Solution Challenge SSTIC 2016

Jean Bernard Beuque
13 Avril 2016

Sommaire

1. Préambule

L'objectif du challenge est de trouver une adresse email @sstic.org.

Le challenge est constitué du fichier *challenge.pcap*.

Avec wireshark, on constate que ce fichier contient le transfert http du fichier challenge.zip

GET /challenge2016/challenge.zip HTTP/1.0

Host: static.sstic.org

User-Agent: Mozilla/5.0 (Wayland; BTTF; Linux x86_128; rv:142.0) Gecko/20100101 Firefox/142.0

*Accept: */**

HTTP/1.0 200 OK

Server: CERN/3.0A

Content-type: application/zip

Content-Length: 52331069

Avec Wireshark, on sauve la conversation http. Après avoir supprimé l'entête http, on obtient le fichier challenge.zip.

Ce fichier contient un jeu écrit en HTML/JS.

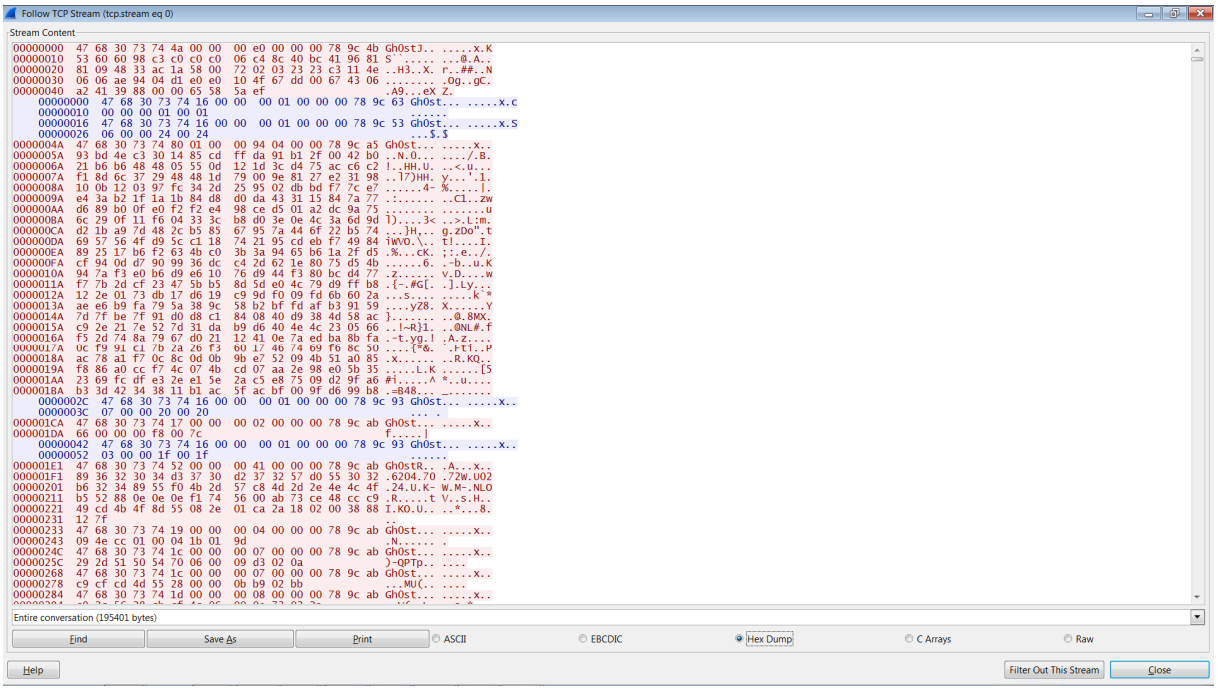
2. Level 1

1. SOS Fantome

Le challenge est constitué d'un fichier pcap : SOS-Fant0me.pcap

Outil : Wireshark

Le fichier pcap contient la communication entre un client et un serveur via le protocole Gh0st.



Tous les paquets du protocole applicatif commencent par le préfixe : « Gh0st ».

Une recherche du mot «« Gh0st » sur internet nous permet de trouver le document *wp-know-your-digital-enemy.pdf*

Le protocole Gh0st est le protocole de Command & Control utilisé par le malware Ghost RAT.

Un paquet du protocole Ghost comporte:

1/ Un prefixe de 5 octets : les caractères : Gh0st

2/ Un entier de 4 octets (msb first) : la longueur en octet du paquet

3/ Un entier de 4 octets (msb first) : La longueur du paquet décompressé

4/ payload du paquet compressé en zlib

- Après decompression zlib, la payload contient :
  - Un code d'opération sur 1 octet.

Packet code	

3	Commande de téléchargement de fichier
2	Directory
105	File size
5	File Data
124	TOKEN_KEYBOARD_DATA

- A partir de ces informations, on développe un programme pour récupérer les fichiers téléchargés via le protocole Gh0st GhostAnalyzer (voir annexe).

- Avec wireshark, on extrait le TCP stream entre le client et le serveur.
- Et avec le programme GhostAnalyzer, on obtient les fichiers : *how_to_rule_the_world.txt, visio_stage2.mp4 et sstic2016-stage1-solution.zip*.

Le fichier *sstic2016-stage1-solution.zip* est un zip crypté qui contient le fichier *solution.txt*.

Le mot de passe de l'archive zip se trouve dans les paquets avec le code TOKEN_KEYBOARD_DATA.

On trouve le mot de passe est : Cyb3rSSTIC_2016.

Une fois décrypté le fichier *solution.txt* contient la clef de validation du challenge : 368BE8C1CC7CC70C2245030934301C15.

2. Calc

Le challenge est constitué de l'archive calc.zip qui contient un fichier SSTIC16.8xp

Les fichiers .8xp contiennent des programmes basic pour les calculatrices Texas Instruments TI83+.

Via le site https://www.cemetech.net/sc/ on obtient un listing du programme basic SSTIC16.

```
: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
:If S=13:Goto 13
:If S=14:Goto 14
:If S=16:Goto 16
:If S=17:Goto 17
:If S=18:Goto 18
:If S=19:Goto 19
:If S=21:Goto 21
:If S=23:Goto 23
:Disp "DISPATCH ERROR"
:Stop
:
:Lbl 3
:sub(Str1,length(Str1)-((A+1)*8)+1,8)->Str1
:Goto 0
:
:Lbl 2
:" "->Str1
:Repeat not(A
: A/2->A
: sub("01",1+not(not(fPart(Ans))),1)+Str1->Str1
: iPart(A->A
:End
:sub(Str1,1,length(Str1)-1)->Str1
:While length(Str1)<32
: "0"+Str1->Str1
:End
:Goto 0
:
:
:Lbl 4
:" "->Str3
:For(I,1,length(Str1)
: If "1"=sub(Str1,length(Str1)-I+1,1) xor "1"=sub(Str2,length(Str1)-I+1,1):Then
: "1"+Str3->Str3
: Else
: "0"+Str3->Str3
: End
:End
:sub(Str3,1,length(Str3)-1)->Str3
:Goto 0
:
:
:Lbl 7
:0->A
:For(I,1,length(Str1)
: If "1"=sub(Str1,length(Str1)-I+1,1):Then
: A+2^(I-1->A
: End
:End
:Goto 0
:
:
:Lbl 15
:"00000000"+sub(Str1,1,24)->Str1
:Goto 0
:
:Lbl 22
:" "->Str1
:0->C
:While A
: iPart(A/16)->C
```

```
: fPart(A/16)*16->B
: sub("0123456789ABCDEF",B+1,1)+Str1->Str1
: C->A
:End
:sub(Str1,1,length(Str1)-1)->Str1
:If length(Str1)<2
:"0"+Str1->Str1
:Goto 0
:
:
:Lbl 1
:Input "Entrez le code : ",Z
:4294967295->C
:0->N
:{0,1996959894,3993919788,2567524794,124634137,1886057615,3915621685,2657392035,249268274,2044508324,377211
5230,2547177864,162941995,2125561021,3887607047,2428444049,498536548,1789927666,4089016648,2227061214,45054
8861,1843258603,4107580753,2211677639,325883990,1684777152,4251122042,2321926636,335633487,1661365465,41953
02755,2366115317,997073096,1281953886,3579855332,2724688242,1006888145,1258607687,3524101629,2768942443,901
097722,1119000684,3686517206,2898065728,853044451,1172266101,3705015759,2882616665,651767980,1373503546,336
9554304,3218104598,565507253,1454621731,3485111705,3099436303,671266974,1594198024,3322730930,2970347812,79
5835527,1483230225,3244367275,3060149565,1994146192,31158534,2563907772,4023717930,1907459465,112637215,268
0153253,3904427059,2013776290,251722036,2517215374,3775830040,2137656763,141376813,2439277719,3865271297,18
02195444,476864866,2238001368,4066508878,1812370925,453092731,2181625025,4111451223,1706088902,314042704,23
44532202,4240017532,1658658271,366619977,2362670323,4224994405,1303535960,984961486,2747007092,3569037538,1
256170817,1037604311,2765210733,3554079995,1131014506,879679996,2909243462,3663771856,1141124467,855842277,
2852801631,3708648649,1342533948,654459306,3188396048,3373015174,1466479909,544179635,3110523913,3462522015
,1591671054,702138776,2966460450,3352799412,1504918807,783551873,3082640443,3233442989,3988292384,259625464
6,62317068,1957810842,3939845945,2647816111,81470997,1943803523,3814918930,2489596804,225274430,2053790376,
3826175755,2466906013,167816743,2097651377,4027552580,2265490386,503444072,1762050814,4150417245,2154129355
,426522225,1852507879,4275313526,2312317920,282753626,1742555852,4189708143,2394877945,397917763,1622183637
,3604390888,2714866558,953729732,1340076626,3518719985,2797360999,1068828381,1219638859,3624741850,29366751
48,906185462,1090812512,3747672003,2825379669,829329135,1181335161,3412177804,3160834842,628085408,13826053
66,3423369109,3138078467,570562233,1426400815,3317316542,2998733608,733239954,1555261956,3268935591,3050360
625,752459403,1541320221,2607071920,3965973030,1969922972,40735498,2617837225,3943577151,1913087877,8390837
1,2512341634,3803740692,2075208622,213261112,2463272603,3855990285,2094854071,198958881,2262029012,40572606
10,1759359992,534414190,2176718541,4139329115,1873836001,414664567,2282248934,4279200368,1711684554,2852811
16,2405801727,4167216745,1634467795,376229701,2685067896,3608007406,1308918612,956543938,2808555105,3495958
263,1231636301,1047427035,2932959818,3654703836,1088359270,936918000,2847714899,3736837829,1202900863,81723
3897,3183342108,3401237130,1404277552,615818150,3134207493,3453421203,1423857449,601450431,3009837614,32947
10456,1567103746,711928724,3020668471,3272380065,1510334235,755167117}->L1
:Z->A
:5->S
:Goto 2
:
:
:Lbl 5
:Str1->Str5
:C->A
:11->S
:Goto 2
:
:
:Lbl 11
:Str1->Str6
:Goto 12
:
:
:Lbl 12
:Str6->Str1
:0->A
:13->S
:Goto 3
:
:
:Lbl 13
:Str1->Str7
:Str5->Str1
:N->A
:14->S
:Goto 3
:
:
:Lbl 14
:Str1->Str1
:Str7->Str2
:6->S
:Goto 4
:
:
:Lbl 6
:Str3->Str1
:8->S
:Goto 7
:
:
:Lbl 8
:L1(A+1)->A
:9->S
```

```
:Goto 2
:
:
:Lbl 9
:Str1->Str8
:Str6->Str1
:10->S
:Goto 15
:
:
:Lbl 10
:Str8->Str2
:16->S
:Goto 4
:
:
:Lbl 16
:Str3->Str1
:11->S
:N+1->N
:If N=4
:Goto 17
:Goto 0
:
:
:Lbl 17
:"11111111111111111111111111111111"->Str2
:18->S
:Goto 4
:
:
:Lbl 18
:Str3->Str1
:19->S
:Goto 7
:
:
: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
:
:
:Lbl 21
:iPart(rand*256)->A
:23->S
:Goto 22
:
:
:Lbl 23
:If X=17:Then
: If Y=5:Then
: Goto X
: End
: 1->X
: 5->Y
:End
:Output(Y,X,Str1)
:X+2->X
:Goto 21
:
:Lbl X
:Stop
```

On remarque le tableau de 256 constantes 0,1996959894 …

Ces constantes sont utilisées pour le calcul d'un CRC32 avec le polynôme 0xEDB88320.

Le programme calcul le CRC32 de la valeur entrée par l'utilisateur. Au Lbl19, il vérifie que le CRC est égal à 3298472535. Si le CRC est correct, le programme affiche la clef de validation du challenge, sinon il affiche le message « PERDU ».

Il faut donc déterminer la valeur dont le CRC32 vaut 3298472535.

Le calcul du CRC est un calcul du reste de la division d'un polynôme par le polynôme générateur du CRC.

Soit g(x) le polynôme générateur du CRC32. (Polynôme de degré 32 à coefficients binaires).

r(x) la valeur du reste de la division (polynôme de degré 31). C'est la valeur du « crc » attendu. Soit 3298472535 ^ 0xFFFFFFFF.

e(x) le polynôme correspondant à la valeur entrée par l'utilisateur (polynôme de degré 31).

i(x) un polynôme de degré 31 dont tous les coefficients sont à 1.

k(x) le polynôme quotient de la division par g(x) (polynôme de degré 31)

L'équation à résoudre est la suivante :

$x^{32} * (e(x) + i(x)) = k(x) * g(x) + r(x)$

g(x) et r(x) sont connus. On trouve les coefficients de k(x) en développant le produit k(x) * g(x) + r(x) dont les coefficients de degré 0 à 31 sont nuls.

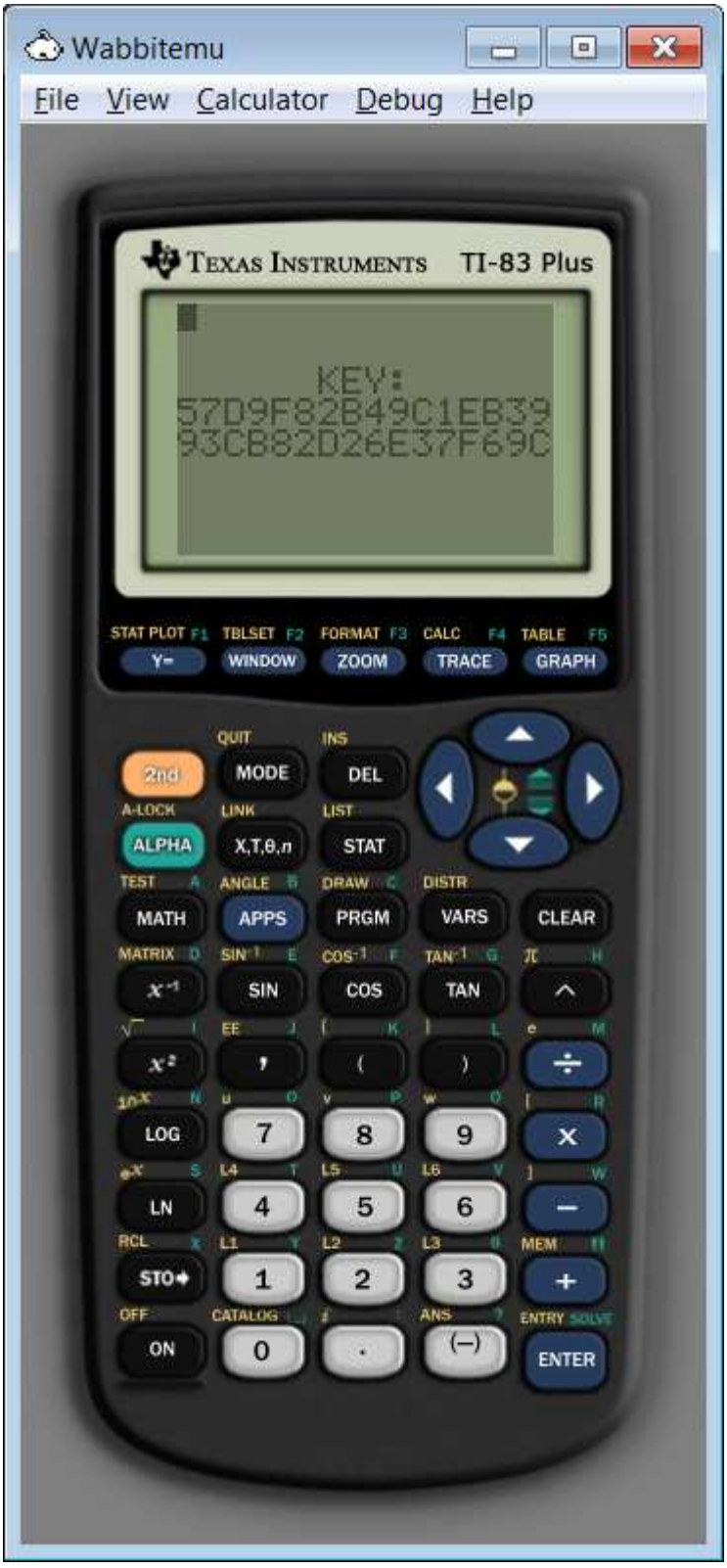
Le programme suivant nous donne la valeur de e(x)

```
#include <stdio.h>

#define gbit64(v,nb)  (((nb)>63)?0:(((v)>>(63-(nb)))&1))
#define sbit64(v,nb,b) (v=(b==1? ((v) | (1Lu<<(63-(nb))) ):((v) & ~(1Lu<<(63-(nb))) )))

/****************************************/
int main(int argc, char *argv[])
{
	unsigned long long g = 0xEDB88320;
	 unsigned long long r = 3298472535  ^ 0xFFFFFFFF;
	unsigned long long prod;
	int i,j;
	unsigned long long g64, k64, r64;
	unsigned int bt;


	// Calcul des coefficients de k(x)
	 k64=0;
	 g64 = (g<<32) | 0x80000000;
	r64 = (r<<32) ;

	for (i=0; i<32; i++)
	{
		bt=0;
		for (j=0; j<=i; j++)
		{
			bt ^= gbit64(g64,j) * gbit64(k64,i-j);
		}
		bt ^= gbit64(r64,i);
```

```
                        sbit64(k64,i,bt);

                }


 // Calcul des coefficients du produit k(x).g(x) +r(x)
                prod=0;
                for (i=0; i<64; i++)
                {
                        bt=0;
                        for (j=0; j<=i; j++)
                        {
                                bt ^= gbit64(g64,j) * gbit64(k64,i-j);
                        }
                        sbit64(prod,i,bt);
                }

                prod ^= (r64);


                prod ^= 0xFFFFFFFF;
                printf("prod=0x%0lX\n",prod);
                printf("prod=%0lu\n",prod);
}
```

```
$ ./icrc
prod=0x5571C16
prod=89594902
```

Après avoir entrée la valeur 89594902, le programme nous donne la clef du challenge : 57D9F82B49C1EB39 93CB82D26E37F69C


Wabbitemu
File View Calculator Debug Help
TEXAS INSTRUMENTS TI-83 Plus
KEY:
57D9F82B49C1EB39
93CB82D26E37F69C
STAT PLOT F1 TBLSET F2 FORMAT F3 CALC F4 TABLE F5
Y= WINDOW ZOOM TRACE GRAPH
2nd MODE DEL
ALPHA X,T,θ,n STAT
MATH APPS PRGM VARS CLEAR
SIN COS TAN
LOG LN STO→ ON
7 8 9 4 5 6 1 2 3 0 .
ENTER

3. Level 2

1. Loader

1. Programme loader.exe

Outil : IDA, ResourcesExtract

En observant les strings dans le fichier loader.exe, on trouve la chaine « BizzaroSSTIC » !

En mettant un breakpoint sur cette chaine, on constate qu'elle est utilisée comme nom de fonte de caractère dans un appel à *CreateFontA*. Une fonte qui s'appelle « BizzaroSSTIC » !

Cette fonte est présente comme ressource binaire dans le programme loader.exe. Elle est installée par un appel à *AddFontMemResourceEx*.

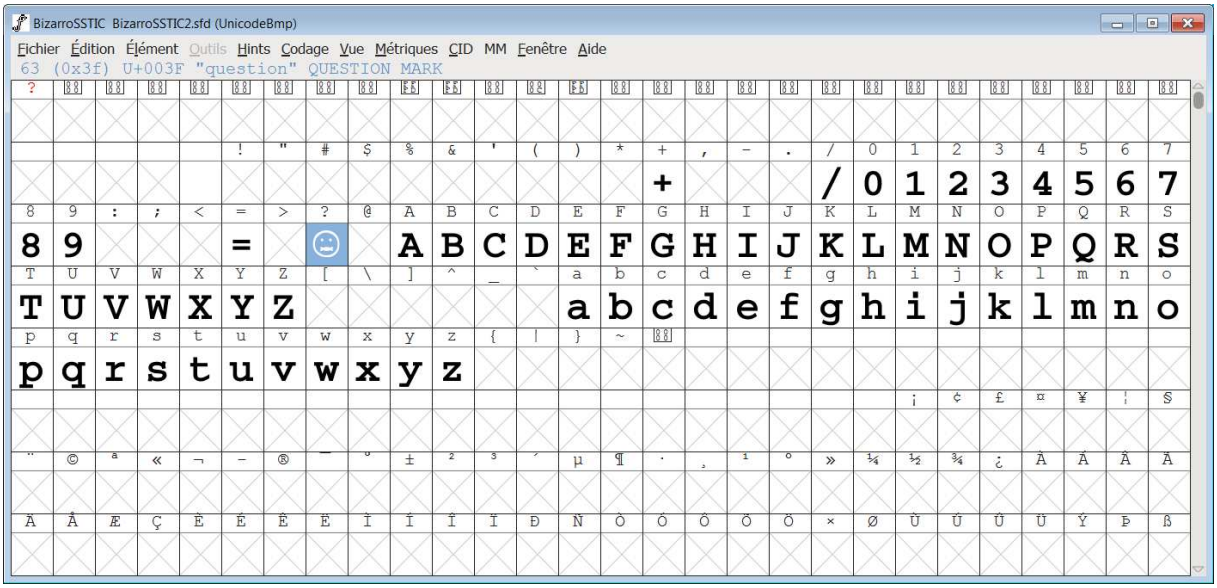
On extrait les ressources du programme loader.exe à l'aide de l'outil *ResourcesExtract.*

On obtient un fichier *loader_1337_8.bin* qui contient la fonte BizzaroSSTIC.

2. La fonte bizzaroSSTIC

Outil : FontForge.

On analyse la fonte à l'aide de FontForge.



Il s'agit d'une fonte TrueType. Un byte code de la VM True Type est associé à chaque caractère.

(NB : La documentation du jeu d'instructions de la VM TrueType est disponible ici :

https://developer.apple.com/fonts/TrueType-Reference-Manual/RM05/Chap5.html)

Le byte code du caractère ‘A’ est le suivant :

```
PUSHB_2
 99
 99
RS
DUP
PUSHB_1
 26                    // Caractère ‘A’
WS
PUSHB_1
 1
ADD
WS
```

Ce code effectue les opérations suivantes :

-Ecriture de la valeur 26 (code du caractère) dans le registre dont le numéro est dans le registre R99.

- Incrémentation du registre R99.

Chaque caractère est associé à un byte code similaire (à l’exception du caractère ‘?’). Seul le code du caractère change.

Le caractère ‘?’ est associé à un byte code différent (Voir annexe). Il effectue les opérations suivantes :

- Chargement sur la pile des valeurs des registres R0 à R21.
- Vérifie la validité d’équations sur les valeurs chargée sur la pile : 22 équations.
- Le registre R99 contient un booléen qui vaut true si les 22 équations sont vérifiées.
- Si le registre R99 vaut true, le caractère est modifié pour obtenir un smiley souriant.
- Si le registre R99 vaut false, le caractère est modifié pour obtenir un smiley triste.

On écrit le programme TrueType ByteCode analyseur (Voir en Annexe) pour obtenir les 22 équations à vérifier.

```
R2 == 19
(R12) * (384) == 186
R3 == 63
R8 == 39
R7 == 35
(((R13) + (6)) * (320)) - (7) == 263
R5 == 26
(R21) + (6) == 28
R18 == 21
(R0) - (3) == 53
(R11) - (6) == 21
(((R19) * (256)) + (6)) * (192) == 582
(R1) * (256) == 20
 R15 == 2
(((R4) * (256)) + (1)) * (448) == 1463
(((R14) * (320)) - (2)) * (320) == 1415
(((R17) + (2)) + (4)) + (2) == 70
(R10) - (1) == 32
(R16) - (2) == 5
((((R9) + (2)) + (7)) * (64)) * (320) == 90
((((((R6) + (5)) - (1)) - (3)) + (5)) - (6)) * (64) == 3
```

```
(((((((R20) - (7)) - (1)) * (256)) + (3)) * (256)) - (2)) * (320) == 1970
```

On résout les équations pour trouver les valeurs suivantes pour les registres R0 à R21. (On a 22 équations et 22 inconnues).

NB : Les valeurs des registres sont des nombres flottant de précision fixe : F26.6. 26 bits pour la partie entière et 6 bits pour la partie fractionnaire (i.e. en 1/64^{ième}).

```
R0 = 56  '4'
R1 = 5   'f'
R2 == 19 't'
R3 == 63 '/'
R4 = 52  '0'
R5 = 26  'A'
R6 = 3   'd'
R7 == 35 'J'
R8 == 39  'N'
R9 = 9   'j'
R10 = 33 'H'
R11 = 27 'B'
R12 = 31 'F'
R13 = 48 'W'
R14 = 57  '5'
R15 == 2 'c'
R16 = 7  'h'
R17 = 62 '+'
R18 == 21 'v'
R19 = 47 'V'
R20 = 32   'G'
R21 = 22 'w'
```

Les valeurs des registres nous donnent la séquence de caractères 4ft/0AdJNjHBFW5ch+vVGw.

On vérifie avec le programme loader que la séquence est correcte : le smiley sourit.

Après décodage base64 de la string « 4ft/0AdJNjHBFW5ch+vVGw== », on trouve la clef de validation du challenge :

```
0xe1 0xfb 0x7f 0xd0 0x07 0x49 0x36 0x31 0xc1 0x15 0x6e 0x5c 0x87 0xeb 0xd5 0x1b
```

2. Huge

3. Le fichier huge.tar

Le challenge est constitué d'un fichier huge.zip d'une taille de 1666 octets.

Après avoir dézippé le fichier, on trouve un fichier huge.tar d'une taille de 107 Ko.

Quand on essaye de « détarrer » l'archive, on obtient un message d'erreur

```
$ tar xvf huge.tar
Huge
tar: Huge: Cannot seek to 25297854992384: Invalid argument
tar: Exiting with failure status due to previous errors
```

En regardant l'entête du fichier tar, on constate qu'il s'agit d'un « SparseFile », un fichier à trou.

Et sa taille est de 128.574.140.715.008 octets, soit 128 Tera octets !

```
PaxHeader/Huge

30 ctime=1456948527.914039536
30 atime=1456948548.569742108
22 GNU.sparse.major=1
22 GNU.sparse.minor=0
24 GNU.sparse.name=Huge
39 GNU.sparse.realsize=128574140715008

0000
0
4096
1627791495168
4096
1627791519744
4096
11168083808256
4096
11168083820544
4096
```

Bien que le filesystem ext4 supporte les « sparse files », la taille maximale d'un fichier est de 16To.

La solution la plus simple est d'utiliser un filesystem qui supporte des fichiers plus grands. XFS qui supporte des fichiers de 8 exabytes devrait convenir.

On tente de « détarrer » dans un volume XFS. Cette fois ça marche, on a notre fichier de 128 To.

```
$ tar xvf huge.tar
$ ls -la Huge
-rwxr-xr-x 1 user group  128574140715008  Jan  1  1970 Huge
```

4. L'ELF géant

Il s'agit d'un exécutable ELF64.

```
$ file Huge
Huge: ELF 64-bit LSB executable, x86-64, version 1 (SYSV), statically linked, corrupted section header size
```

La structure du fichier ELF est particulièrement simple. Il ne comporte aucune section header et seulement 3 segments qui nous donnent le mapping de l'exécutable en mémoire.

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

There are no sections in this file.
There are no sections to group in this file.

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

There is no dynamic section in this file.
There are no relocations in this file.
The decoding of unwind sections for machine type Advanced Micro Devices X86-64 is not currently supported.
Dynamic symbol information is not available for displaying symbols.
No version information found in this file.
```

Utilisons gdb pour désassembler le point d'entrée du programme

```
$ gdb Huge

(gdb) b *0x51466a42e705        // Entry point address

(gdb) run

(gdb) x /30i $rip
=> 0x51466a42e705:    sub    $0x1000,%rsp
   0x51466a42e70c:    and    $0xfffffffffffffc00,%rsp
   0x51466a42e713:    mov    %rsp,%rbp
   0x51466a42e716:    xor    %rax,%rax
   0x51466a42e719:    inc    %rax                  // rax=1 : write syscall
   0x51466a42e71c:    xor    %rdi,%rdi
   0x51466a42e71f:    inc    %rdi                  // rdi=1 : stdout
   0x51466a42e722:    movabs $0x5a48156509b7,%rsi    // address of the string : "Please enter the password:"
   0x51466a42e72c:    mov    $0x1b,%edx             // length of the string
   0x51466a42e731:    syscall                  // write("Please enter the password:")
   0x51466a42e733:    xor    %rax,%rax         // rax=0 : read syscall
   0x51466a42e736:    xor    %rdi,%rdi         // rdi=0: stdin
   0x51466a42e739:    mov    %rbp,%rsi         // Address of the input buffer : $rsp
   0x51466a42e73c:    mov    $0x400,%edx       // Length of the input buffer
   0x51466a42e741:    syscall                  // read() : string adr: $rsi
   0x51466a42e743:    mov    %rsp,%rax
```

```
0x51466a42e746:    mov   %rsp,%rdi
0x51466a42e749:    mov   %rsp,%rsi
0x51466a42e74c:    movabs $0x10f338cf000c,%r15
0x51466a42e756:    callq  *%r15
0x51466a42e759:    movabs $0x43abdb4a000c,%rbx
0x51466a42e763:    jmpq   *%rbx
0x51466a42e765:    add    %al,(%rax)
0x51466a42e767:    add    %al,(%rax)
0x51466a42e769:    add    %al,(%rax)
0x51466a42e76b:    add    %al,(%rax)
```

Le programme demande d'entrer un mot de passe. La chaine de caractère entrée par l'utilisateur est mise sur la pile à l'adresse $rsp.

Le programme effectue ensuite un call à l'adresse 0x10f338cf000c.

En désassemblant le code à cette adresse, on ne trouve que des zéros (Correspondant à l'opcode : add %al,(%rax)).

En fait les zéros sont utilisés comme « NOP slide » avant d'attendre le code de la routine.

```
(gdb) x /30xi 0x10f338cf000c
  0x10f338cf000c:    add   %al,(%rax)
  0x10f338cf000e:    add   %al,(%rax)
  0x10f338cf0010:    add   %al,(%rax)
  0x10f338cf0012:    add   %al,(%rax)
  0x10f338cf0014:    add   %al,(%rax)
  0x10f338cf0016:    add   %al,(%rax)
  0x10f338cf0018:    add   %al,(%rax)
  0x10f338cf001a:    add   %al,(%rax)
  0x10f338cf001c:    add   %al,(%rax)
```

Pour trouver les adresses où se trouvent le code réel du programme Huge, on écrit le programme HugeReader (en Annexe) qui :

- localise les zones de données non nulles dans le fichier huge.tar

- Trouve leur offset dans le fichier Huge (en utilisant les offsets des blocs dans l'entête du fichier tar).

- Trouve leur adresses virtuelle dans l'exécutable mappé en mémoire (en utilisant les informations des « programs headers » du fichier ELF).

On obtient la liste suivante :

```
Code Area found: 2, 684
fpos=17affef72ac
vaddr=2c7affef62ac
Code Area found: 2, 788
fpos=17affef7314
vaddr=2c7affef6314
Code Area found: 4, 3022
fpos=a2845ab4bce
vaddr=352845ab3bce
Code Area found: 6, 1874
fpos=17021da9d752
vaddr=42021da9c752
Code Area found: 7, 2798
fpos=18abdb4a1aee
vaddr=43abdb4a0aee
Code Area found: 9, 3608
```

```
fpos=1ee7e541ce18
vaddr=49e7e541be18
Code Area found: 9, 4152
fpos=1ee7e541d038
vaddr=49e7e541c038
Code Area found: 11, 3552
fpos=2b270680fde0
vaddr=4a170682ede0
Code Area found: 12, 1785
fpos=32566a40f6f9
vaddr=51466a42e6f9
Code Area found: 14, 1316
fpos=3adb440a5524
vaddr=59cb440c4524
Code Area found: 15, 2475
fpos=3b58156319ab
vaddr=5a48156509ab
Code Area found: 17, 880
fpos=50e4b0dd0370
vaddr=6f4b0e0f370
Code Area found: 19, 1446
fpos=538a380185a6
vaddr=99a380575a6
Code Area found: 19, 1568
fpos=538a38018620
vaddr=99a38057620
Code Area found: 21, 1352
fpos=5ae338cb8548
vaddr=10f338cf7548
Code Area found: 23, 3620
fpos=746ee246be24
vaddr=2a7ee24aae24
```

La zone de code située après l'adresse 0x10f338cf000c est en 0x10f338cf7548. On trouve le code de la routine qui est appelé après les « NOP slides ».

```
(gdb) x /60i 0x10f338cf7548
  0x10f338cf7548:     mov    $0x10,%ecx
  0x10f338cf754d:     movzbl (%rsi),%ebx
  0x10f338cf7550:     sub    $0x30,%bl
  0x10f338cf7553:     jb     0x10f338cf75ac
  0x10f338cf7555:     cmp    $0xa,%bl
  0x10f338cf7558:     jb     0x10f338cf7571
  0x10f338cf755a:     sub    $0x11,%bl
  0x10f338cf755d:     jb     0x10f338cf75ac
  0x10f338cf755f:     cmp    $0x6,%bl
  0x10f338cf7562:     jb     0x10f338cf756e
  0x10f338cf7564:     sub    $0x20,%bl
  0x10f338cf7567:     jb     0x10f338cf75ac
  0x10f338cf7569:     cmp    $0x6,%bl
  0x10f338cf756c:     jae    0x10f338cf75ac
  0x10f338cf756e:     add    $0xa,%bl
  0x10f338cf7571:     inc    %rsi
  0x10f338cf7574:     shl    $0x8,%ebx
  0x10f338cf7577:     mov    (%rsi),%bl
  0x10f338cf7579:     sub    $0x30,%bl
  0x10f338cf757c:     jb     0x10f338cf75ac
  0x10f338cf757e:     cmp    $0xa,%bl
  0x10f338cf7581:     jb     0x10f338cf759a
  0x10f338cf7583:     sub    $0x11,%bl
  0x10f338cf7586:     jb     0x10f338cf75ac
  0x10f338cf7588:     cmp    $0x6,%bl
  0x10f338cf758b:     jb     0x10f338cf7597
  0x10f338cf758d:     sub    $0x20,%bl
  0x10f338cf7590:     jb     0x10f338cf75ac
  0x10f338cf7592:     cmp    $0x6,%bl
  0x10f338cf7595:     jae    0x10f338cf75ac
  0x10f338cf7597:     add    $0xa,%bl
  0x10f338cf759a:     inc    %rsi
```

```
  0x10f338cf759d:    shl   $0x4,%bl
  0x10f338cf75a0:    shr   $0x4,%bx
  0x10f338cf75a4:    mov   %bl,(%rdi)
  0x10f338cf75a6:    inc   %rdi
  0x10f338cf75a9:    loop  0x10f338cf754d
  0x10f338cf75ab:    retq
  0x10f338cf75ac:    add   %al,(%rax)
  0x10f338cf75ae:    add   %al,(%rax)
  0x10f338cf75b0:    add   %al,(%rax)
================================
Convertit chaine hexa ASCII en hexa.
longueur : 16 bytes.
$rsi : source chaine ascii
$rdi : buffer hexa
```

Cette routine convertit une chaine ASCII de 32 caractères hexadécimaux à l'adresse $rsi en liste d'entier dans le buffer à l'adresse $rdi. Si un des caractères n'est pas dans l'intervalle '0'-'9' ,'a'-'f' ou 'A'-'F', la routine saute dans le néant à l'adresse « 0x10f338cf75ac ».

Au retour de cette routine le programme saute à l'adresse $0x43abdb4a000c. Après les « NOP slides », on trouve le code à l'adresse 0x43abdb4a0aee.

```
(gdb) x /30i $rip
=> 0x43abdb4a0aee:    nop
  0x43abdb4a0aef:    movabs $0x43abdb4a0b0b, %rbx
  0x43abdb4a0af9:    cmpb   $0x29, (%rsp)
  0x43abdb4a0afd:    jne    0x43abdb4a0b09
  0x43abdb4a0aff:    movabs $0x4a170682000c, %rbx
  0x43abdb4a0b09:    jmpq   *%rbx
```

Ce code vérifie que le premier octet du mot de passe est bien 0x29. S'il est différent, le programme saute dans le néant à l'adresse 0x43abdb4a0b0b sinon on continue à l'adresse 0x4a170682000c.

0x7fffffffd400: **0x29** XX XX XX XX XX XX XX

0x7fffffffd408: XX XX XX XX XX XX XX XX

Le bloc de code suivant est à l'adresse 0x4a170682ede0.

```
  0x4a170682ede0:    nop
  0x4a170682ede1:    cmpw   $0xd17e, 0x2(%rsp)
  0x4a170682ede8:    movabs $0x6f4b0e0000c, %rbx
  0x4a170682edf2:    movabs $0x4a170682ee02, %r14
  0x4a170682edfc:    cmovne %r14, %rbx
  0x4a170682ee00:    jmpq   *%rbx
  0x4a170682ee02:    add    %al, (%rax)
  0x4a170682ee04:    add    %al, (%rax)
```

Ce code vérife que les octets 2 et 3 du password sont à 0x7e et 0xd1. Si c'est bon on continue en $0x6f4b0e0000c, sinon on saute dans le vide en 0x4a170682ee02.

0x7fffffffd400: 0x29 XX **0x7e** **0xd1** XX XX XX XX

0x7fffffffd408: XX XX XX XX XX XX XX XX

Le bloc de code suivant est à l'adresse 0x6f4b0e0f370.

```
0x6f4b0e0f370:      cmpb   $0x8c,0xb(%rsp)
0x6f4b0e0f375:      movabs $0x6f4b0e0f38f,%rbx
0x6f4b0e0f37f:      movabs $0x49e7e541000c,%r14
0x6f4b0e0f389:      cmove  %r14,%rbx
0x6f4b0e0f38d:      jmpq   *%rbx
0x6f4b0e0f38f:      add    %al,(%rax)
0x6f4b0e0f391:      add    %al,(%rax)
```

On vérifie que l'octet 12 du password est à 0x8c. Si c'est bon on continue en $0x49e7e541000c, sinon on saute dans le vide en 0x6f4b0e0f38f.

0x7fffffffd400: 0x29 XX 0x7e 0xd1 XX XX XX XX

0x7fffffffd408: XX XX XX **0x8c** XX XX XX XX

Le bloc de code suivant est à l'adresse 0x49e7e541be18.

```
(gdb) x /30i 0x49e7e541be18
  0x49e7e541be18:     nop
  0x49e7e541be19:     push   %rax
  0x49e7e541be1a:     lea    0x10b(%rsp),%rax                    // al = 03
  0x49e7e541be22:     movzbq 0x9(%rsp),%rbx                      // 2nd byte of the password in rbx
  0x49e7e541be28:     movb   $0x0,(%rax)
  0x49e7e541be2b:     lea    0x6(%rip),%rcx       # 0x49e7e541be38
  0x49e7e541be32:     lea    (%rcx,%rbx,2),%rcx
  0x49e7e541be36:     jmpq   *%rcx
  0x49e7e541be38:     add    %al,(%rax)
```

Ce bloc de code :

- Charge la valeur 3 dans al
- Met à zéro un compteur à l'adresse (%rax)
- saute à l'adresse 0x49e7e541be38 + 2* (2ieme octet du mot de passe).

Maintenant les "NOP slides" add %al,(%rax) ne sont plus des NOP mais incrémentent le compteur à l'adresse (%rax).

On finit par atteindre l'adresse 0x49e7e541c038.

```
(gdb) x /30i 0x49e7e541c038
  0x49e7e541c038:     cmpb   $0x65,(%rax)
  0x49e7e541c03b:     movabs $0x352845ab000c,%rbx
  0x49e7e541c045:     movabs $0x49e7e541c056,%r14
  0x49e7e541c04f:     cmovne %r14,%rbx
  0x49e7e541c053:     pop    %rax
  0x49e7e541c054:     jmpq   *%rbx
  0x49e7e541c056:     add    %al,(%rax)
  0x49e7e541c058:     add    %al,(%rax)
  0x49e7e541c05a:     add    %al,(%rax)
```

La valeur attendue dans le compteur (%rax) est 0x65.

Il faut trouver la valeur de l'octet du password correspondante :

((0xc038 – 0xbe38 – 2 * bl)/2) * 3 = 0x65 (mod 256)

(0x100 – bl)* 3 = 0x65 (mod 256)

On trouve bl = 0x89.

Si la valeur est correcte, on saute en $0x352845ab000c.

0x7fffffffd400: 0x29 **0x89** 0x7e 0xd1 XX XX XX XX

0x7fffffffd408: XX XX XX 0x8c XX XX XX XX

La suite est en 0x352845ab3bce.

```
(gdb) x /30i 0x352845ab3bce
  0x352845ab3bce:     lea    0x0(%rip),%rbx       # 0x352845ab3bd5
  0x352845ab3bd5:     xor    0xc(%rsp),%ebx
  0x352845ab3bd9:     cmp    $0xa9b00f5c,%ebx
  0x352845ab3bdf:     movabs $0x352845ab3bf9,%rbx
  0x352845ab3be9:     movabs $0x59cb440c000c,%r14
  0x352845ab3bf3:     cmove  %r14,%rbx
  0x352845ab3bf7:     jmpq   *%rbx
  0x352845ab3bf9:     add    %al,(%rax)
  0x352845ab3bfb:     add    %al,(%rax)
```

Ici le code vérifie que le xor des 4 derniers octets du password avec 0x45ab3bd5 est égal à 0xa9b00f5c.

Les 4 derniers octets du password sont donc 0xEC1B3489 = 0x45ab3bd5 xor 0xa9b00f5c

Si la valeur est correcte, on continue en 0x59cb440c000c.

0x7fffffffd400: 0x29 0x89 0x7e 0xd1 XX XX XX XX

0x7fffffffd408: XX XX XX 0x8c **0x89 0x34 0x1b 0xec**

```
(gdb) x /30i 0x59cb440c4524
  0x59cb440c4524:     movabs $0x59cbc8cc0b83,%rbx
  0x59cb440c452e:     mov    0x19(%rip),%rcx       # 0x59cb440c4556
  0x59cb440c4535:     push   %rax
  0x59cb440c4536:     mov    0x10(%rsp),%eax
  0x59cb440c453a:     xor    %rbx,%rax
  0x59cb440c453d:     movabs $0x2a7ee24a000c,%rbx
  0x59cb440c4547:     cmpxchg %rbx,%rcx
```

```
0x59cb440c454b:    pop   %rax
0x59cb440c454c:    jmpq  *%rcx
0x59cb440c454e:    push  %rsi
0x59cb440c454f:    rex.RB or $0x44,%al
0x59cb440c4552:    lret
0x59cb440c4553:    pop   %rcx
0x59cb440c4554:    add   %al,(%rax)
```

Ce bloc de code vérifie que le xor des octets 8 à 11 du mot de passe avec 0x59cb c8cc0b83 est égal à 0x59cb 440c 4556.

0x59cb c8cc0b83 xor 0x59cb 440c 4556 = 0x8cc04ed5

On trouve 0x8cc04ed5 pour les octets du password.

0x7fffffffd400: 0x29 0x89 0x7e 0xd1 XX XX XX XX

0x7fffffffd408: **0xd5 0x4e 0xc0 0x8c** 0x89 0x34 0x1b 0xec

Si la valeur est correcte, on continue en 0x2a7ee24a000c.

```
(gdb) x /30i 0x2a7ee24aae24
  0x2a7ee24aae24:    nop
  0x2a7ee24aae25:    push  %rax
  0x2a7ee24aae26:    lea   0x18(%rsp),%rdi
  0x2a7ee24aae2b:    lea   0x42(%rip),%rsi       # 0x2a7ee24aae74 // adr. buffer constantes  0x3f fe 24 b8 78 38 1e 71 6d cb
  0x2a7ee24aae32:    mov   $0xa,%ecx
  0x2a7ee24aae37:    rep movsb %ds:(%rsi),%es:(%rdi)
  0x2a7ee24aae39:    mov   0xc(%rsp),%ebx
  0x2a7ee24aae3d:    xor   %ebx,0x1c(%rsp)
  0x2a7ee24aae41:    fclex
  0x2a7ee24aae44:    fldt  0x18(%rsp)
  0x2a7ee24aae48:    fld   %st(0)
  0x2a7ee24aae4a:    fcos
  0x2a7ee24aae4c:    fcompp
  0x2a7ee24aae4e:    fstsw %ax
  0x2a7ee24aae51:    and   $0xffdf,%ax
  0x2a7ee24aae55:    cmp   $0x4000,%ax
  0x2a7ee24aae59:    movabs $0x99a3805000c,%rbx
  0x2a7ee24aae63:    movabs $0x2a7ee24aae7e,%r14
  0x2a7ee24aae6d:    cmovne %r14,%rbx
  0x2a7ee24aae71:    pop   %rax
  0x2a7ee24aae72:    jmpq  *%rbx
  0x2a7ee24aae74:    lret
  0x2a7ee24aae75:    insl  (%dx),%es:(%rdi)
  0x2a7ee24aae76:    jno   0x2a7ee24aae96
  0x2a7ee24aae78:    cmp   %bh,-0x48(%rax)
  0x2a7ee24aae7b:    and   $0xfe,%al
  0x2a7ee24aae7d:    (bad)
  0x2a7ee24aae7e:    add   %al,(%rax)
  0x2a7ee24aae80:    add   %al,(%rax)
  0x2a7ee24aae82:    add   %al,(%rax)
```

Cette partie du code :

- Copie dans un buffer sur la pile une suite de 10 octets : 0x3f fe 24 b8 78 38 1e 71 6d cb
- Calcul le xor des octets 24 b8 78 38 octets dans le buffer avec les octets 5 à 10 du password
- Vérifie que la valeur flottante dans le buffer de 10 octets est le point fixe du cosinus (e.g. cos(x) = x).

Le programme suivant nous donne la valeur du point fixe du cosinus. (NB: Il faut utiliser des long double pour avoir des nombres flottant sur 80 bits).

```
#include <stdio.h>
#include <math.h>

void dump(unsigned char *ptr, int lg)
{
  int i;

  for (i=0; i<lg; i++)
     printf("%02X ",*ptr++);

  printf("\n");
}

int main()
{
    long double theta=0.75;
    int i;

    for (i=0; i<100; i++)
        theta = cosl (theta);

   printf("%Lf\n",theta);
    dump ((unsigned char *)&theta, sizeof(theta));

return(0);
}
```

On trouve 3F FE BD 34 AE EC 1E 71 6D CC comme point fixe du cosinus.

On trouve les octets du password BD 34 AE EC xor 24 B8 78 38 = 99 8C D6 D4

0x7fffffffd400: 0x29 0x89 0x7e 0xd1 **0xd4 0xd6 0x8c 0x99**

0x7fffffffd408: 0xd5 0x4e 0xc0 0x8c 0x89 0x34 0x1b 0xec

On a finalement trouvé la valeur du mot de passe : 29897ed1d4d68c99d54ec08c89341bec.

Le programme nous donne la clef de validation du challenge.

```
Please enter the password: 29897ed1d4d68c99d54ec08c89341bec
The key is: E574B514667F6AB2D83047BB871A54F5
```

4. Level 3

1. USB

Le challenge est constitué d'une archive usb.zip qui contient deux fichiers img.bz2 et userSSTIC.bin.

1. Analyse de l'image img.bz2

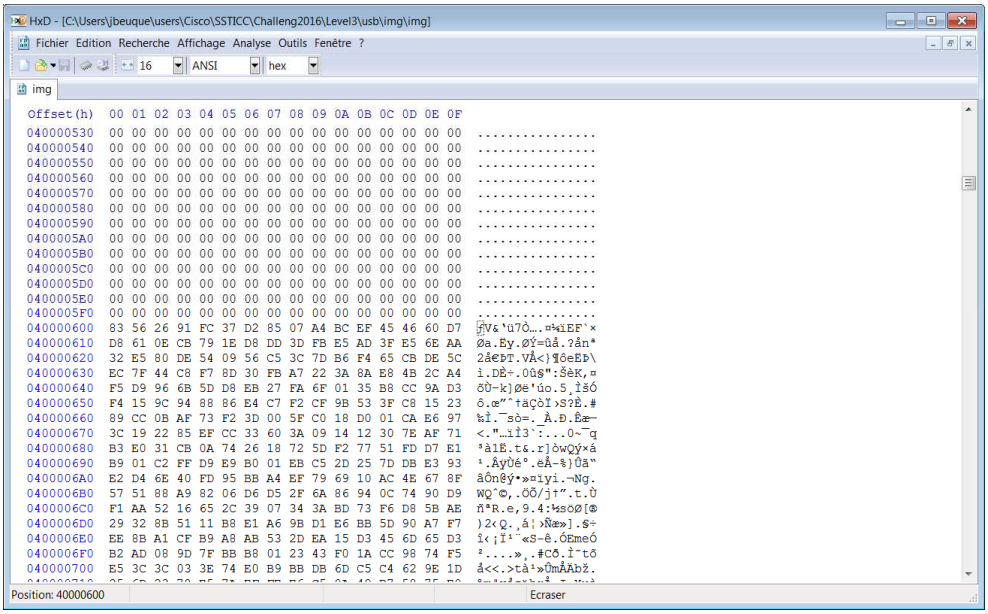
Le fichier img.bz2 après décompression bzip contient une image raw d'un disque de 8G.

```
$ fdisk -l img

Disk img: 7.5 GiB, 8004829184 bytes, 15634432 sectors
Units: sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disklabel type: dos
Disk identifier: 0x7766c1e1

Device Boot  Start    End      Sectors  Size  Id  Type
img1         3        2097154  2097152  1G    b   W95 FAT32
img2         2099201  3147775  1048575  512M  b   W95 FAT32
img3         3147777  3168256  20480    10M   b   W95 FAT32
img4         3168257  3987455  819199   400M  b   W95 FAT32
```

Cette image contient 4 partitions FAT32.

Quand on essaye de monter ces partitions on constate qu'elles sont vides à l'exception de la troisième partition qui contient le fichier *fsfs-ii-2.pdf* (Le livre « Free Software, Free Society » de Richard Stallman).

On remarque un espace anormalement grand entre la fin de la partition 1 et le début de la partition 2. En dumpant cette zone, on voit qu'elle semble contenir des données chiffrées.



De même, la zone située après la fin de la 4ème partition semble également contenir des données chiffrées.

⇨ Cette image disque doit contenir des données cryptées situées entre les partitions.

2. Analyse du programme userSSTIC.bin

Outil : IDA

Le fichier userSSTIC.bin contient un programme exécutable windows 64 bits.

Le programme principale commence avec la fonction sub_140001F00().

La fonction sub_140001F00() du programme effectue les opérations suivantes :

- Création d'un thread avec la fonction de démarrage InitDriver()
- Création d'un thread avec la fonction de démarrage sub_1400018D0()
- Attente de synchronisation sur l'évènement hHandle : WaitForSingleObject

Le thread InitDriver effectue les opérations suivantes :

- Création du fichier « drvSSTIC.sys » à partir d'une ressource binaire dans le fichier userSSTIC.bin.
- Chargement et démarrage du driver « drvSSTIC.sys » en appelant le service control Manager : OpenSCManager / CreateServiceA / StartServiceA.

Le thread sub_1400018D0() effectue les opérations suivantes :

- Enregistre une window class : RegisterClassEx
- Création d'une fenêtre cachée : CreateWindowExA /ShowWindow(SW_HIDE)
- Boucle des messages windows : GetMessageA / TranslateMessage / DispatchMessageA

La fonction callback de la fenêtre (qui a été enregistré lors de l'appel à RegisterClassEx) est la fonction sub_140001800().

La fonction windows callback sub_140001800 :

Cette fonction teste si le message reçu est :

(uMsg == WM_DEVICECHANGE, WPARAM == DBT_DEVICEARRIVAL et LPARAM. dbcv_devicetype == DBT_DEVTYP_VOLUME)

⇨ Ce message est envoyé quand une clef USB est insérée et que des volumes de stockage deviennent disponibles.

Pour chaque volume logique du device inseré la fonction sub_140001720() est appelé avec en paramètre la lettre du volume.

La fonction sub_140001720() :

- Appel les fonctions ZwQueryInformationProcess() et ZwSetInformationProcess() afin de mettre la lettre du volume dans l'octet de poids faible du champ « Default Hard Error Mode » du process courant.

- Appel SetEvent sur le hHandle pour débloquer le thread principal du process userSSTIC.

Le thread principal dans la fonction sub_140001F00 reprend son exécution :

- Création d'un thread avec la fonction de démarrage sub_140001890 ()
- Attente de la terminaison du thread sub_140001890 : WaitForSingleObject

Le thread sub_140001890 effectue les opérations suivantes :

- Boucle de polling : Tant que le dword à l'adresse 7FFE02C8h n'a pas changé faire un sleep d'1 seconde.
- Dès que la valeur a changé, on sauvegarde le dword à l'adresse 7FFE02C8h dans la variable globale qword_14001CB28.
- Fin du thread
- ⇨ NB : La variable à l'adresse 7FFE02C8h contient l'adresse d'un buffer partagé qui a été alloué par le driver drvSSTIC.sys.

Le thread principal dans la fonction sub_140001F00 reprend son exécution :

- 4 évènements de synchronisation : E0 à E3 sont créés dans le buffer partagé avec le driver : CreateEventA.
- Appel de la fonction sub_1400015D0()

La fonction sub_1400015D0() effectue les opérations suivantes :

- Parcours des fichiers dans le répertoire c:\SSTIC\
- Pour chaque fichier :
    - Attente sur l'évènement E0 : WaitForSingleObject.
    - Reset de l'évènement E0. : ResetEvent
    - Appel de la fonction sub_140001450() avec en paramètre le nom du fichier courant.
    - Activation de l'évènement E1. : SetEvent.

La fonction sub_140001450(filename) :

- Récupère la longueur du fichier : GetFileSize
- Place la longueur du fichier dans le buffer partagé à l'offset +32.
- Lit les données du fichier dans le buffer partagé à partir de l'offset +52.
- Appel la fonction sub_140001300() pour calculer le MD5 du fichier et placer celui dans le buffer partagé à l'offset +36.

De retour dans la fonction principale sub_140001F00():

- Activation de l'événement E3 : SetEvent

- Attente sur l'événement E2 : WaitForSingleObject.
- Appel de la fonction sub_140001CE0() pour arrêter le driver : ControlService(SERVICE_CONTROL_STOP).

3. Analyze du driver dvrSSTIC.sys

La fonction de démarrage du driver est la fonction DriverEntry.

Elle appel la fonction principale du driver sub_11BF0().

La fonction sub_11BF0() effectue les opérations suivantes :

- Enregistrement du device : IoCreateDevice
- Appel de la fonction sub_11B04()

La fonction sub_11B04() effectue les opérations suivantes :

- Création d'un thread avec la fonction de démarrage sub_118D8
- Attente de la fin du thread

Le thread sub_118D8 effectue les opérations suivantes :

- Boucle de polling :
    - Appel de ZwQuerySystemInformation pour obtenir le tableau system process information
    - Pour chaque process, appel de ZwQueryInformationProcess pour obtenir le « Default Hard Error Mode »
    - On vérifie si l'octet de poids faible du « Default Hard Error Mode » est compris entre 'A' et 'z'
        - Si c'est le cas, on a trouvé le process « userSSTIC », on retourne la valeur du processId et l'octet de poids faible du « Default Hard Error Mode » qui est la valeur de la lettre du volume de stockage. Fin du thread.
    - Faire un Sleep du thread : KeDelayExecutionThread

On reprend l'exécution dans la fonction sub_11B04() qui retourne la chaine « \Dos\Device\<lettre du volume trouvé> ».

On retourne dans la fonction principale sub_11BF0 :

- Appel de la fonction sub_11728 :
    - Cette fonction démarre un thread dans le process « userSSTIC » qui va allouer un buffer partagé de 2 Mbytes. L'adresse du buffer alloué est dans l'adresse FFFFF780000002C8. (Le process userSSTIC effectue un polling sur cet emplacement qui correspond à l'adresse 7FFE02C8h dans l'espace mémoire user).
- Appel de la fonction sub_12C8C :
    - Cette fonction trouve le disque physique associé au volume « \Dos\Device\<lettre du volume trouvé> ». \\??\PhysicalDrive<i>", lit la table de partition du disque (Appel IOCTL_DISK_GET_DRIVE_LAYOUT_EX) et renseigne la structure à l'adresse qword_2151C0

```
[qword_2151C0 + 0] = 0xC1C1C1C1
[qword_2151C0 + 24] : Disk Size in bytes on 64 bits.
[qword_2151C0 +  4] : Nb of partition.
[qword_2151C0 +  32] :  Offset du premier secteur vide (0)  : 0x0A00.
[qword_2151C0 +  40] :  Offset ecriture dans secteur libre
[qword_2151C0 +  52] : Start offset in bytes of 1st partition
[qword_2151C0 +  92] : End offset in bytes of 1st partition
[qword_2151C0 +  52 + 8*i] : Start offset in bytes of ieme partition
[qword_2151C0 +  92 + 8*i] : End offset in bytes of ieme partition
[qword_2151C0 + 124] = 0xC1C1C1C1
```

- Appel de la fonction sub_12F60 : Ecriture sur le disque dans le premier secteur libre du bloc d'information qword_2151C0.
- Création d'un thread dans le process userSSTIC avec la fonction de démarrage sub_1148C().

Le thread sub_1148C() est utilisé pour positionner un flag quand le process userSSTIC a terminé la liste des fichiers à envoyer au driver. Il effectue les opérations suivantes :

- Attente de l'évènement E3 : KeWaitForSingleObject
- Positionne le flag dword_215180 à true

- Boucle de traitement des fichiers envoyé par le process userSSTIC :
    - Activation de l'événement E0 : KeSetEvent
    - Attente de l'évènement E1 : KeWaitForSingleObject
    - Reset de l'évènement E0 : KeResetEvent
    - Création d'un thread dans le process userSSTIC avec la fonction de démarrage sub_1110C.
    - Incrémentation du compteur dword_15128 : Nombre de fichier traité.
    - Reset de l'évènement E1. : KeResetEvent
    - Si le flag dword_215180 est à true // Tous les fichiers ont été reçu.
        - On active l'évenement E2 : KeSetEvent
        - On appel la fonction sub_13020()

- Fin de la boucle.

La fonction sub_1110C effectue les opérations suivantes :

- Tirage aléatoire d'une clef de 128 bits : RtlRandomEx
- Chiffrement RC4 des données du fichier envoyé par le process userSSTIC dans le buffer partagé.
- Allocation d'un nouveau bloc dans la liste circulaire des blocs de data.

```
qword_15120 : Liste circulaire des fichiers à écrire (crée dans sub_1110C) (écrit dans  sub_13020)
        * Block info : 64 octets
                +0 : Addr Next block
                +8 : Addr  qword_15120
                +16: Data size +36 in bytes (32 bits)
                +20: Data size in bytes (32 bits)
                +24: Clef RC4 (128 bits)
                +40: MD5 of the file data (128 bits)
                +56: Addr bloc data (64 bits)
        * Block Data: Data file size.  (Payload : RC4 encrypted).
```

La fonction sub_13020 effectue les opérations suivantes :

- Parcours de la liste circulaire des blocs de données à écrire :
    - Copie dans un buffer des informations suivantes pour chaque bloc :
        - +0: size of payload in bytes : 32 bits
        - +4: Clef RC4: 128 bits
        - +20: MD5 of the file data
        - +36: Data payload (RC4 encrypted)
    - Chiffrement RC6 (mode CBC) du buffer avec la clef fixe « 551C2016B00B5F00 ». (NB : le chiffrement RC6 a été modifié, le sens des rotations a été inversé.)
    - Ecriture sur le disque des informations suivantes (Les données sont écrites avant la première partition, entre les partitions et après la fin de la dernière):
        - Entête :
        - Nb_fichier: 32bits.
        - "551C2016B00B5F00" : ASCII string 128 bits.
        - Buffer RC6 crypté.

4. Extraction et déchiffrement des fichiers cachés

On peut maintenant extraire et déchiffrer les données de l'image de la clef USB.

Le programme extractData permet d'extraire les données qui sont situées entre les partitions de l'image du disque.

Le programme decryptDataFS déchiffre les données extraite avec du RC6 en mode CBC avec la clef « 551C2016B00B5F00 ».

Le programme decryptFiles déchiffre les fichiers avec du RC4 (avec les clefs disponibles dans les entêtes des blocs).

On trouve 8 fichiers :

Le premier fichier est un fichier texte qui nous donne un mot de passe pour une archive zip :

```
password for the zip file : !WooYouAreSuchAnAwesomeGuy!
```

Le 5ième fichier est une archive zip cryptée qui contient une image jpeg et un fichier nommé key.

On ouvre le zip avec le mot de passe ci-dessus pour trouver la clef du challenge dans le fichier key :

09 28 BD E1 E3 ED 89 69 86 32 DB FF 4A 23 11 38.

2. Video

Le challenge est constitué de l'archive video.zip.

Cette archive contient :

- Une vidéo mpeg4 : Airlhes_CYBER_SECRET_possible_exfiltration.mp4
- Un fichier d'installation d'un screen save : Airlhes_screensaver_setup.exe
- Un fichier texte : la lettre de mission : mission.txt

La « lettre de mission » nous explique que le screen saver est utilisé pour exfiltrer des données…

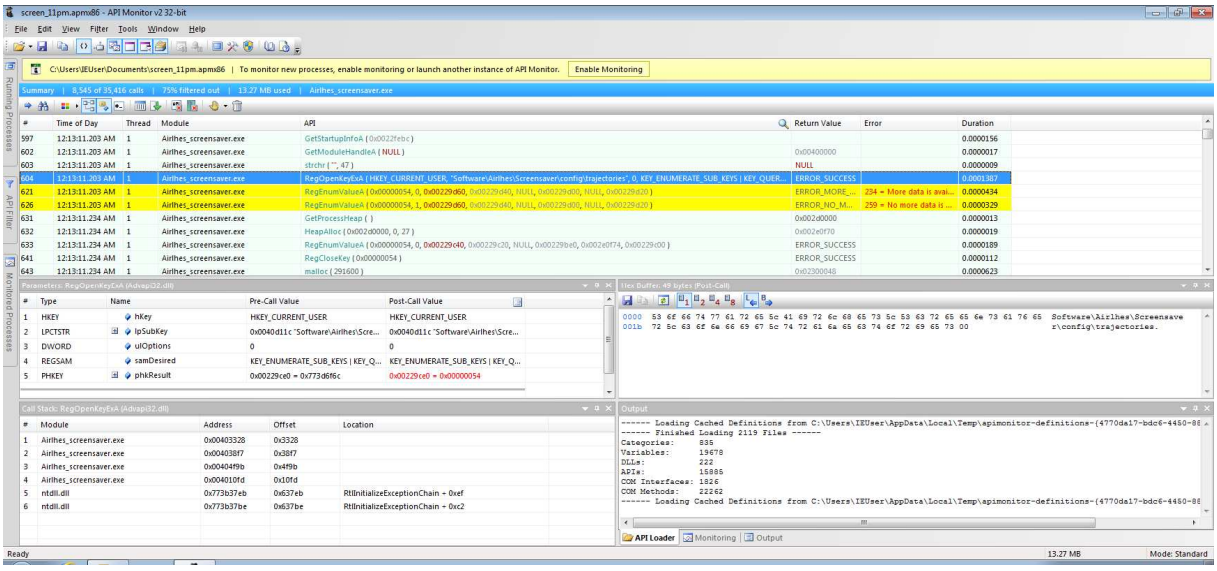
1. Analyse du programme Airlhes_screensaver.scr

Outils : API Monitor, IDA

Après avoir installé le screen saver. On récupere le fichier Airlhes_screensaver.scr qui est un executable windows.

On execute ce programme avec l'outil API Monitor.

On constate que le programme va lire dans la registry la clef « Software\Airlhes\Screensaver\config\trajectories ».



Avec regedit, on trouve la séquence suivante dans la clef Software\Airlhes\Screensaver\config\trajectories.

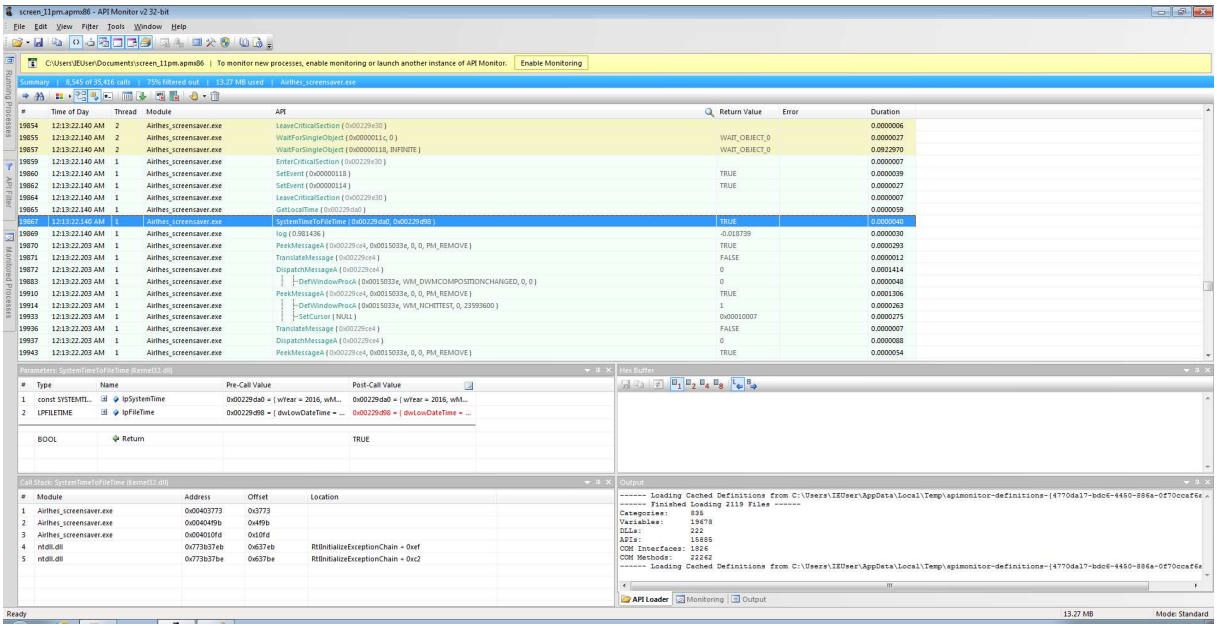
0x78, 0xDA, 0xF3, 0x48, 0xCD, 0xC9, 0xC9, 0x57, 0x48, 0x2B, 0xCA, 0x4C, 0xCD, 0x4B, 0x51, 0xB0, 0xD2, 0x04, 0x00, 0x2B, 0x74, 0x05, 0x10

(NB : Il s'agit d'une séquence compressée avec zlib. Après « décompression », on trouve la chaine : `48 65 6C 6C 6F 20 66 72 69 65 6E 64 20 3A 29`. Soit en ASCII : « `Hello friend :)` ».

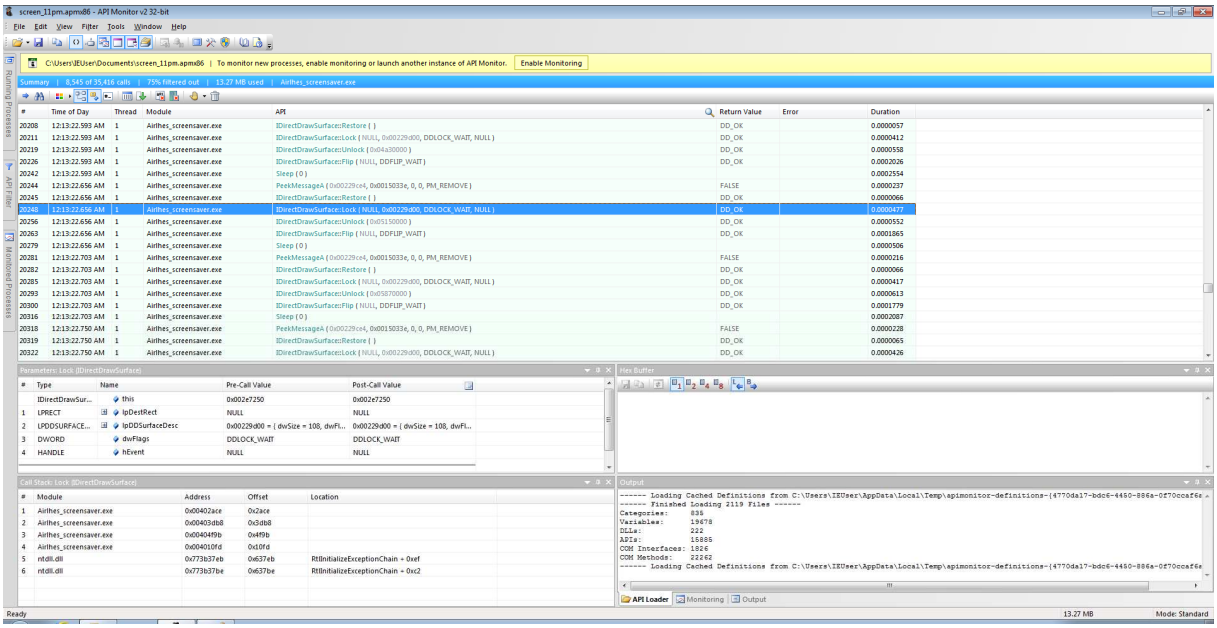
⇨ `La séquence « exfiltrée » par le screen saver doit se trouver dans cette clef de registry.`

On remarque également que le programme recupère l'heure courante en appelant GetLocalTime.

⇨ Le comportement du screen saver change suivant l'heure courante. Entre 23H03 et 0H57, à la place du motif habituel, le screen saver affiche cycliquement une séquence de couleur en pleine écran. Cette séquence de couleur doit dépendre des valeurs dans la clef Software\Airlhes\Screensaver\config\trajectories de la registry.



Le programme utilise DirectDraw pour les graphiques. Avec API monitor, on peut placer un breakpoint sur l'appel à IDirectDrawSurface::Lock. Dés que le programme s'arrête sur le breakpoint, on attache un debugger (IDA) pour analyser le code qui remplit la surface graphique.



La routine d'encodage des couleurs est la suivante :

```
dword_40F120 : cpteur : 0,1,2, 0,1,2, ...
dword_40F100 : cpteur : 0,    1,    2,  .... 0x1A  (27 valeurs)

TlsIndex+74h : Adr 7931C0   // Byte to encode
TlsIndex+94h : Adr 7931E0   // = (TlsIndex+74h) % 7 +1
TlsIndex+B4h : Adr 793200 // Index in the color lookup table dword_405020    = ( (TlsIndex+B4h ) + (TlsIndex+94h) ) & 0x07



loc_403E7E:                 ; CODE XREF: UPX0:00403C01↑j
UPX0:00403E7E mov    esi, dword_40F120   // cpteur 0,1,2 ..
UPX0:00403E84 test   esi, esi
UPX0:00403E86 jnz    loc_403F47
UPX0:00403E8C mov    ecx, dword_40F100   // cpteur
UPX0:00403E92 mov    ebx, [ebp-6084h]
UPX0:00403E98 movzx  eax, cl
UPX0:00403E9B movzx  edx, byte ptr [ebx+eax]
UPX0:00403E9F mov    eax, [ebp-6024h]
UPX0:00403EA5 xor    dl, [eax+ecx]
UPX0:00403EA8 mov    ecx, 25h
UPX0:00403EAD mov    dword_40F120, 1
UPX0:00403EB7 mov    eax, edx
UPX0:00403EB9 mov    byte ptr TlsIndex+74h, dl
UPX0:00403EBF mul    cl             // ax = dl * 25h
UPX0:00403EC1 mov    ecx, edx
UPX0:00403EC3 shr    ax, 8          // ax = (dl*25h)/100h
UPX0:00403EC7 sub    ecx, eax       // ecx = dl - (dl*25h)/100h
UPX0:00403EC9 shr    cl, 1          // ecx = (dl - (dl*25h)/100h)/2
UPX0:00403ECB add    eax, ecx       // eax = (dl*25h)/100h + (dl - (dl*25h)/100h)/2
UPX0:00403ECD shr    al, 2          // eax = ((dl*25h)/100h + (dl - (dl*25h)/100h)/2)/4
UPX0:00403ED0 lea    ecx, ds:0[eax*8] // ecx = (((dl*25h)/100h + (dl - (dl*25h)/100h)/2)/4 ) * 8
UPX0:00403ED7 sub    ecx, eax       // ecx = (((dl*25h)/100h + (dl - (dl*25h)/100h)/2)/4 ) * 7
```

```
UPX0:00403ED9 sub    edx, ecx        // edx = dl - ((dl*25h)/100h + (dl - (dl*25h)/100h)/2)/4 ) * 7
UPX0:00403EDB lea    ecx, [edx+1]    // ecx = dl - ((dl*25h)/100h + (dl - (dl*25h)/100h)/2)/4 ) * 7 + 1   // ecx = (dl %7) +1
UPX0:00403EDE mov    byte ptr TlsIndex+94h, cl
UPX0:00403EE4 add    ecx, TlsIndex+0B4h
UPX0:00403EEA and    ecx, 7
UPX0:00403EED mov    TlsIndex+0B4h, ecx
UPX0:00403EF3
UPX0:00403EF3 loc_403EF3:                        ; CODE XREF: UPX0:00403FC4↓j
UPX0:00403EF3                                    ; UPX0:00403FE2↓j ...
UPX0:00403EF3 mov    ecx, dword_405020[ecx*4]
UPX0:00403EFA mov    [ebp-6070h], ecx



loc_403F47:                          ; CODE XREF: UPX0:00403E86↑j
UPX0:00403F47 movzx  ecx, byte ptr TlsIndex+74h
UPX0:00403F4E mov    ebx, 25h
UPX0:00403F53 mov    eax, ecx
UPX0:00403F55 mov    edx, ecx
UPX0:00403F57 mul    bl          // ax = cl * 25h
UPX0:00403F59 shr    ax, 8       // ax = (cl*25h)/100h
UPX0:00403F5D sub    edx, eax    // edx = cl - (cl*25h)/100h
UPX0:00403F5F shr    dl, 1       // edx = (cl - (cl*25h)/100h) / 2
UPX0:00403F61 add    edx, eax    // edx = (cl*25h)/100h + (cl - (cl*25h)/100h) / 2
UPX0:00403F63 shr    dl, 2       // edx = ((cl*25h)/100h + (cl - (cl*25h)/100h) / 2 ) /4
UPX0:00403F66 mov    eax, edx
UPX0:00403F68 mov    ecx, edx
UPX0:00403F6A mov    byte ptr TlsIndex+74h, dl  // edx = ((cl*25h)/100h + (cl - (cl*25h)/100h) / 2 ) /4        // edx = cl /7
UPX0:00403F70 mul    bl                    // ax = (((cl*25h)/100h + (cl - (cl*25h)/100h) / 2 ) /4 ) * 25h  = dl * 25h
UPX0:00403F72 lea    ebx, [esi+1]             // ebx = dword_40F120 + 1
UPX0:00403F75 shr    ax, 8                 // ax = (dl*25h)/100
UPX0:00403F79 sub    ecx, eax              // ecx = dl - (dl*25h)/100
UPX0:00403F7B shr    cl, 1                 // ecx = (dl - (dl*25h)/100 ) /2
UPX0:00403F7D add    eax, ecx              // eax = (dl*25h)/100 + (dl - (dl*25h)/100 ) /2
UPX0:00403F7F shr    al, 2                 // eax = ((dl*25h)/100 + (dl - (dl*25h)/100 ) /2 ) /4
UPX0:00403F82 lea    ecx, ds:0[eax*8]         // ecx = (((dl*25h)/100 + (dl - (dl*25h)/100 ) /2 ) /4 ) * 8
UPX0:00403F89 sub    ecx, eax              // ecx = (((dl*25h)/100 + (dl - (dl*25h)/100 ) /2 ) /4 ) * 7
UPX0:00403F8B mov    eax, ebx              // eax = dword_40F120 + 1
UPX0:00403F8D sub    edx, ecx              // edx = dl - (((dl*25h)/100 + (dl - (dl*25h)/100 ) /2 ) /4 ) * 7
UPX0:00403F8F lea    ecx, [edx+1]             // ecx = dl - (((dl*25h)/100 + (dl - (dl*25h)/100 ) /2 ) /4 ) * 7 + 1  // ecx = (dl %7) +1
UPX0:00403F92 mov    edx, 55555556h
UPX0:00403F97 imul   edx                   // edx:eax = eax * 55555556h = (dword_40F120 + 1 ) * 55555556h
UPX0:00403F99 mov    eax, ebx              // eax = dword_40F120 + 1
UPX0:00403F9B mov    byte ptr TlsIndex+94h, cl
UPX0:00403FA1 sar    eax, 1Fh              // eax = eax / 2^31 = (dword_40F120 + 1 )/2^31
UPX0:00403FA4 add    ecx, TlsIndex+0B4h
UPX0:00403FAA sub    edx, eax              // edx = ((dword_40F120 + 1 ) * 55555556h)/2^32 - (dword_40F120 + 1 )/2^31
UPX0:00403FAC lea    eax, [edx+edx*2]         // eax = ( ((dword_40F120 + 1 ) * 55555556h)/2^32 - (dword_40F120 + 1 )/2^31 ) * 3
UPX0:00403FAF mov    edx, ebx              // edx = dword_40F120 + 1
UPX0:00403FB1 sub    edx, eax              // edx = dword_40F120 + 1 - ( ((dword_40F120 + 1 ) * 55555556h)/2^32 - (dword_40F120 + 1
)/2^31 ) * 3
UPX0:00403FB3 and    ecx, 7
UPX0:00403FB6 test   edx, edx
UPX0:00403FB8 mov    TlsIndex+0B4h, ecx
UPX0:00403FBE mov    dword_40F120, edx
UPX0:00403FC4 jnz    loc_403EF3
UPX0:00403FCA mov    eax, dword_40F100
UPX0:00403FCF xor    edx, edx
UPX0:00403FD1 add    eax, 1
UPX0:00403FD4 div    dword ptr [ebp-6020h]  // Divide by 0x1B : Remainder in edx , quotient in eax
UPX0:00403FDA test   edx, edx
UPX0:00403FDC mov    dword_40F100, edx
UPX0:00403FE2 jnz    loc_403EF3
UPX0:00403FE8 mov    dword_40C940, 0B003BA88h
UPX0:00403FF2 jmp    loc_403EF3
```

Le principe est le suivant :

- Les octets à encoder sont obtenus en calculant le xor des octets dans la clef de registry « Software\Airlhes\Screensaver\config\trajectories » (prefixé avec la longueur sur 4 octets)

```
009A67A8  17 00 00 00 78 DA F3 48  CD C9 C9 57 48 2B CA 4C
009A67B8  CD 4B 51 B0 D2 04 00 2B  74 05 10
```

avec une clef constante (xor Key).

```
00405041            30 A4 3F 6D 28 04 23 36  2A 32 DC AD 0B A0 4B E8
00405051            20 1F 64 84 0A F4 C4 C7  8A 8D C0 A2 C4 40 19 A1
```

- Chaque octet à encoder est décomposé en base 7. On a donc 3 valeurs en base 7 par octet. L'interet de la base 7 est de permettre un codage des transitions de couleur :
    - On a huit couleurs possibles pour l'écran (8 combinaisons de RGB avec chaque composante à 0 ou à 0xFF).
    - Pour chaque nouvelle valeur (modulo 7), on code une transition de couleur. (On a 7 transitions de couleur possibles puisqu'il y a 8 couleurs).
- La fonction *encode* du programme *ColorDecoder* (en annexe) donne une implémentation en C de cet algorithme.

2. Décodage de la vidéo

On extrait la séquence de couleur de la vidéo à l'aide de FFMPEG. On trouve la séquence:

```
{CYAN, MAGENTA, CYAN, MAGENTA, BLUE, MAGENTA, RED, BLUE, BLACK, RED,  BLUE,  CYAN, WHITE, YELLOW, BLUE, RED, BLACK,
MAGENTA, WHITE, GREEN, YELLOW,BLACK,RED,YELLOW,CYAN
,WHITE,BLACK,CYAN,BLACK,GREEN,WHITE,YELLOW,BLUE,YELLOW,BLUE,MAGENTA,BLUE,MAGENTA,BLACK,WHITE,
BLUE,CYAN,BLUE,GREEN,BLUE,WHITE,BLUE,WHITE,RED,WHITE,GREEN,CYAN,RED,YELLOW,CYAN,GREEN,BLUE,RED,GREEN,CYAN,
RED,YELLOW,MAGENTA,WHITE,GREEN,MAGENTA,CYAN,YELLOW,WHITE,BLACK,RED,CYAN,WHITE,BLACK,CYAN,BLACK,BLUE,RED,WHITE,M
AGENTA,  BLACK,GREEN,RED,MAGENTA};
```

Le programme ColorDecoder (en annexe) permet d'obtenir la séquence décodée suivante :

```
18 00 00 00 78 DA 0B 16 34 17 2B DA C5 58 F9 CB
6E 62 57 F3 BE 7B 5B 00 32 50 07 7E.
```

Cette séquence contient une longueur sur 32 bits suivie de données compressées (24 octets) avec zlib.

Après décompression avec zlib, on trouve la clef du challenge :

53 11 37 16 72 BA 01 79 FA 3E 91 8A 83 BE DE B4

5. Final

On obtient le fichier final.txt.

```
Coucou !

Tu as presque réussi le challenge !
l01p1 y'4qe3553 z41y : 8Y6d5j9Vy88HUGHfGSKsJvqA@ffgvp.bet
```

Après décodage ROT13, on trouve l'adresse email finale:

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

6. Annexes

1. GhostAnalyszer

```
#include <stdio.h>
#include <string.h>
#include <assert.h>
#include "zlib.h"

#if defined(MSDOS) || defined(OS2) || defined(WIN32) || defined(__CYGWIN__)
#  include <fcntl.h>
#  include <io.h>
#  define SET_BINARY_MODE(file) setmode(fileno(file), O_BINARY)
#else
#  define SET_BINARY_MODE(file)
#endif


/*****************************************************************************/
unsigned char buffer[65536];
unsigned char bufferOut[65536];
#define CHUNK 16384
/*****************************************************************************/
int infBuffer(unsigned char *bfIn, int lgIn, unsigned char *bfOut, int *lgOut)
{
   int ret;
   unsigned have;
   z_stream strm;
   unsigned char in[CHUNK];
   unsigned char out[CHUNK];

   /* allocate inflate state */
   strm.zalloc = Z_NULL;
   strm.zfree = Z_NULL;
   strm.opaque = Z_NULL;
   strm.avail_in = 0;
   strm.next_in = Z_NULL;
   ret = inflateInit(&strm);
   if (ret != Z_OK)
      return ret;

   /* decompress until deflate stream ends or end of file */
      *lgOut=0;
      strm.avail_in = lgIn;
      memcpy(in, bfIn, lgIn);
      strm.next_in = in;

      /* run inflate() on input until output buffer not full */
      do {
         strm.avail_out = CHUNK;
         strm.next_out = out;
         ret = inflate(&strm, Z_NO_FLUSH);
         assert(ret != Z_STREAM_ERROR);  /* state not clobbered */
         switch (ret) {
         case Z_NEED_DICT:
            ret = Z_DATA_ERROR;     /* and fall through */
         case Z_DATA_ERROR:
         case Z_MEM_ERROR:
            (void)inflateEnd(&strm);
            return ret;
         }
         have = CHUNK - strm.avail_out;
         memcpy(bfOut, out, have);
         *lgOut+=have;
```

```
                bfOut +=have;

    } while (strm.avail_out == 0);

    /* done when inflate() says it's done */
  if (ret != Z_STREAM_END) printf("Inflate Error\n");

  /* clean up and return */
  (void)inflateEnd(&strm);
  return ret == Z_STREAM_END ? Z_OK : Z_DATA_ERROR;
}


/* report a zlib or i/o error */
void zerr(int ret)
{
  switch (ret) {
  case Z_ERRNO:
       fputs("error reading \n", stderr);
    break;
  case Z_STREAM_ERROR:
    fputs("invalid compression level\n", stderr);
    break;
  case Z_DATA_ERROR:
    fputs("invalid or incomplete deflate data\n", stderr);
    break;
  case Z_MEM_ERROR:
    fputs("out of memory\n", stderr);
    break;
  case Z_VERSION_ERROR:
    fputs("zlib version mismatch!\n", stderr);
  }
}
/*****************************************************************************/
int read32lsb(unsigned char *bf)
{
        int val;

        val=bf[3];
        val<<=8;
        val+=bf[2];
        val<<=8;
        val+=bf[1];
        val<<=8;
        val+=bf[0];

return(val);
}
/*****************************************************************************/
int parseCmd(unsigned char *bufferOut, int lgOut)
{
        int code;
        int lg;
        char aslash=0x5c;
        char *fname;

        static FILE *fdata=NULL;
        static int flength=0;

        code = bufferOut[0];

        if (code == 3)
        {
                printf("Download file: %s\n",bufferOut+1);

                fname = strrchr(bufferOut+1,aslash);

                if (fname != NULL)
                {
                        fname++;
                        fdata = fopen(fname,"wb");
```

```
                }
        }
        else if (code == 2)
        {
                printf("list directory: %s\n",bufferOut+1);
        }
        else if (code == 105)
        {
                lg = read32lsb(bufferOut+5);
                flength = lg;
                printf("File Size = %d\n",lg);
        }
        else if (code == 5)
        {
                printf("File Data \n");
      if (fdata != NULL)
      {
                        fwrite(bufferOut+9, 1, lgOut-9, fdata);
                        flength -= (lgOut-9);
                        if (flength <=0)
                        {
                                fclose(fdata);
                                flength = 0;
                                fdata = NULL;
                        }
                }
        }

        return(0);
}
/*****************************************************************************/
int parseHeader(FILE *f, unsigned char *header, FILE *fout)
{
        int lg, dlg;
        int ret;
    int lgOut;

        if (strncmp(header,"Gh0st",5)!=0)
        {
                printf("Bad Header\n");
        }
        else
        {
                lg = header[8];
                lg <<=8;
                lg += header[7];
                lg <<=8;
                lg += header[6];
                lg <<=8;
                lg += header[5];

                dlg = header[12];
                dlg <<=8;
                dlg += header[11];
                dlg <<=8;
                dlg += header[10];
                dlg <<=8;
                dlg += header[9];
        }

        printf("length = %d, decomp_lg = %d\n", lg, dlg);


        ret = fread(buffer, 1, lg-13, f);

        if (ret != lg -13)
                printf("Bad length\n");
        else
        {
      infBuffer(buffer, lg-13, bufferOut, &lgOut);
                if (lgOut != dlg)
```

```
                                printf("Error: lgOut = %d\n", lgOut);
                        else
                        {
                                printf("Code=%d(0x%x)\n",bufferOut[0], bufferOut[0]);
                                fwrite(header, 1, 13, fout);
                                fwrite(bufferOut, 1, lgOut, fout);

                                parseCmd(bufferOut, lgOut);
                        }
                }
                return(ret);

}
/***********************************/
int rGhost(FILE *f, FILE *fout)
{
                unsigned char header[13];
                int ret;

                do {
                        ret = fread(header, 1, 13, f);

                     if (ret == 13)
                                parseHeader(f, header, fout);
                } while (ret == 13);

                return(0);
}
/***********************************/
int main(int argc, char *argv[])
{
                FILE *fch;
                FILE *fchout;

                if (argc < 3)
                        exit(-1);

                fch = fopen(argv[1],"rb");
                if (fch == NULL)
                        exit(-2);

                fchout = fopen(argv[2],"wb");
                if (fchout == NULL)
                        exit(-2);


                rGhost(fch, fchout);

        fclose(fch);
        fclose(fchout);
                return(0);
}
```

2. HugeReader

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



#define MAXT 51
#define BSIZE 8192
static int nb_block=0;
```

```
static unsigned long long offsettable[MAXT];
static long long bsize[MAXT];

static unsigned char  datablock[MAXT][BSIZE];
/*****************************************************/
static long long vmemtab[6] = { 0x20000,       0x2b0000000000 ,
                        0x2b0000000000, 0x49f000000000,
                        0x49f000000000, 0x600000000000};

static long long vmemfileoffset[3] = { 0x49effffe1000, 0x1000, 0x2affffffe1000};

static long long vmemfileoffset2[6] ;
/*****************************************************/
int loadHuge(FILE *f)
{
 int ret;
 char lg[256];
 int i;
 unsigned long long offst;

 ret = fseek(f, 0x600, SEEK_SET);
 if (ret != 0)
                return(-1);

 if (fgets(lg, 256, f) == NULL)
    return(-1);

 nb_block = atoi(lg);


 for (i=0; i<nb_block*2; i++)
 {
                fgets(lg, 256, f);
                offst = atoll(lg);
                if ((i& 0x1) == 0)
                {
                                offsettable[i] = offst;

                }
                else
                                bsize[i/2] = offst;

 }

 ret = fseek(f, 0x800, SEEK_SET);
 for (i=0; i<nb_block; i++)
 {
                  ret = fread(datablock[i] , 1, bsize[i] , f);
                 printf("%d\n", ret);
 }

 for (i=0; i<nb_block; i++)
 {
                offsettable[2*i+1] = offsettable[2*i] + bsize[i];
 }
 offsettable[2*nb_block] = 0xFFFFFFFFFFFFFFFF;



return(0);
}
/*****************************************************/
long long vmem2file(long long vaddr)
{
 int i;
 long long off;
 long long fpos=0;

 for (i=0; i<3; i++)
 {
                if ((vaddr >= vmemtab[2*i]) && (vaddr < vmemtab[2*i+1]))
```

```
		{
			off = vaddr - vmemtab[2*i];
			fpos = vmemfileoffset[i] + off ;
		}
 }

return(fpos);

}
/*************************************************************/
long long file2vmem(long long fpos)
{
 int i;
 long long off;
 long long vaddr=0;

 for (i=0; i<3; i++)
 {
     vmemfileoffset2[2*i]=vmemfileoffset[i];
     vmemfileoffset2[2*i+1]= vmemfileoffset[i] + (vmemtab[2*i+1] - vmemtab[2*i] );
 }

 for (i=0; i<3; i++)
 {
		if ((fpos >= vmemfileoffset2[2*i]) && (fpos < vmemfileoffset2[2*i+1]))
		{
			off = fpos - vmemfileoffset2[2*i];
			vaddr = vmemtab[2*i] + off ;
		}
 }

return(vaddr);

}
/*****************************************************/
int findCodeArea()
{
 int i,j;
 unsigned char val;
 int cpt;
 long long fpos;
 long long vaddr;

 for (i=0; i<nb_block; i++)
 {
     cpt = 0;

     for (j=0; j<bsize[i]; j++)
     {
		val = datablock[i][j];
		if (val == 0)
		{
			if (cpt <8)
				cpt ++;
		}
		else
		{

			if (cpt==8)
			{
				printf("Code Area found: %d, %d\n",i,j);
				fpos = offsettable[2*i] + j;
				printf("fpos=%llx\n", fpos);
				vaddr = file2vmem(fpos);
				printf("vaddr=%llx\n", vaddr);
			}
			cpt = 0;
	    }
     }
 }
```

```
}
/*****************************************************/

int main(int argc, char *argv[])
{
FILE *fch;
unsigned char buff[4096];
long long fpos;
long long vaddr;


fch = fopen("huge.tar","rb");
if (fch ==NULL)
                return(-1);

loadHuge(fch);

fclose(fch);



findCodeArea();

return(0);

}
```

3. QuestionMark ByteCode

```
0:PUSHB_1
1: 0
2:RS
3:PUSHB_1
4: 1
5:RS
6:PUSHB_1
7: 2
8:RS
9:PUSHB_1
10: 3
11:RS
12:PUSHB_1
13: 4
14:RS
15:PUSHB_1
16: 5
17:RS
18:PUSHB_1
19: 6
20:RS
21:PUSHB_1
22: 7
23:RS
24:PUSHB_1
25: 8
26:RS
27:PUSHB_1
28: 9
```

```
29:RS
30:PUSHB_1
31: 10
32:RS
33:PUSHB_1
34: 11
35:RS
36:PUSHB_1
37: 12
38:RS
39:PUSHB_1
40: 13
41:RS
42:PUSHB_1
43: 14
44:RS
45:PUSHB_1
46: 15
47:RS
48:PUSHB_1
49: 16
50:RS
51:PUSHB_1
52: 17
53:RS
54:PUSHB_1
55: 18
56:RS
57:PUSHB_1
58: 19
59:RS
60:PUSHB_1
61: 20
62:RS
63:PUSHB_1
64: 21
65:RS
66:ROLL
67:PUSHW_1
68: 256
69:MUL
70:PUSHB_1
71: 18
72:MINDEX
73:PUSHW_1
74: 256
75:MUL
76:PUSHB_1
77: 20
78:MINDEX
79:PUSHW_1
80: 19
81:EQ
82:PUSHB_1
83: 99
84:RS
85:AND
86:PUSHB_1
87: 99
88:SWAP
89:WS
90:PUSHB_1
91: 10
92:MINDEX
93:PUSHB_1
94: 6
95:ADD
96:PUSHB_1
97: 4
98:MINDEX
99:PUSHB_1
```

```
100: 6
101:ADD
102:PUSHB_1
103: 10
104:MINDEX
105:PUSHW_1
106: 320
107:MUL
108:PUSHB_1
109: 11
110:MINDEX
111:PUSHW_1
112: 384
113:MUL
114:PUSHW_1
115: 186
116:EQ
117:PUSHB_1
118: 99
119:RS
120:AND
121:PUSHB_1
122: 99
123:SWAP
124:WS
125:PUSHB_1
126: 18
127:MINDEX
128:PUSHW_1
129: 63
130:EQ
131:PUSHB_1
132: 99
133:RS
134:AND
135:PUSHB_1
136: 99
137:SWAP
138:WS
139:ROLL
140:PUSHW_1
141: 320
142:MUL
143:PUSHB_1
144: 14
145:MINDEX
146:PUSHW_1
147: 39
148:EQ
149:PUSHB_1
150: 99
151:RS
152:AND
153:PUSHB_1
154: 99
155:SWAP
156:WS
157:PUSHB_1
158: 7
159:SUB
160:PUSHB_1
161: 14
162:MINDEX
163:PUSHW_1
164: 35
165:EQ
166:PUSHB_1
167: 99
168:RS
169:AND
170:PUSHB_1
```

```
171: 99
172:SWAP
173:WS
174:PUSHB_1
175: 13
176:MINDEX
177:PUSHB_1
178: 2
179:ADD
180:PUSHB_1
181: 6
182:MINDEX
183:PUSHB_1
184: 6
185:ADD
186:PUSHB_1
187: 16
188:MINDEX
189:PUSHW_1
190: 256
191:MUL
192:PUSHB_1
193: 17
194:MINDEX
195:PUSHB_1
196: 3
197:SUB
198:PUSHB_1
199: 8
200:MINDEX
201:PUSHB_1
202: 1
203:ADD
204:PUSHB_1
205: 11
206:MINDEX
207:PUSHB_1
208: 2
209:ADD
210:PUSHB_1
211: 10
212:MINDEX
213:PUSHB_1
214: 7
215:SUB
216:SWAP
217:PUSHB_1
218: 4
219:ADD
220:PUSHB_1
221: 8
222:MINDEX
223:PUSHW_1
224: 263
225:EQ
226:PUSHB_1
227: 99
228:RS
229:AND
230:PUSHB_1
231: 99
232:SWAP
233:WS
234:PUSHB_1
235: 16
236:MINDEX
237:PUSHW_1
238: 26
239:EQ
240:PUSHB_1
241: 99
```

```
242:RS
243:AND
244:PUSHB_1
245: 99
246:SWAP
247:WS
248:PUSHB_1
249: 7
250:MINDEX
251:PUSHB_1
252: 7
253:ADD
254:PUSHB_1
255: 9
256:MINDEX
257:PUSHW_1
258: 28
259:EQ
260:PUSHB_1
261: 99
262:RS
263:AND
264:PUSHB_1
265: 99
266:SWAP
267:WS
268:PUSHB_1
269: 7
270:MINDEX
271:PUSHW_1
272: 192
273:MUL
274:PUSHB_1
275: 8
276:MINDEX
277:PUSHB_1
278: 2
279:SUB
280:PUSHB_1
281: 12
282:MINDEX
283:PUSHB_1
284: 6
285:SUB
286:PUSHB_1
287: 10
288:MINDEX
289:PUSHW_1
290: 21
291:EQ
292:PUSHB_1
293: 99
294:RS
295:AND
296:PUSHB_1
297: 99
298:SWAP
299:WS
300:PUSHB_1
301: 13
302:MINDEX
303:PUSHB_1
304: 5
305:ADD
306:PUSHB_1
307: 9
308:MINDEX
309:PUSHW_1
310: 53
311:EQ
312:PUSHB_1
```

```
313: 99
314:RS
315:AND
316:PUSHB_1
317: 99
318:SWAP
319:WS
320:SWAP
321:PUSHW_1
322: 21
323:EQ
324:PUSHB_1
325: 99
326:RS
327:AND
328:PUSHB_1
329: 99
330:SWAP
331:WS
332:PUSHB_1
333: 7
334:MINDEX
335:PUSHW_1
336: 448
337:MUL
338:PUSHB_1
339: 11
340:MINDEX
341:PUSHB_1
342: 1
343:SUB
344:PUSHB_1
345: 4
346:MINDEX
347:PUSHW_1
348: 320
349:MUL
350:PUSHB_1
351: 6
352:MINDEX
353:PUSHW_1
354: 64
355:MUL
356:PUSHB_1
357: 6
358:MINDEX
359:PUSHW_1
360: 582
361:EQ
362:PUSHB_1
363: 99
364:RS
365:AND
366:PUSHB_1
367: 99
368:SWAP
369:WS
370:PUSHB_1
371: 8
372:MINDEX
373:PUSHW_1
374: 20
375:EQ
376:PUSHB_1
377: 99
378:RS
379:AND
380:PUSHB_1
381: 99
382:SWAP
383:WS
```

```
384:PUSHB_1
385: 9
386:MINDEX
387:PUSHW_1
388: 2
389:EQ
390:PUSHB_1
391: 99
392:RS
393:AND
394:PUSHB_1
395: 99
396:SWAP
397:WS
398:PUSHW_1
399: 320
400:MUL
401:PUSHB_1
402: 6
403:MINDEX
404:PUSHB_1
405: 2
406:ADD
407:PUSHB_1
408: 7
409:MINDEX
410:PUSHB_1
411: 1
412:SUB
413:PUSHB_1
414: 6
415:MINDEX
416:PUSHW_1
417: 1463
418:EQ
419:PUSHB_1
420: 99
421:RS
422:AND
423:PUSHB_1
424: 99
425:SWAP
426:WS
427:PUSHB_1
428: 4
429:MINDEX
430:PUSHW_1
431: 1415
432:EQ
433:PUSHB_1
434: 99
435:RS
436:AND
437:PUSHB_1
438: 99
439:SWAP
440:WS
441:PUSHB_1
442: 5
443:MINDEX
444:PUSHB_1
445: 1
446:SUB
447:ROLL
448:PUSHW_1
449: 70
450:EQ
451:PUSHB_1
452: 99
453:RS
454:AND
```

```
455:PUSHB_1
456: 99
457:SWAP
458:WS
459:PUSHB_1
460: 3
461:SUB
462:SWAP
463:PUSHW_1
464: 256
465:MUL
466:PUSHB_1
467: 4
468:MINDEX
469:PUSHW_1
470: 32
471:EQ
472:PUSHB_1
473: 99
474:RS
475:AND
476:PUSHB_1
477: 99
478:SWAP
479:WS
480:PUSHB_1
481: 4
482:MINDEX
483:PUSHB_1
484: 2
485:SUB
486:PUSHW_1
487: 5
488:EQ
489:PUSHB_1
490: 99
491:RS
492:AND
493:PUSHB_1
494: 99
495:SWAP
496:WS
497:SWAP
498:PUSHB_1
499: 5
500:ADD
501:PUSHB_1
502: 6
503:SUB
504:ROLL
505:PUSHW_1
506: 90
507:EQ
508:PUSHB_1
509: 99
510:RS
511:AND
512:PUSHB_1
513: 99
514:SWAP
515:WS
516:PUSHW_1
517: 64
518:MUL
519:SWAP
520:PUSHB_1
521: 3
522:ADD
523:SWAP
524:PUSHW_1
525: 3
```

```
526:EQ
527:PUSHB_1
528: 99
529:RS
530:AND
531:PUSHB_1
532: 99
533:SWAP
534:WS
535:PUSHW_1
536: 256
537:MUL
538:PUSHB_1
539: 2
540:SUB
541:PUSHW_1
542: 320
543:MUL
544:PUSHW_1
545: 1970
546:EQ
547:PUSHB_1
548: 99
549:RS
550:AND
551:PUSHB_1
552: 99
553:SWAP
554:WS
555:PUSHB_1
556: 99
557:RS
558:IF
559:PUSHB_5
560: 0
561: 1
562: 2
563: 3
564: 4
565:PUSHB_5
566: 5
567: 6
568: 7
569: 8
570: 9
571:ELSE
572:PUSHB_5
573: 60
574: 61
575: 62
576: 63
577: 64
578:PUSHB_5
579: 65
580: 66
581: 67
582: 68
583: 69
584:EIF
585:SVTCA[y-axis]
586:PUSHB_1
587: 0
588:SCFS
589:PUSHB_1
590: 0
591:SCFS
592:PUSHB_1
593: 0
594:SCFS
595:PUSHB_1
596: 0
```

```
597:SCFS
598:PUSHB_1
599: 0
600:SCFS
601:PUSHB_1
602: 0
603:SCFS
604:PUSHB_1
605: 0
606:SCFS
607:PUSHB_1
608: 0
609:SCFS
610:PUSHB_1
611: 0
612:SCFS
613:PUSHB_1
614: 0
615:SCFS
```

4. TrueType ByteCode analyser

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

#define MAXSYM 1024
#define MAXINST 256
#define MAXSTACK 256
#define MAXSTORAGE 256
#define MAXCODE 2048

typedef  char TTsym_data[MAXSYM];

typedef  char TTInst[MAXINST];


typedef struct _stack
{
   int sp;
   TTsym_data selt[MAXSTACK];
} TTStack;

typedef struct _data_storage
{
   TTsym_data selt[MAXSTORAGE];
} TTStorage;

typedef struct _code
{
   int nb_inst;
   TTInst inst[MAXCODE];
} TTCode;

/*************************************/
int isNum(char *str)
{
   int lg;
   int i;

   lg = strlen(str);
   for (i=0; i<lg; i++)
      if  (str[i]!=' ' && (str[i]<'0' || str[i] >'9'))
         return(0);
```

```
return(1);
}
/*************************************/
int run_code(TTCode *code, TTStack *stack, TTStorage *storage)
{
   int i,j;
   char *pts;
   char *pts1;
   char *pts2;
   TTsym_data vals;
   int val;
   int val1, val2;
   TTsym_data tmpS;

   for (i=0; i<code->nb_inst; i++)
   {
      if ((strcmp(code->inst[i],"PUSHB_1")==0) || (strcmp(code->inst[i],"PUSHW_1")==0))
      {
         val = atoi(code->inst[i+1]);
         sprintf(vals,"%d",val);
         strcpy(stack->selt[stack->sp] , vals);
         stack->sp++;
         i++;
      }
      else if (strcmp(code->inst[i],"PUSHB_2")==0)
      {
         for (j=0; j<2; j++)
         {
            val = atoi(code->inst[i+1]);
            sprintf(vals,"%d",val);
            strcpy(stack->selt[stack->sp] , vals);
            stack->sp++;
            i++;
         }
      }
      else if (strcmp(code->inst[i],"PUSHB_5")==0)
      {
         for (j=0; j<5; j++)
         {
            val = atoi(code->inst[i+1]);
            sprintf(vals,"%d",val);
            strcpy(stack->selt[stack->sp] , vals);
            stack->sp++;
            i++;
         }
      }
      else if (strcmp(code->inst[i],"RS")==0)
      {
         pts = stack->selt[stack->sp-1];
         //printf("RS  : %d, %s \n",i, pts);
         //dump_stack(stack);
         if (isNum(pts))
         {
            val = atoi(pts);
            strcpy(stack->selt[stack->sp-1] , storage->selt[val]);
         }
         else
         {
            printf("RS with symbolic value \n");
            exit(-1);
         }

         //printf("RS  : %d, val=%d \n",i, val);
         //dump_stack(stack);
         //dump_storage(storage);
      }
      else if (strcmp(code->inst[i],"WS")==0)
      {
         pts = stack->selt[stack->sp-2];
         if (isNum(pts))
         {
```

```
      val = atoi(pts);
      strcpy(storage->selt[val], stack->selt[stack->sp-1]);
      stack->sp-=2;
    }
    else
    {
      dump_stack(stack);
      dump_storage(storage);
      printf("WS with symbolic value : %d, %s \n",i, pts);
      exit(-1);
    }
    //printf("WS  : I%d, val=%d \n",i, val);
    //dump_stack(stack);
    //dump_storage(storage);
  }
  else if (strcmp(code->inst[i],"SWAP")==0)
  {
    strcpy(tmpS, stack->selt[stack->sp-2]);
    strcpy(stack->selt[stack->sp-2], stack->selt[stack->sp-1]);
    strcpy(stack->selt[stack->sp-1], tmpS);
  }
  else if (strcmp(code->inst[i],"DUP")==0)
  {
    strcpy(tmpS, stack->selt[stack->sp-1]);
    strcpy(stack->selt[stack->sp], tmpS);
    stack->sp++;
  }
  else if (strcmp(code->inst[i],"ROLL")==0)
  {
    strcpy(tmpS, stack->selt[stack->sp-3]);
    strcpy(stack->selt[stack->sp-3], stack->selt[stack->sp-2]);
    strcpy(stack->selt[stack->sp-2], stack->selt[stack->sp-1]);
    strcpy(stack->selt[stack->sp-1], tmpS);
  }
  else if (strcmp(code->inst[i],"MINDEX")==0)
  {
    pts = stack->selt[stack->sp-1];
    if (isNum(pts))
    {
      val = atoi(pts);
      strcpy(tmpS, stack->selt[stack->sp-val-1]);
      for (j=val; j>0; j--)
      {
        strcpy(stack->selt[stack->sp-j-1], stack->selt[stack->sp-j]);
      }
      strcpy(stack->selt[stack->sp-2], tmpS);
      stack->sp--;
    }
    else
    {
      printf("MINDEX with symbolic value : %d, %s \n",i, pts);
      exit(-1);
    }
  }
  else if (strcmp(code->inst[i],"MUL")==0)
  {
    pts1 = stack->selt[stack->sp-2];
    pts2 = stack->selt[stack->sp-1];


    sprintf(tmpS, "(%s) * (%s)",pts1, pts2);
    stack->sp--;
    strcpy(stack->selt[stack->sp-1], tmpS);
  }
  else if (strcmp(code->inst[i],"ADD")==0)
  {
    pts1 = stack->selt[stack->sp-2];
    pts2 = stack->selt[stack->sp-1];

    if (isNum(pts1) && isNum(pts2))
    {
```

```
            val1 = atoi(pts1);
            val2 = atoi(pts2);
            sprintf(tmpS, "%d",val1 + val2);
        }
        else
          sprintf(tmpS, "(%s) + (%s)",pts1, pts2);

        stack->sp--;
        strcpy(stack->selt[stack->sp-1], tmpS);
    }
    else if (strcmp(code->inst[i],"SUB")==0)
    {
        pts1 = stack->selt[stack->sp-2];
        pts2 = stack->selt[stack->sp-1];

        sprintf(tmpS, "(%s) - (%s)",pts1, pts2);
        stack->sp--;
        strcpy(stack->selt[stack->sp-1], tmpS);
    }
    else if (strcmp(code->inst[i],"AND")==0)
    {
        pts1 = stack->selt[stack->sp-2];
        pts2 = stack->selt[stack->sp-1];

        sprintf(tmpS, "(%s) & (%s)",pts1, pts2);
        stack->sp--;
        strcpy(stack->selt[stack->sp-1], tmpS);
    }
    else if (strcmp(code->inst[i],"EQ")==0)
    {
        pts1 = stack->selt[stack->sp-2];
        pts2 = stack->selt[stack->sp-1];

        printf("Equality test: Ist=%d %s == %s \n", i, pts1, pts2);
        sprintf(tmpS, "(%s) == (%s)",pts1, pts2);
        stack->sp-=2;
        strcpy(stack->selt[stack->sp], tmpS);
        stack->sp++;

    }
    else if (strcmp(code->inst[i],"IF")==0)
    {
        pts1 = stack->selt[stack->sp-1];
        printf("IF test: Ist=%d %s == %s \n", i, pts1);
    }
    else

    {
        printf("UNKNOW Instruction: %s \n", code->inst[i]);
        exit(-1);
    };

  }

}
/*************************************/
int dump_code(TTCode *code)
{
  int i;

  for (i=0; i<code->nb_inst; i++)
  {
     printf("%d:%s\n",i,code->inst[i]);
  }

  printf("\n");
  return(0);
}
/*************************************/
int dump_stack(TTStack *stack)
{
```

```
   int i;

   for (i=0; i<stack->sp; i++)
   {
      printf("Stck %d:%s\n",i,stack->selt[i]);
   }

   printf("\n");
   return(0);
}
/*************************************/
int dump_storage(TTStorage *storage)
{
   int i;

   for (i=0; i<100; i++)
   {
      if (strcmp (storage->selt[i],"")!=0)
         printf("Stor %d:%s\n",i,storage->selt[i]);
   }

   printf("\n");
   return(0);
}
/*************************************/
int init_storage(TTStorage *storage)
{
   int i;

   for (i=0; i<100; i++)
   {
     sprintf(storage->selt[i],"R%d",i);
   }

   //sprintf(storage->selt[99],"0");
   sprintf(storage->selt[99],"22");

   printf("\n");
   return(0);
}
/*************************************/
int load_code(char *filename, TTCode *code)
{
FILE *fch;
int nb_inst;
char *inst;
TTsym_data tmpInst;
int lg;

fch = fopen(filename, "r");
if (fch == NULL)
   return(-1);

nb_inst = 0;


do {
   inst = fgets(tmpInst, MAXINST, fch);
   if (inst != NULL)
   {
      lg = strlen(tmpInst);
      tmpInst[lg-2]=0; // remove CR,LF
      strcpy(code->inst[nb_inst], tmpInst);
      nb_inst++;
   }
}
while (inst != NULL);

code->nb_inst = nb_inst;
```

```
fclose(fch);

return(0);
}
/*************************************/
int main(int argc, char *argv[])
{
TTCode zeCode;
TTStack zeStack;
TTStorage zeStorage;

if (argc<2)
   return(-1);

memset(&zeCode, 0, sizeof(zeCode));
memset(&zeStack, 0, sizeof(zeStack));
memset(&zeStorage, 0, sizeof(zeStorage));

load_code(argv[1], &zeCode);
dump_code(&zeCode);

init_storage(&zeStorage);

run_code(&zeCode, &zeStack, &zeStorage);
//run_code(&zeCode, &zeStack, &zeStorage);
dump_stack(&zeStack);
dump_storage(&zeStorage);

return(0);
}
```

5. Img_extractData

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

#define SCT_SIZE 512

long part_of7[]={   3,        2097154,
                 2099201,    3147775,
                 3147777,    3168256,
                 3168257,    3987455,
                15634432,    15634432};

long firstSect = 0x200;
long lastSect = 0x79B98C00;

/***************************************/
int writeSect(long sectNum, FILE* fin, FILE *fout)
{
   unsigned char buff[SCT_SIZE];


   printf("Sect: %d\n",sectNum);

   fseek(fin, sectNum*SCT_SIZE, SEEK_SET);
   fread(buff, 1, SCT_SIZE, fin);
   fwrite(buff, 1, SCT_SIZE, fout);
}
/***************************************/
int extractData(FILE *fin, FILE * fout)
{
   long sectNum;
   long lastSectNum ;
   int prt=0;

   lastSectNum = lastSect / SCT_SIZE;
   sectNum = firstSect / SCT_SIZE;

   while ((sectNum <= lastSectNum) && (prt <=4))
   {

      if (sectNum == part_of7[2*prt])
      {
         sectNum = part_of7[2*prt+1] +1;
         prt++;
      }
      else
      {
         writeSect(sectNum, fin ,fout);
         sectNum++;
      }

   }
   printf("%d,%d\n",sectNum, prt);

   return(0);
}
/***************************************/
int main(int argc, char *argv[])
{
   FILE *fchin;
   FILE *fchout;

   if (argc<2)
      return(-1);

   fchin = fopen(argv[1],"rb");
   if (fchin ==NULL)
```

```
    return(-1);

  fchout = fopen("data.bin","wb");
  if (fchout ==NULL)
    return(-1);


  extractData(fchin, fchout);

  fclose(fchout);

  fclose(fchin);

  return(0);
}
```

6. decryptDataFS

```
/* This is an independent implementation of the encryption algorithm:   */
/*                                                                      */
/*       RC6 by Ron Rivest and RSA Labs                                 */
/*                                                                      */
/* which is a candidate algorithm in the Advanced Encryption Standard   */
/* programme of the US National Institute of Standards and Technology.  */
/*                                                                      */
/* Copyright in this implementation is held by Dr B R Gladman but I     */
/* hereby give permission for its free direct or derivative use subject */
/* to acknowledgment of its origin and compliance with any conditions   */
/* that the originators of the algorithm place on its exploitation.     */
/*                                                                      */
/* Dr Brian Gladman (gladman@seven77.demon.co.uk) 14th January 1999     */



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

#include "std_defs.h"



static char *alg_name[] = { "rc6", "rc6.c", "rc6" };


char **cipher_name()

{

  return alg_name;

}



#define f_rnd(i,a,b,c,d)                \
    u = rotr(d * (d + d + 1), 5);       \
    t = rotr(b * (b + b + 1), 5);       \
    a = rotr(a ^ t, u) + l_key[i];      \
    c = rotr(c ^ u, t) + l_key[i + 1]



#define i_rnd(i,a,b,c,d)                \
    u = rotr(d * (d + d + 1), 5);       \
```

```
    t = rotr(b * (b + b + 1), 5);      \
    c = rotl(c - l_key[i + 1], t) ^ u;  \
    a = rotl(a - l_key[i], u) ^ t



u4byte  l_key[44];  /* storage for the key schedule        */



/* initialise the key schedule from the user supplied key   */



u4byte *set_key(const u4byte in_key[], const u4byte key_len)

{
   u4byte  i, j, k, a, b, l[8], t;


   l_key[0] = 0xb7e15163;

   for(k = 1; k < 44; ++k)
      l_key[k] = l_key[k - 1] + 0x9e3779b9;


   for(k = 0; k < key_len / 32; ++k)
      l[k] = in_key[k];



   t = (key_len / 32) - 1; // t = (key_len / 32);



   a = b = i = j = 0;


   for (k = 0; k < 132; ++k)
   {   a = rotr(l_key[i] + a + b, 3); b += a;


       b = rotr(l[j] + b, b);


       l_key[i] = a; l[j] = b;

       i = (i == 43 ? 0 : i + 1); // i = (i + 1) % 44;

       j = (j == t ? 0 : j + 1);  // j = (j + 1) % t;

   }



   return l_key;

};



/* encrypt a block of text  */



void encrypt(const u4byte in_blk[4], u4byte out_blk[4])

{   u4byte  a,b,c,d,t,u;
```

```
    a = in_blk[0]; b = in_blk[1] + l_key[0];
    c = in_blk[2]; d = in_blk[3] + l_key[1];

    f_rnd( 2,a,b,c,d); f_rnd( 4,b,c,d,a);
    f_rnd( 6,c,d,a,b); f_rnd( 8,d,a,b,c);
    f_rnd(10,a,b,c,d); f_rnd(12,b,c,d,a);
    f_rnd(14,c,d,a,b); f_rnd(16,d,a,b,c);
    f_rnd(18,a,b,c,d); f_rnd(20,b,c,d,a);
    f_rnd(22,c,d,a,b); f_rnd(24,d,a,b,c);
    f_rnd(26,a,b,c,d); f_rnd(28,b,c,d,a);
    f_rnd(30,c,d,a,b); f_rnd(32,d,a,b,c);
    f_rnd(34,a,b,c,d); f_rnd(36,b,c,d,a);
    f_rnd(38,c,d,a,b); f_rnd(40,d,a,b,c);

    out_blk[0] = a + l_key[42]; out_blk[1] = b;
    out_blk[2] = c + l_key[43]; out_blk[3] = d;
};

/* decrypt a block of text  */

void decrypt(const u4byte in_blk[4], u4byte out_blk[4])
{   u4byte  a,b,c,d,t,u;

    d = in_blk[3]; c = in_blk[2] - l_key[43];
    b = in_blk[1]; a = in_blk[0] - l_key[42];

    i_rnd(40,d,a,b,c); i_rnd(38,c,d,a,b);
    i_rnd(36,b,c,d,a); i_rnd(34,a,b,c,d);
    i_rnd(32,d,a,b,c); i_rnd(30,c,d,a,b);
    i_rnd(28,b,c,d,a); i_rnd(26,a,b,c,d);
    i_rnd(24,d,a,b,c); i_rnd(22,c,d,a,b);
    i_rnd(20,b,c,d,a); i_rnd(18,a,b,c,d);
    i_rnd(16,d,a,b,c); i_rnd(14,c,d,a,b);
    i_rnd(12,b,c,d,a); i_rnd(10,a,b,c,d);
    i_rnd( 8,d,a,b,c); i_rnd( 6,c,d,a,b);
    i_rnd( 4,b,c,d,a); i_rnd( 2,a,b,c,d);
```

```
   out_blk[3] = d - l_key[1]; out_blk[2] = c;

   out_blk[1] = b - l_key[0]; out_blk[0] = a;

}


/*****************************************************/
int fch_decrypt()
{
   FILE *fch;

   FILE *fchout;

   unsigned char pbk[16];
   unsigned char cbk[16];
   unsigned char dcbk[16];
   int ret;
   int i;

   memset(pbk, 0, 16);
   memset(cbk, 0, 16);

   fch = fopen("data.bin","rb");
   if (fch == NULL)
      return(-1);

   fchout = fopen("dataClear.bin","wb");
   if (fchout ==NULL)
      return(-1);

   fseek(fch, 0x40, SEEK_SET);
   do
   {
      ret = fread(cbk,1, 16, fch);
      if (ret ==16)
      {
         decrypt(cbk, dcbk);
         for (i=0; i<16; i++)
         {
            dcbk[i] = dcbk[i] ^ pbk[i];
         }
         memcpy(pbk, cbk, 16);

         fwrite(dcbk, 1, 16, fchout);
      }
   }
   while (ret == 16);


   fclose(fchout);
   fclose(fch);

   return(0);
}
/*****************************************************/
int main()
{

   unsigned char in_key[16];
   int key_len=128;
   char v5[] = "551C2016B00B5F00";
   unsigned int *ptr;
   unsigned int val=0;
   int i;


   memset(in_key, 0, 16);
```

```
  ptr = in_key;
  for (i=0; i<16; i++)
  {
     val += v5[i];
     if (((i+1)&3)==0)
     {
        *ptr++ = val;
         val=0;
     }
     else
        val<<=8;
  }


  set_key(in_key, key_len);

  fch_decrypt();

}
```

7. decryptFiles

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



// Key Scheduling Algorithm
// Input: state - the state used to generate the keystream
//       key - Key to use to initialize the state
//       len - length of key in bytes
void ksa(unsigned char state[], unsigned char key[], int len)
{
  int i,j=0,t;

  for (i=0; i < 256; ++i)
    state[i] = i;
  for (i=0; i < 256; ++i) {
    j = (j + state[i] + key[i % len]) % 256;
    t = state[i];
    state[i] = state[j];
    state[j] = t;
  }
}

// Pseudo-Random Generator Algorithm
// Input: state - the state used to generate the keystream
//       out - Must be of at least "len" length
//       len - number of bytes to generate
void prga(unsigned char state[], unsigned char out[], int len)
{
  int i=0,j=0,x,t;
  unsigned char key;

  for (x=0; x < len; ++x)  {
    i = (i + 1) % 256;
    j = (j + state[i]) % 256;
    t = state[i];
    state[i] = state[j];
    state[j] = t;
    out[x] = state[(state[i] + state[j]) % 256];
  }
}
/*****************************************************************************/
void decryptRC4(unsigned char data[], int len, unsigned char state[])
{
   int i=0,j=0,x,t;
```

```
  unsigned char key;

  for (x=0; x < len; ++x)  {
    i = (i + 1) % 256;
    j = (j + state[i]) % 256;
    t = state[i];
    state[i] = state[j];
    state[j] = t;
    data[x] = data[x] ^ ( state[(state[i] + state[j]) % 256] );
   }
}
/*************************************************************************/
int decryptFile(unsigned char data[], int lg, unsigned char *key)
{
   unsigned char rc4State[256];
   FILE *fch;
   char filename[16];
   static int fctr=0;

   ksa(rc4State,  key, 16);

   decryptRC4(data, lg, rc4State);

   sprintf(filename,"fch%d.bin",fctr++);
   fch = fopen(filename,"wb");
   if (fch ==NULL)
      return(-1);

   fwrite(data, 1, lg, fch);


   fclose(fch);

   return(0);
}
/*************************************************************************/
int extractFiles(FILE *fchin)
{
   unsigned char header [36];
   int ret;
   int flg;
   int ltot=0;
   unsigned char krc4[16];
   unsigned char md5[16];

   static unsigned char dataBuff[2*1024*1024];

   do
   {
      ret = fread(header,1, 36, fchin);
      if (ret ==36)
      {
         flg = header[3];
         flg<<=8;
         flg += header[2];
         flg<<=8;
         flg += header[1];
         flg<<=8;
         flg += header[0];
         memcpy(krc4,header+4,16);
         memcpy(md5,header+20,16);

         ltot+= flg;
         printf("File Length=%d\n",flg);

         if (flg <= 2*1024*1024)
         {

            fread(dataBuff, 1, flg, fchin);
            decryptFile(dataBuff, flg, krc4);
         }
```

```
        //fseek(fchin, flg, SEEK_CUR);



      }
   }
   while (ret >0);

   printf("total:%d\n",ltot);

 return(0);
}
/***************************************/
int main(int argc, char *argv[])
{
   FILE *fchin;


   if (argc<2)
      return(-1);

   fchin = fopen(argv[1],"rb");
   if (fchin ==NULL)
      return(-1);




   extractFiles(fchin);



   fclose(fchin);

   return(0);
}
```

8. Color decoder

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


static unsigned char colorLut[]= {0xB3, 0x8E, 0x48, 0x4F, 0xB3, 0x71, 0xB7, 0x4F,  0xB3, 0x71, 0x48, 0x4F, 0x4C, 0x8E, 0xB7, 0x4F,
0x4C, 0x8E, 0x48, 0x4F, 0x4C, 0x71, 0xB7, 0x4F,  0x4C, 0x71, 0x48, 0x4F, 0xB3, 0x8E, 0xB7, 0x4F};


static unsigned char xorKey[]=  {0x30, 0xA4, 0x3F, 0x6D, 0x28, 0x04, 0x23, 0x36,  0x2A, 0x32, 0xDC, 0xAD, 0x0B, 0xA0, 0x4B, 0xE8,
0x20, 0x1F, 0x64, 0x84, 0x0A, 0xF4, 0xC4, 0xC7,  0x8A, 0x8D, 0xC0, 0xA2, 0xC4, 0x40, 0x19, 0xA1};

static unsigned char xorKey2[]=  {0x28, 0x30, 0xA4, 0x3F, 0x6D, 0x28, 0x04, 0x23, 0x36,  0x2A, 0x32, 0xDC, 0xAD, 0x0B, 0xA0, 0x4B, 0xE8,
0x20, 0x1F, 0x64, 0x84, 0x0A, 0xF4, 0xC4, 0xC7,  0x8A, 0x8D, 0xC0, 0xA2, 0xC4, 0x40, 0x19, 0xA1};



static int pow7[3]={1, 7, 49};
```

```
/****************************************/
int decrypt_Clut()
{
   int i;
   unsigned int *lptr;

   lptr = (unsigned int *) colorLut;
   for (i=0; i<8; i++)
   {
      *lptr ^= 0x4FB78EB3;

      lptr++;
   }

}
/****************************************/
int encode(unsigned char *in, int inLen, unsigned long *colors)
{
   int i,j;
   unsigned char val;
   unsigned char rem;
   unsigned char idx;
   int nbCol=0;

   unsigned int *lptr;
   lptr = (unsigned int *) colorLut;

   idx = 6;
   for (i=0; i<inLen; i++)
   {
      val = in[i] ^ xorKey[i];


      for (j=0; j<3; j++)
      {
         rem = (val%7) +1;

         idx += rem;
         idx &= 0x07;


         *colors++ = lptr[idx];
         nbCol++;

         val = val /7;
      }

   }


   return(nbCol);
}
/****************************************************/
int findClut(unsigned long color)
{
   int i;

   unsigned int *lptr;

   lptr = (unsigned int *) colorLut;
   for (i=0; i<8; i++)
   {
      if (*lptr == color)
        return(i);
      lptr++;
   }

   return(-1);

}
```

```
/*************************************************/
int decode(unsigned long *colors, int nbCol, unsigned char *out)
{
   int i,j;
   int cptr;
   int iclt;
   int idx;
   int rem;

   unsigned char val;

   val = 0;
   idx = 6;

   cptr=0;
   j=0;
   for (i=0; i<nbCol; i++)
   {
      iclt = findClut(colors[i]);
      if (iclt < 0)
         return(-1);

      rem = (iclt - idx) ;
      if (rem < 0)
         rem +=8;


      idx = iclt;
      rem = rem - 1;
      if (rem < 0)
         rem +=7;


      val = val + rem * pow7[j];
      j++;
      if (j == 3)
      {

         *out++ = val ^ xorKey2[cptr];
         val=0;
         cptr ++;
         j=0;
      }

   }

   return(cptr);
}
/*************************************************/
int dump(unsigned char *buff, int size)
{
  int i;

  for (i=0; i<size; i++)
  {
    if ((i % 16  ) == 0)
                printf("\n");

    printf("%02X ",buff[i]);
  }

 printf("\n");
}
/***************************************/
int dumpColors(unsigned long *colors, int lg)
{
   int i;

   printf("\n");
   for (i=0; i<lg; i++)
   {
```

```
      printf("%2d: 0x%8X\n",i,colors[i]);
   }
}
/***************************************/
int main(int argc, char *argv[])
{

   unsigned long colors[90];
   int ret;
   unsigned char out[27];
   int i;

#define CYAN    0x00FFFF
#define MAGENTA 0xFF00FF
#define YELLOW  0xFFFF00
#define BLUE    0x0000FF
#define RED     0xFF0000
#define GREEN   0x00FF00
#define WHITE   0xFFFFFF
#define BLACK   0x000000



   unsigned long mcolors2[]={CYAN, MAGENTA, CYAN, MAGENTA, BLUE, MAGENTA, RED, BLUE, BLACK, RED,  BLUE,  CYAN, WHITE, YELLOW,
BLUE, RED, BLACK, MAGENTA, WHITE, GREEN,
                        YELLOW,BLACK,RED,YELLOW,CYAN
,WHITE,BLACK,CYAN,BLACK,GREEN,WHITE,YELLOW,BLUE,YELLOW,BLUE,MAGENTA,BLUE,MAGENTA,BLACK,WHITE,

BLUE,CYAN,BLUE,GREEN,BLUE,WHITE,BLUE,WHITE,RED,WHITE,GREEN,CYAN,RED,YELLOW,CYAN,GREEN,BLUE,RED,GREEN,CYAN,

RED,YELLOW,MAGENTA,WHITE,GREEN,MAGENTA,CYAN,YELLOW,WHITE,BLACK,RED,CYAN,WHITE,BLACK,CYAN,BLACK,BLUE,RED,WHITE,M
AGENTA,
                        BLACK,GREEN,RED,MAGENTA};



   decrypt_Clut();



   ret = decode( mcolors2,  84, out);
   dump(out, ret);

}
```