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 :

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

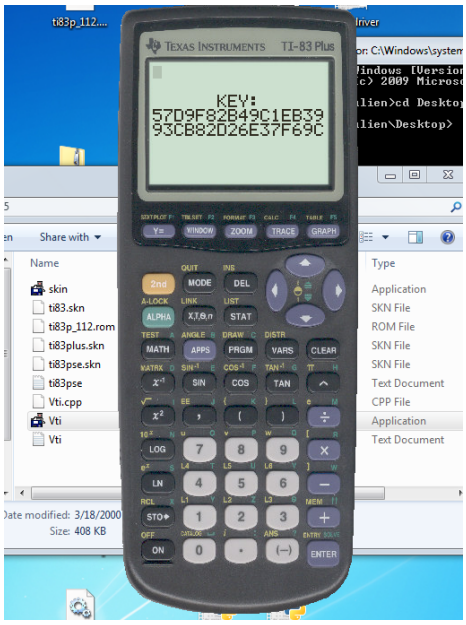
Puis effectue une boucle de calcul avec en paramètre le nombre entré par l'utilisateur. Si le résultat de ce calcul vaut 3298472535 le programme génère la clé du challenge grâce au générateur aléatoire de la calculatrice. L'entrée utilisateur sert alors de graine au générateur aléatoire :

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

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

```
Arr = [0,1996959894,3993919788,2567524794,124634137,1886057615,3915621685,...

user_input = 89594902

state = 0xFFFFFFFF

for i in range(4):

    byte1 = state & 0xFF
    byte2 = (user_input >> (8 * i)) & 0xFF

    byte3 = byte2 ^ byte1

    X = Arr[byte3]

    state = state >> 8

    state = state ^ X

state = state ^ 0xFFFFFFFF

if state == 3298472535:
    print "W00t!"
    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 :

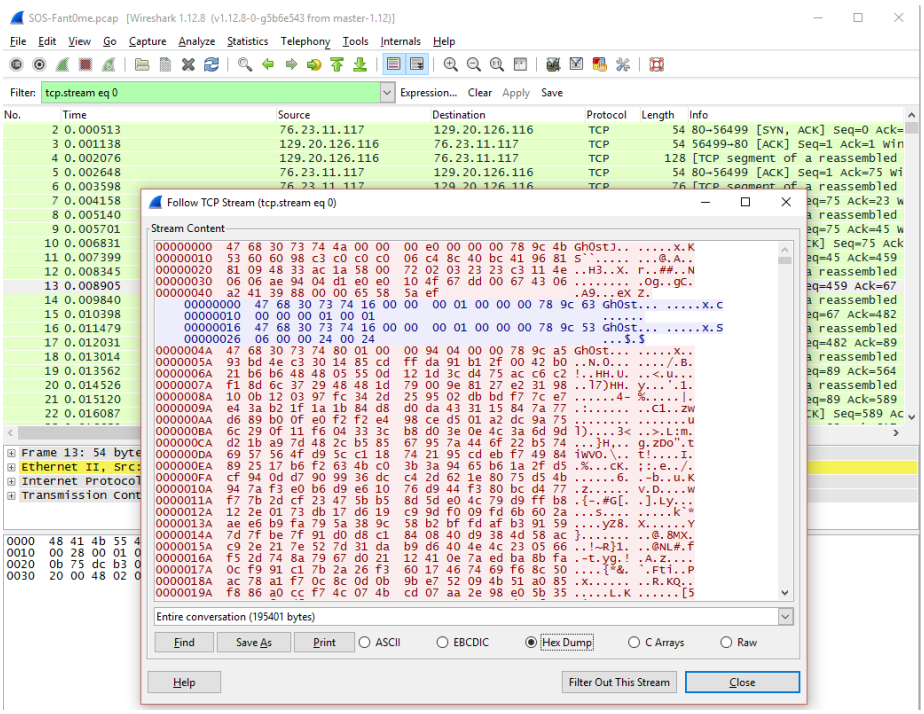


Pour obtenir la clé : **57D9F82B49C1EB3993CB82D26E37F69C**.

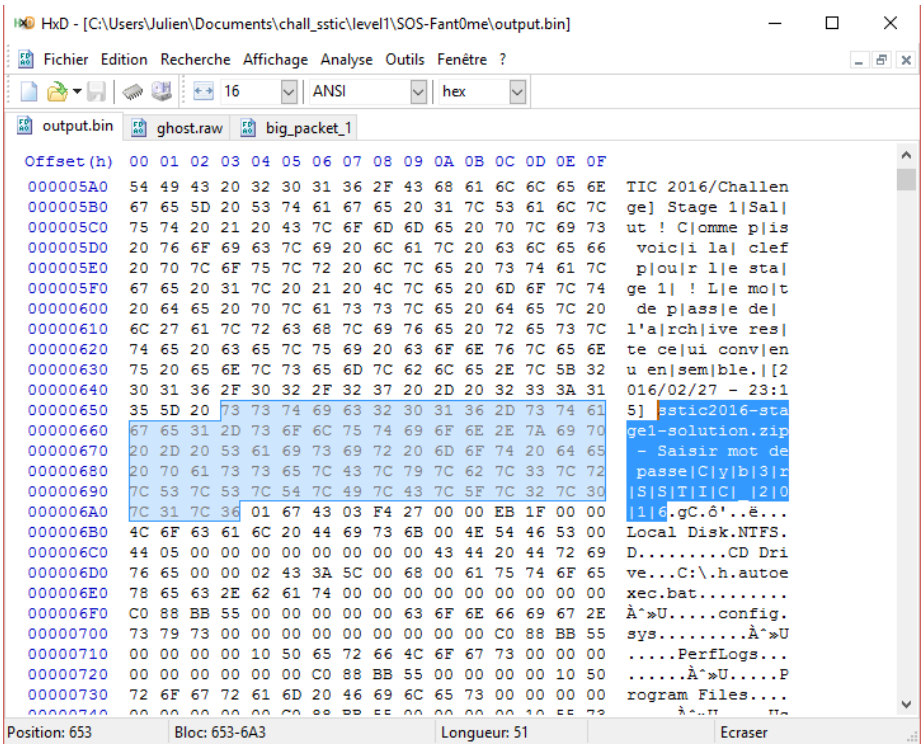
SOS-Fant0me

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

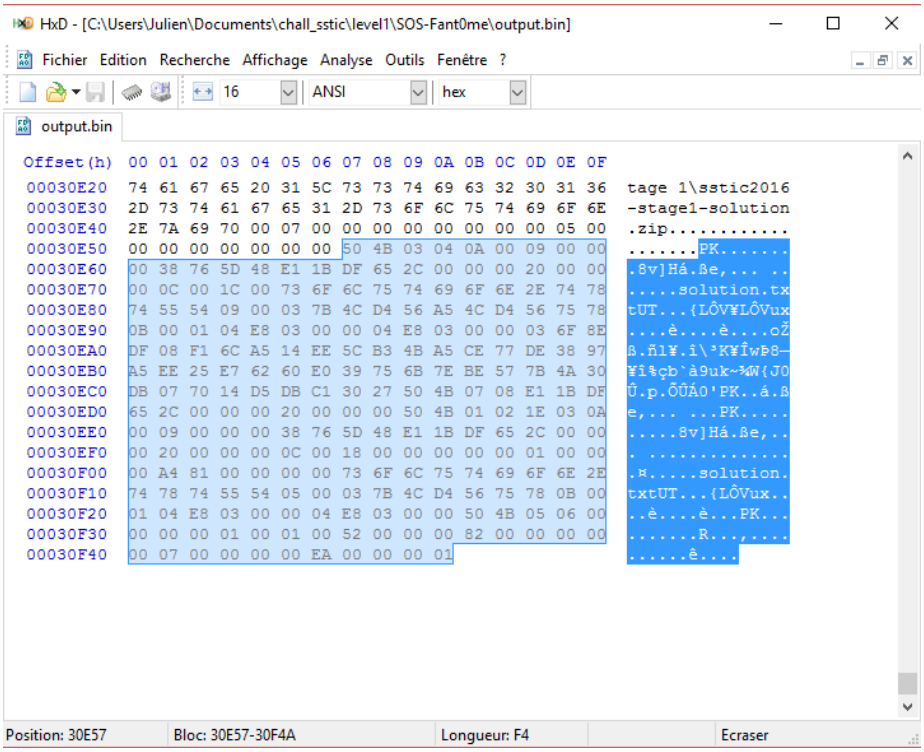
1. http ://www.cert.ssi.gouv.fr/site/CERTA-2014-ACT-002/CERTA-2014-ACT-002.html



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 chaine hexadécimale de 16 octets. Rapidement, un GUID est généré en "xorand" deux buffers en dur dans le binaire :

```
loc_100005CA:                           ; CODE XREF:
                MOVnd       R7, [SP+0x85E0+hc_buf2_1D8_]
                MOVbw       R7, [R7]
                MOVqd       R4, SP+0x85E0+buf1_copy
                MOVdd       R5, [SP+0x85E0+counter_loop_1]
                EXTNDD64    R5, R5
                ADD64       R4, R5
                MOVbw       R4, [R4]
                XOR32       R7, R4
                MOVnd       R4, [SP+0x85E0+hc_buf2_1D8_] ;
                MOVbw       [R4], R7
                MOVdd       R7, [SP+0x85E0+counter_loop_1]
                MOVqw       R7, R7+1
                MOVdd       [SP+0x85E0+counter_loop_1]
                MOVnd       R7, [SP+0x85E0+hc_buf2_1D8_]
                ADD64       R7, R6+1
                MOVnd       [SP+0x85E0+hc_buf2_1D8_], R7
                MOVdd       R7, [SP+0x85E0+counter_loop_1]
                CMPI32wgte  R7, 0x10
                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* :

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

Dont l'interface est la suivante :

```
typedef struct _EFI_DECOMPRESS_PROTOCOL {
    EFI_DECOMPRESS_GET_INFO GetInfo;
    EFI_DECOMPRESS_DECOMPRESS Decompress;
} EFI_DECOMPRESS_PROTOCOL;
```

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

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

```
.data:10001472                  db    0
.data:10001473                  db    0
.data:10001474                  db 0x5C
.data:10001475                  db    0
.data:10001476                  db    0
.data:10001477                  db    0
.data:10001478                  db    0
.data:10001479                  db 0x3C
.data:1000147A                  db 0x4C
.data:1000147B                  db 0x8D
.data:1000147C                  db 0x82
.data:1000147D                  db 0x33
```

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

```
#! /usr/bin/env python2

#~ python implementation of tianocore decompress
#~ see EDK-files BaseUefiDecompressLib.c and PeiLib/Decompress.c
#~ date: 03/2012
#~ author: Jakob Heinemann, jakob@jakobheinemann.de
#~ license: GPL
#~
#~ usage:
#~    tiano = TIANO_DECOMP()
#~    uncompbuf = tiano.Decompress(compbuf)
```

Le buffer décompressé est le suivant :

```
secret data: cb41dcb1d89746705a7fe998f11acce7
```

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

```
loc_1000091A:                           ; CODE XREF: check_key+448
    MOVdd       R7, [SP+0x85E0+counter_loop_1]
    EXTNDD64    R7, R7
    MOVnd       R4, [SP+0x85E0+buf_param]
    ADD64       R4, R7      ; Op1 += Op2
    MOVbw       [SP+0x85E0+p1], [R4]
    MOVdd       [SP+0x85E0+p2], [SP+0x85E0+counter_loop_1]
    CALL32      swap_char
    MOVbd       R4, [SP+0x85E0+or_sum]
    MOVqd       R5, SP+0x85E0+convertedstring
    MOVdd       R1, [SP+0x85E0+counter_loop_1]
    EXTNDD64    R1, R1      ; Sign-extend a dword value
    ADD64       R5, R1      ; Op1 += Op2
    MOVbw       R5, [R5]    ; Move byte
    XOR32       R7, R5      ; Op1 ^= Op2
    OR32        R4, R7      ; Op1 |= Op2
    MOVbd       [SP+0x85E0+or_sum], R4 ; Move byte
    MOVdd       R7, [SP+0x85E0+counter_loop_1]
    MOVqw       R7, R7+1
    MOVdd       [SP+0x85E0+counter_loop_1], R7
    MOVdd       R7, [SP+0x85E0+counter_loop_1]
    CMPI32wgte  R7, 0x10
    JMP8cc      loc_1000091A
```

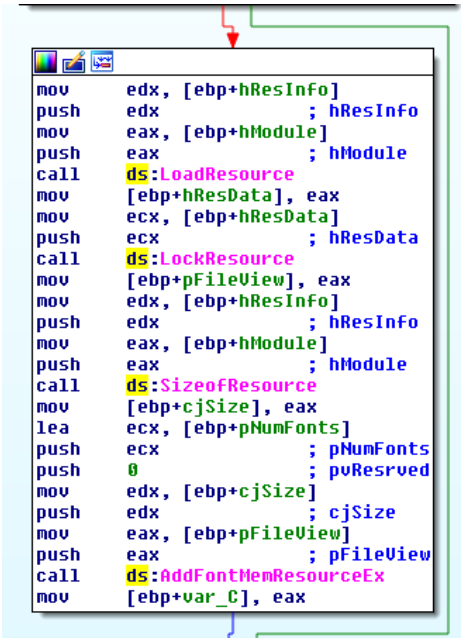
Il suffit d'exécuter le script python suivant :

```python
stored_key = "cb41dcb1d89746705a7fe998f11acce7".decode("hex")

def swap_char(input_char, counter):
    shift = counter % 8
    c = (input_char << shift) | (input_char >> (8 - shift))
    c = (~c) & 0xFF
    return c

s = ""

for i in range(16):
    x = swap_char(ord(stored_key[i]) ,i)
    s += chr(x)

print "key is : %s" % s.encode("hex")
```

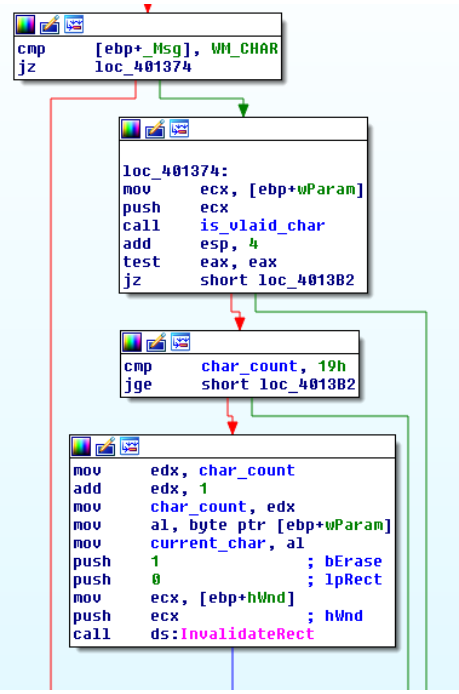
La clé solution est : **347d8c72720d6ec7a501583be0bccc0c**.

loader

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



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 "/=+ ?" :



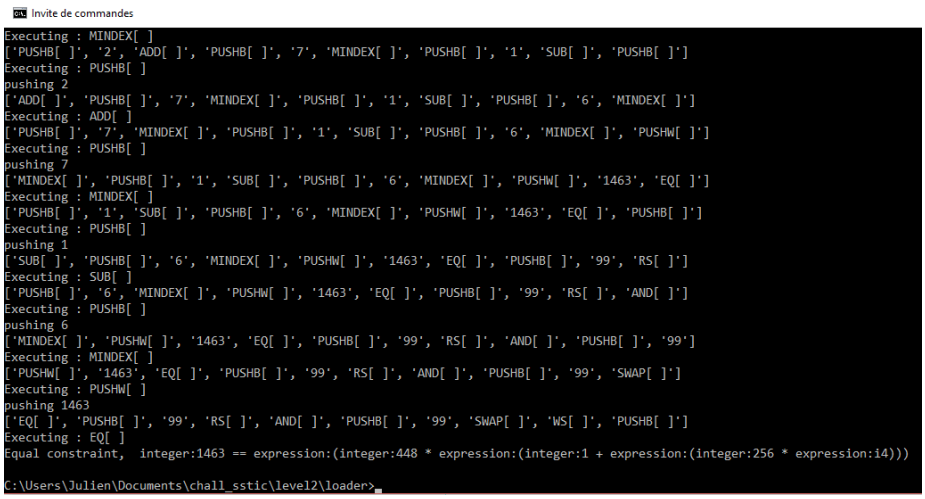
Si le nombre de caractères maximal n'est pas atteint et que le caractère est autorisé il est alors affiché, donc envoyé à la fonte chargée par le noyau.

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 :

```
<TTGlyph name="L" xMin="42" yMin="0" xMax="557" yMax="574">
  <contour>
    <pt x="234" y="488" on="1"/>
    <pt x="234" y="86" on="1"/>
    <pt x="467" y="86" on="1"/>
    <pt x="467" y="277" on="1"/>
    <pt x="557" y="277" on="1"/>
    <pt x="557" y="0" on="1"/>
    <pt x="42" y="0" on="1"/>
    <pt x="42" y="86" on="1"/>
    <pt x="140" y="86" on="1"/>
    <pt x="140" y="488" on="1"/>
    <pt x="42" y="488" on="1"/>
    <pt x="42" y="574" on="1"/>
    <pt x="353" y="574" on="1"/>
    <pt x="353" y="488" on="1"/>
  </contour>
  <instructions><assembly>
      PUSHB[ ]  /* 2 values pushed */
      99 99
      RS[ ] /* ReadStore */
      DUP[ ]  /* DuplicateTopStack */
      PUSHB[ ]  /* 1 value pushed */
      37
      WS[ ] /* WriteStore */
      PUSHB[ ]  /* 1 value pushed */
      1
      ADD[ ]  /* Add */
      WS[ ] /* WriteStore */
    </assembly></instructions>
</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 :



Voici le script qui a généré la sortie précédente :

```python
import sys
import math

CELL_TYPE_INTEGER = 0
CELL_TYPE_EXPRESSION = 1

rs_99 = 1

def F26Dot6(N):

    frac = 0.015625
    big_part = N >> 6

    small_part = 0

    for i in range(6):
        print N & 1
        if (N & 1) == 1:
            small_part += frac
        N = N >> 1
        print "frac", frac
        frac = frac*2

    return big_part + small_part

class Cell():

    def __init__(self, type, content):
        self.type = type
        self.content = content

    def __str__(self):
        if ( self.type == CELL_TYPE_INTEGER):
            return "integer:%d" % self.content
        else:
            return "expression:%s" % self.content

class Stack():

    def __init__(self):
        self.storage = []

    def push(self, element):
        self.storage = [element] + self.storage

    def pop(self):
        element = self.storage[0]
        self.storage = self.storage[1:]
```

```
49
50          return element
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 mindex(i_stream, stack):
135     elt = stack.pop()
136
137     if ( elt.type == CELL_TYPE_EXPRESSION ):
138         print "Cannot mindex 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
```

```
handlers = {}
handlers["PUSHB[ ]"] = push
handlers["PUSHW[ ]"] = push
handlers["MINDEX[ ]"] = mindex
handlers["ROLL[ ]"] = roll
handlers["MUL[ ]"] = mul
handlers["ROLL[ ]"] = roll
handlers["EQ[ ]"] = eq
handlers["RS[ ]"] = rs
handlers["AND[ ]"] = _and
handlers["SWAP[ ]"] = swap
handlers["WS[ ]"] = ws
handlers["ADD[ ]"] = add
handlers["SUB[ ]"] = sub

instructions = []
values = {}
values[0] = 56
values[1] = 5
values[2] = 19
values[3] = 63
values[5] = 26
values[7] = 35
values[8] = 39
values[11] = 27
values[12] = 31
values[13] = 48
values[15] = 2
values[18] = 21
values[19] = 47
values[21] = 22

#Ajout des valeurs sur la stack initiale
#Si la valeur est connue on pousse un nombre, dans le cas contraire une
#expression.
stack = Stack()
for i in range(22):
    if (i in values.keys()):
        stack.push( Cell(CELL_TYPE_INTEGER, values[i]) )
    else:
        stack.push( Cell(CELL_TYPE_EXPRESSION,"i%d" % i) )


instructions.append("ROLL[ ]")
instructions.append("PUSHW[ ]")
instructions.append("256")
instructions.append("MUL[ ]")
instructions.append("PUSHB[ ]")
instructions.append("18")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("256")
instructions.append("MUL[ ]")
instructions.append("PUSHB[ ]")
instructions.append("20")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("19")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHB[ ]")
instructions.append("10")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("6")
instructions.append("ADD[ ]")
instructions.append("PUSHB[ ]")
instructions.append("4")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("6")
instructions.append("ADD[ ]")
instructions.append("PUSHB[ ]")
instructions.append("10")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("320")
instructions.append("MUL[ ]")
instructions.append("PUSHB[ ]")
instructions.append("11")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("384")
instructions.append("MUL[ ]")
instructions.append("PUSHW[ ]")
instructions.append("186")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
```

```
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHB[ ]")
instructions.append("18")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("63")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("ROLL[ ]")
instructions.append("PUSHW[ ]")
instructions.append("320")
instructions.append("MUL[ ]")
instructions.append("PUSHB[ ]")
instructions.append("14")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("39")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHB[ ]")
instructions.append("7")
instructions.append("SUB[ ]")
instructions.append("PUSHB[ ]")
instructions.append("14")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("35")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHB[ ]")
instructions.append("13")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("2")
instructions.append("ADD[ ]")
instructions.append("PUSHB[ ]")
instructions.append("6")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("6")
instructions.append("ADD[ ]")
instructions.append("PUSHB[ ]")
instructions.append("16")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("256")
instructions.append("MUL[ ]")
instructions.append("PUSHB[ ]")
instructions.append("17")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("3")
instructions.append("SUB[ ]")
instructions.append("PUSHB[ ]")
instructions.append("8")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("1")
instructions.append("ADD[ ]")
instructions.append("PUSHB[ ]")
instructions.append("11")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("2")
instructions.append("ADD[ ]")
instructions.append("PUSHB[ ]")
instructions.append("10")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("7")
instructions.append("SUB[ ]")
instructions.append("SWAP[ ]")
instructions.append("PUSHB[ ]")
instructions.append("4")
instructions.append("ADD[ ]")
instructions.append("PUSHB[ ]")
```

```
instructions.append("8")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("263")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHB[ ]")
instructions.append("16")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("26")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHB[ ]")
instructions.append("7")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("7")
instructions.append("ADD[ ]")
instructions.append("PUSHB[ ]")
instructions.append("9")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("28")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHB[ ]")
instructions.append("7")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("192")
instructions.append("MUL[ ]")
instructions.append("PUSHB[ ]")
instructions.append("8")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("2")
instructions.append("SUB[ ]")
instructions.append("PUSHB[ ]")
instructions.append("12")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("6")
instructions.append("SUB[ ]")
instructions.append("PUSHB[ ]")
instructions.append("10")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("21")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHB[ ]")
instructions.append("13")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("5")
instructions.append("ADD[ ]")
instructions.append("PUSHB[ ]")
instructions.append("9")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("53")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
```

```
instructions.append("SWAP[ ]")
instructions.append("PUSHW[ ]")
instructions.append("21")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHB[ ]")
instructions.append("7")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("448")
instructions.append("MUL[ ]")
instructions.append("PUSHB[ ]")
instructions.append("11")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("1")
instructions.append("SUB[ ]")
instructions.append("PUSHB[ ]")
instructions.append("4")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("320")
instructions.append("MUL[ ]")
instructions.append("PUSHB[ ]")
instructions.append("6")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("64")
instructions.append("MUL[ ]")
instructions.append("PUSHB[ ]")
instructions.append("6")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("582")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHB[ ]")
instructions.append("8")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("20")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHB[ ]")
instructions.append("9")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("2")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHW[ ]")
instructions.append("320")
instructions.append("MUL[ ]")
instructions.append("PUSHB[ ]")
instructions.append("6")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("2")
instructions.append("ADD[ ]")
instructions.append("PUSHB[ ]")
instructions.append("7")
instructions.append("MINDEX[ ]")
instructions.append("PUSHB[ ]")
instructions.append("1")
instructions.append("SUB[ ]")
instructions.append("PUSHB[ ]")
instructions.append("6")
instructions.append("MINDEX[ ]")
instructions.append("PUSHW[ ]")
instructions.append("1463")
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")
```

```
instructions.append("MUL[ ]")
instructions.append("SWAP[ ]")
instructions.append("PUSHB[ ]")
instructions.append("3")
instructions.append("ADD[ ]")
instructions.append("SWAP[ ]")
instructions.append("PUSHW[ ]")
instructions.append("3")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHW[ ]")
instructions.append("256")
instructions.append("MUL[ ]")
instructions.append("PUSHB[ ]")
instructions.append("2")
instructions.append("SUB[ ]")
instructions.append("PUSHW[ ]")
instructions.append("320")
instructions.append("MUL[ ]")
instructions.append("PUSHW[ ]")
instructions.append("1970")
instructions.append("EQ[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("AND[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("SWAP[ ]")
instructions.append("WS[ ]")
instructions.append("PUSHB[ ]")
instructions.append("99")
instructions.append("RS[ ]")
instructions.append("IF[ ]")
instructions.append("PUSHB[ ] /* 5 values pushed */")
instructions.append("0 1 2 3 4")
instructions.append("PUSHB[ ] /* 5 values pushed */")
instructions.append("5 6 7 8 9")
instructions.append("ELSE[ ]  /* Else */")
instructions.append("PUSHB[ ] /* 5 values pushed */")
instructions.append("60 61 62 63 64")
instructions.append("PUSHB[ ] /* 5 values pushed */")
instructions.append("65 66 67 68 69")
instructions.append("EIF[ ] /* EndIf */")
instructions.append("SVTCA[0] /* SetFPVectorToAxis */")
instructions.append("PUSHB[ ]")
instructions.append("0")
instructions.append("SCFS[ ]  /* SetCoordFromStackFP */")
instructions.append("PUSHB[ ]")
instructions.append("0")
instructions.append("SCFS[ ]  /* SetCoordFromStackFP */")
instructions.append("PUSHB[ ]")
instructions.append("0")
instructions.append("SCFS[ ]  /* SetCoordFromStackFP */")
instructions.append("PUSHB[ ]")
instructions.append("0")
instructions.append("SCFS[ ]  /* SetCoordFromStackFP */")
instructions.append("PUSHB[ ]")
instructions.append("0")
instructions.append("SCFS[ ]  /* SetCoordFromStackFP */")
instructions.append("PUSHB[ ]")
instructions.append("0")
instructions.append("SCFS[ ]  /* SetCoordFromStackFP */")
instructions.append("PUSHB[ ]")
instructions.append("0")
instructions.append("SCFS[ ]  /* SetCoordFromStackFP */")
instructions.append("PUSHB[ ]")
instructions.append("0")
instructions.append("SCFS[ ]  /* SetCoordFromStackFP */")
instructions.append("PUSHB[ ]")
instructions.append("0")
instructions.append("SCFS[ ]  /* SetCoordFromStackFP */")
instructions.append("PUSHB[ ]")
instructions.append("0")
instructions.append("SCFS[ ]  /* SetCoordFromStackFP */")

execute(handlers, instructions, stack)
```

On obtient finalement la clé encodée en base64 : "4ft/0AdJNjHBFW5ch+vVGw". Un fois le bourrage ajouté et la chaine décodée on obtient la clé : **e1fb7fd007493631c1156e5c87ebd51b**.

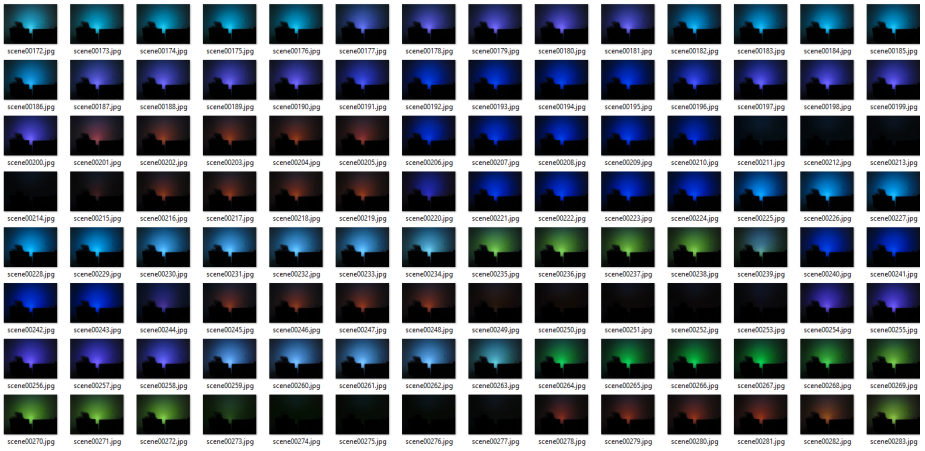
On peut enfin passer au niveau 3 et rencontrer les gens qui ont popularisé le hacking en France (ou pas).

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 chaine 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.

```python
import zlib
import struct

index_to_color = {}
index_to_color[0] = "rouge"
index_to_color[1] = "vert"
index_to_color[2] = "jaune"
index_to_color[3] = "bleu"
index_to_color[4] = "violet"
index_to_color[5] = "cyan"
index_to_color[6] = "blanc"
index_to_color[7] = "noir"

color_to_index = {}
color_to_index["rouge"] = 0
color_to_index["vert"] = 1
color_to_index["jaune"] = 2
color_to_index["bleu"] = 3
color_to_index["violet"] = 4
color_to_index["cyan"] = 5
color_to_index["blanc"] = 6
color_to_index["noir"] = 7

#Chaine hardcodee du binaire
xor_string =    "2830
      a43f6d280423362a32dcad0ba04be8201f64840af4c4c78a8dc0a2c44019a143823814fd6c90e07e2a40dfd3f23e".decode(
      "hex")

def generate_byte_string(video_colors):

    len_array = len(video_colors)

    if (len_array % 3) != 0:
```

```
        print "da fuck ?"
        return None

    out_buf = ""

    previous_color = 6
    i = 0
    j = 0

    while(i < len_array):

        #On prend un triplet de couleurs pour decode un byte
        triplet = video_colors[:3]
        video_colors = video_colors[3:]

        parts = []

        for color in triplet:

            encoded_data = (color_to_index[color] - previous_color) % 8
            previous_color = color_to_index[color]

            x = (encoded_data - 1 ) % 7

            parts.append(x)

        parts.reverse()

        #Conversion en base 6
        byte = 0
        for part in parts:
            byte += byte*6 + part

        #Finalement on xor avec un octet de la chaine hardcodee
        byte = byte ^ ord( xor_string[j] )

        i += 3
        j += 1

        out_buf += chr(byte)

    return out_buf

#Video notees patiemment sans perdre (top) de dixiemes
video_colors = ["cyan", "violet", "cyan"]
video_colors += ["violet","bleu","violet"]
video_colors += ["rouge","bleu","noir"]
video_colors += ["rouge","bleu","cyan"]
video_colors += ["blanc","jaune","bleu"]
video_colors += ["rouge","noir","violet"]
video_colors += ["blanc","vert","jaune"]
video_colors += ["noir","rouge","jaune"]
video_colors += ["cyan","blanc","noir"]
video_colors += ["cyan","noir","vert"]
video_colors += ["blanc","jaune","bleu"]
video_colors += ["jaune","bleu","violet"]
video_colors += ["bleu","violet","noir"]
video_colors += ["blanc","bleu","cyan"]
video_colors += ["bleu","vert","bleu"]
video_colors += ["blanc","bleu","blanc"]
video_colors += ["rouge","blanc","vert"]
video_colors += ["cyan","rouge","jaune"]
video_colors += ["cyan","vert","bleu"]
video_colors += ["rouge","vert","cyan"]
video_colors += ["rouge","jaune","violet"]
video_colors += ["blanc","vert","violet"]
video_colors += ["cyan","jaune","blanc"]
video_colors += ["noir","rouge","cyan"]
video_colors += ["blanc","noir","cyan"]
video_colors += ["noir","bleu","rouge"]
video_colors += ["blanc","violet","noir"]
video_colors += ["vert","rouge","violet"]


out_buf = generate_byte_string(video_colors)
#On ignore les 4 octets de taille
out_buf = out_buf[4:]

s = zlib.decompress(out_buf)
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 :

```
typedef struct FileStruct {
    Event ReadyToHashFile
    Event FileHashed
    Event Event3
    Event Event4
    DWORD FileSize
    char Md5Sum[16]
    PVOID FileBuffer
} FileStruct;
```

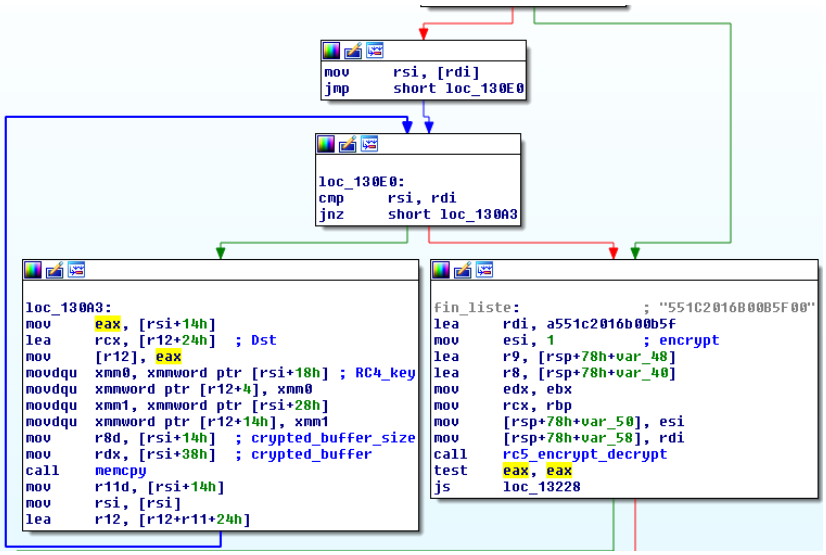
Le programme *userland* est chargé de détecter le branchement d'un clé USB. Lorsque celle-cie est branché, un message de type *DBT_DEVICEARRIVAL* est envoyé à la fonction de gestion de la fenêtre du programme qui transmet alors le nom du volume monté au noyau grâce à la fonction *ZwSetInformationProcess*.

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.

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

```
typedef struct CryptedFileHeader {
    DWORD FileSize
    char Rc4Key[16]
    char Md5Sum[16]
} 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 librairies 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 l'a 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 :

```
if __name__ == '__main__':

    data = open("RawDecryptedBuffer.bin","rb").read()

    offset = 0
    i = 1

    while( offset < len(data)) :
        file_size = struct.unpack("<I", data[offset:offset+4])[0]

        rc4_key = data[offset+4:offset+4+0x10]
        md5_sum = data[offset+0x14:offset+0x14+0x10]

        file_buffer = data[offset+0x24:offset+0x24+file_size]

        offset += file_size + 0x24

        buffer = WikipediaARC4(rc4_key).crypt(file_buffer)

        open("output%d.bin"%i,"wb").write(buffer)

        i = i + 1
```

On trouve finalement une archive chiffrée et une 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 :)