = 0
6     i = 1
7
8     while( offset < len(data)) :
9         file_size = struct.unpack("<I", data[offset:offset+4])[0]
10
11         rc4_key = data[offset+4:offset+4+0x10]
12         md5_sum = data[offset+0x14:offset+0x14+0x10]
13
14         file_buffer = data[offset+0x24:offset+0x24+file_size]
15
16         offset += file_size + 0x24
17
18         buffer = WikipediaARC4(rc4_key).crypt(file_buffer)
19
20         open("output%d.bin"%i, "wb").write(buffer)
21
22         i = i + 1

```

On trouve finalement une archive chiffrée et un mot de passe qui nous permettent de retrouver la clé de l’épreuve : **0928BDE1E3ED89698632DBFF4A231138**

## **Conclusion**

Ainsi s'achève ma modeste solution. J'ai passé de bons moments sur ce challenge et je remercie les auteurs pour leur superbe travail. Des challenges variés et des trolls nombreux. Je souhaite beaucoup de courage à celles et ceux qui auront à concevoir le challenge l'année prochaine :)