Challenge SSTIC 2010

Une solution proposée par Nicolas RUFF

Contexte

Plateforme

La solution proposée ci-dessous a été entièrement réalisée sur un système Windows Seven (x64).

Les commandes Unix sont tirées de la suite Cygwin 1.7[1].

Le logiciel IDA Pro 5.6 a été utilisé[2]. Il faut signaler que la version d'évaluation gratuite[3] peut être utilisée pour résoudre le Challenge, car elle inclut le support Linux/ARM (analyse statique et débogage).

Description du Challenge

Le Challenge SSTIC 2010 est publié avec la description suivante :

Le défi consiste à analyser la copie intégrale de la mémoire physique d'un mobile utilisant le système d'exploitation Android.

L'objectif est d'y retrouver une adresse e-mail (...@sstic.org).

Le contenu de la mémoire physique est disponible ici : http://www.sstic.org/media/SSTIC2010/concours_sstic_2010

Empreinte SHA-256 : 78c9ab57cdb8ba1eb7fde9a1d165fe86a6789320b63fb079f6ad8cd8dbebe037 concours_sstic_2010

Après avoir téléchargé le fichier proposé et vérifié l'empreinte SHA-256, la commande `file` permet d'identifier la cible. Espérons que personne ne soit resté bloqué à cette étape ☺.

```
C:\> file concours_sstic_2010
concours_sstic_2010: 7-zip archive data, version 0.3
```

Après décompression du fichier, une première vérification s'impose :

```
C:\> strings challv2 | grep -i sstic.org
```

Aucune réponse : il va falloir travailler un peu plus …

[1] http://www.cygwin.com/
[2] http://www.hex-rays.com/
[3] http://hex-rays.com/idapro/idadowndemo.htm

Mise en place de l’environnement Android

Description du système Android

Android est un système d’exploitation proposé par Google, principalement destiné aux terminaux mobiles (en attendant Google Chrome OS). Il est basé sur Linux, et disponible en Open Source. Il faut signaler que la documentation du projet est d’excellente qualité.

Google encourage l’utilisation du langage Java pour le développement d’applications Android, afin d’assurer une portabilité maximale entre équipements. La JVM intégrée à Android est nommée Dalvik. Elle n’est pas iso-fonctionnelle avec la JVM proposée par Sun (y compris la version mobile MIDP / CLDC). Les différences étant largement documentées sur Internet, il est inutile de s’attarder dessus. Le format « JAR » est remplacé par le format « APK » (qui reste un fichier ZIP), et le format « CLASS » est remplacé par le format « DEX ».

Les points d’entrée à connaitre sont les suivants.

SDK

Le SDK (kit de développement d’applications Java) est librement téléchargeable[4].

Il est proposé en version Windows, Linux (x86) et Mac OS X (x86). Ces 3 versions disposent du même niveau de support, ce qui est appréciable. L’utilisation de l’environnement de développement Eclipse est fortement recommandée, car le SDK s’intègre naturellement dans cet environnement (si disponible).

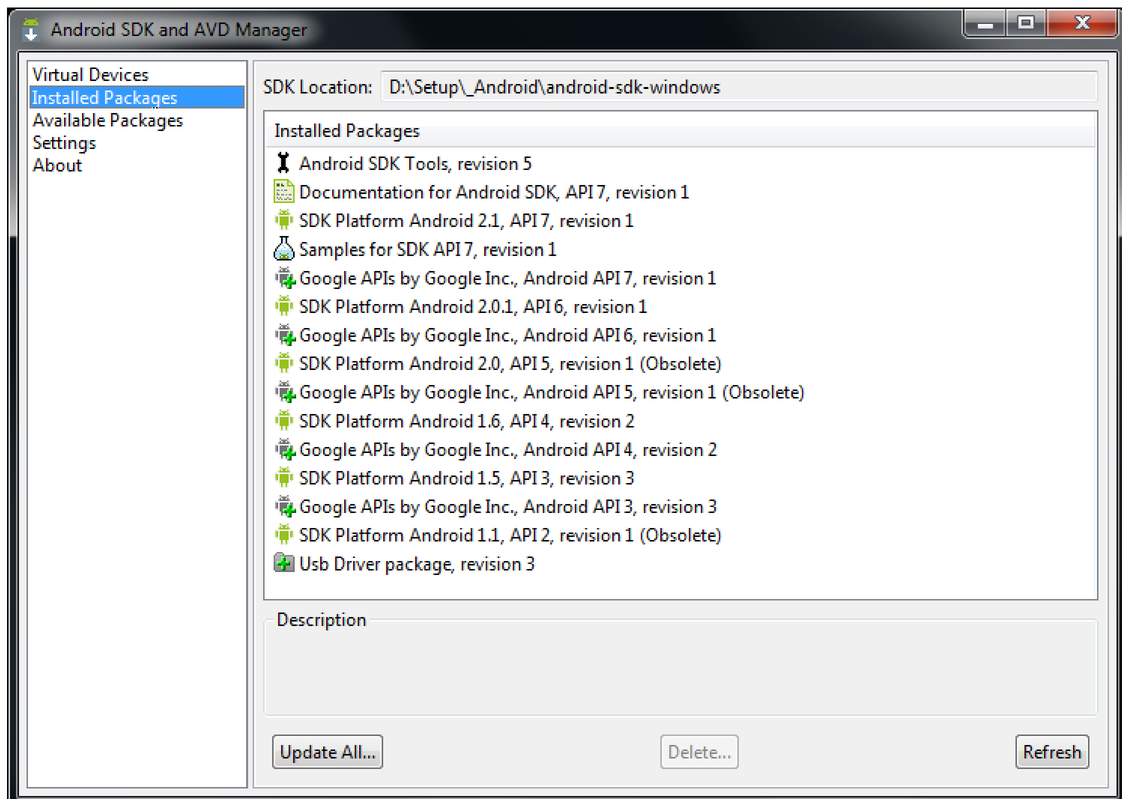
L’installation sous Windows s’effectue avec la commande suivante :

```
./SDK Setup.exe
```

L’installation sous Linux s’effectue avec la commande :

```
./tools/android
```

[4] http://developer.android.com/sdk/index.html



GUI d'installation du SDK sous Windows

Il faut noter qu'à la date de rédaction de ce document, l'utilisation de HTTPS comme protocole de téléchargement des packages ne fonctionne pas (contrairement à ce qu'indique la documentation en ligne). Il est nécessaire de forcer l'utilisation de HTTP.

Après installation, il est recommandé de définir la variable d'environnement `SDK_ROOT` à la racine d'installation, et d'ajouter le répertoire `SDK_ROOT/tools` dans le `PATH`.

Le SDK comprend plusieurs outils d'importance. Les trois outils indispensables pour la suite sont :

- adb.exe
- emulator.exe
- ddms.bat

NDK

Le NDK (kit de développement d'applications natives) est librement téléchargeable[5].

Il est également proposé pour Windows, Linux et Mac OS X. Il s'agit essentiellement d'un environnement de compilation croisée pour Linux/ARM, basé sur GCC 4.2.1 ou 4.4.0 (dans la version 3 du NDK). GCC est livré précompilé pour toutes les plateformes supportées dans le répertoire `./build/prebuilt`.

[5] http://developer.android.com/sdk/ndk/index.html

Ce kit de développement n'est pas destiné à compiler des applications 100% natives, mais plutôt destiné à compiler des librairies « critiques » qui devront être appelées depuis une application « Dalvik » par le biais de l'API Java Native Interface (JNI).

Noyau

Le code du noyau (et tout l'environnement d'exécution afférent, dont la machine Dalvik) est disponible le site officiel « kernel.org »[6].

Dans le cadre du Challenge, seule la consultation des sources est nécessaire. Il a toutefois été constaté qu'une image complète, amorçable dans l'émulateur, pouvait être construite à partir des sources du noyau.

A titre de référence, le système de fichiers utilisé par défaut est YAFFS2[7].

Utilisation dans le contexte du Challenge

Point(s) de départ

La recherche de chaines de caractères est toujours une bonne idée pour commencer.

Tentons d'identifier la version :

```
C:\> strings challv2 | grep –i gcc

Linux version 2.6.29-00255-g7ca5167 (digit@digit.mtv.corp.google.com) (gcc
version 4.4.0 (GCC) ) #9 Tue Dec 1 16:12:35 PST 2009

(…)
```

Il s'agit donc de la dernière version d'Android à la date de rédaction de ce document (2.1).

Nous savons également que le concours est organisé par l'ANSSI :

```
C:\> strings challv2 | grep –i anssi | sort | uniq

(…)

/data/dalvik-cache/data@app@com.anssi.secret.apk@classes.dex

/data/dalvik-cache/data@app@com.anssi.textviewer.apk@classes.dex

<package name="com.anssi.secret" codePath="/data/app/com.anssi.secret.apk"
system="false" ts="1270823989000" version="1" userId="10025">

<package name="com.anssi.textviewer"
codePath="/data/app/com.anssi.textviewer.apk" system="false"
ts="1270823961000" version="1" userId="10024">

Displayed activity com.anssi.secret/.SecretJNI: 1233 ms (total 1233 ms)

Starting com.anssi.secret
```

[6] http://android.git.kernel.org/

[7] http://en.wikipedia.org/wiki/YAFFS#YAFFS2

```
Starting com.anssi.textviewer

Trying    to    load    lib    /data/data/com.anssi.secret/lib/libhello-jni.so
0x43d017f8

(…)
```

Nous découvrons que deux applications potentiellement intéressantes ont été exécutées : `com.anssi.textviewer.apk` et `com.anssi.secret.apk`, cette dernière faisant appel à `libhello-jni.so`.

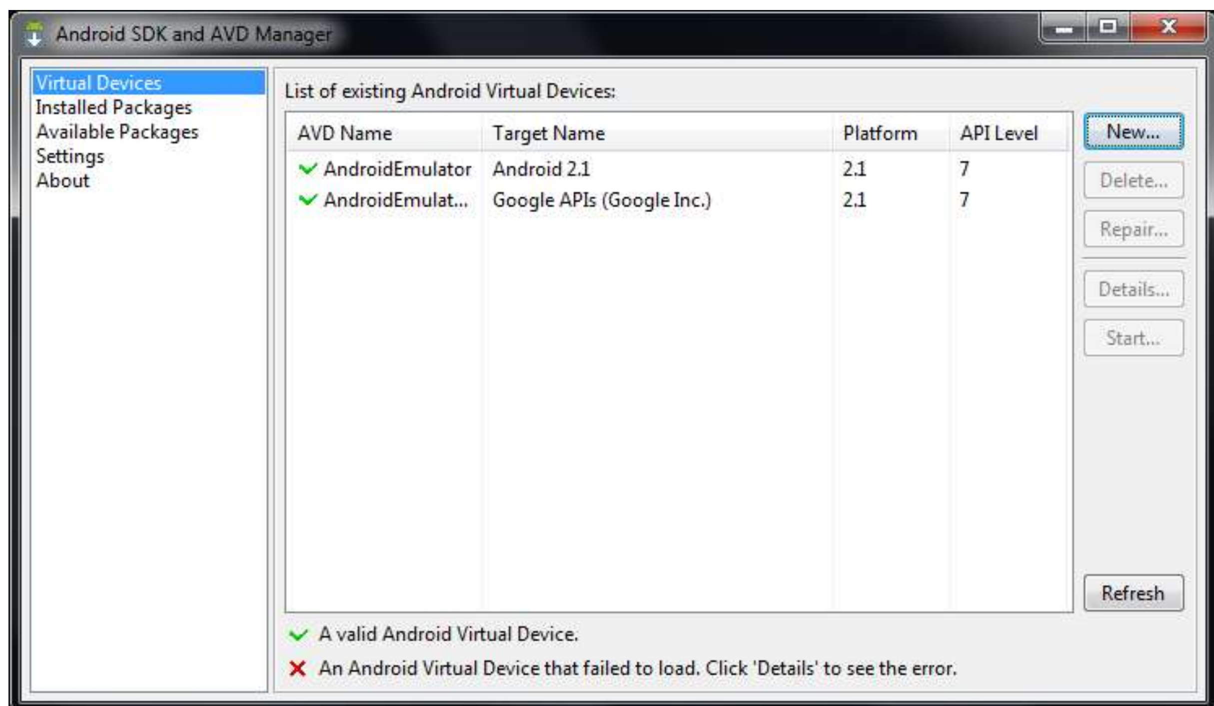
Une idée est alors récupérer en mémoire le contenu des fichiers « APK ».

Analyse mémoire

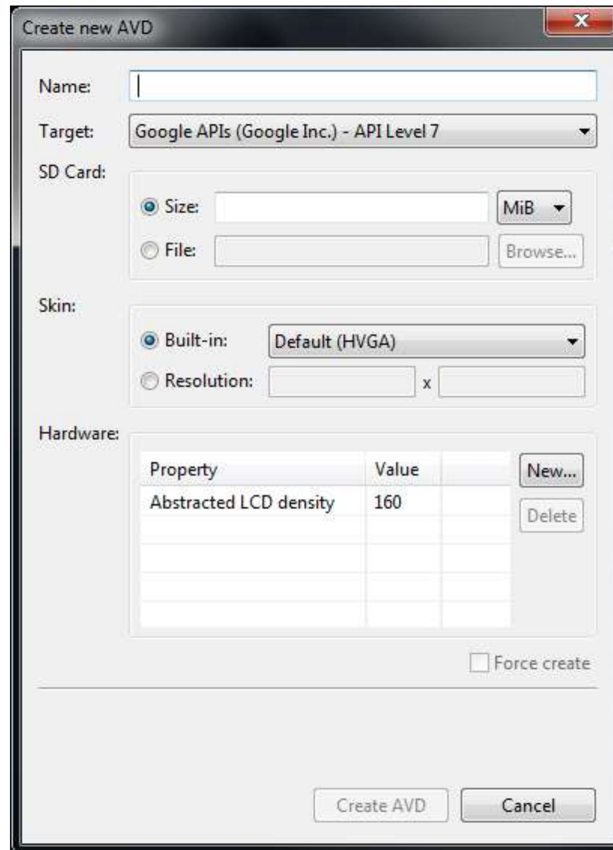
A la lecture des chaines suivantes, on peut supposer que l'image mémoire a été extraite de l'émulateur et non d'un téléphone « physique ».

```
C:\> strings challv2 | grep -i emulator

# adbd on at boot in emulator
Registering null Bluetooth Service (emulator)
it runs on the emulator
(…)
```

L'émulateur Android est construit autour de QEmu. Il est relativement facile de créer un nouveau système émulé, car des modèles par défaut sont proposés.



Liste des systèmes émulés (vide par défaut)



Ajout d'un nouveau système émulé

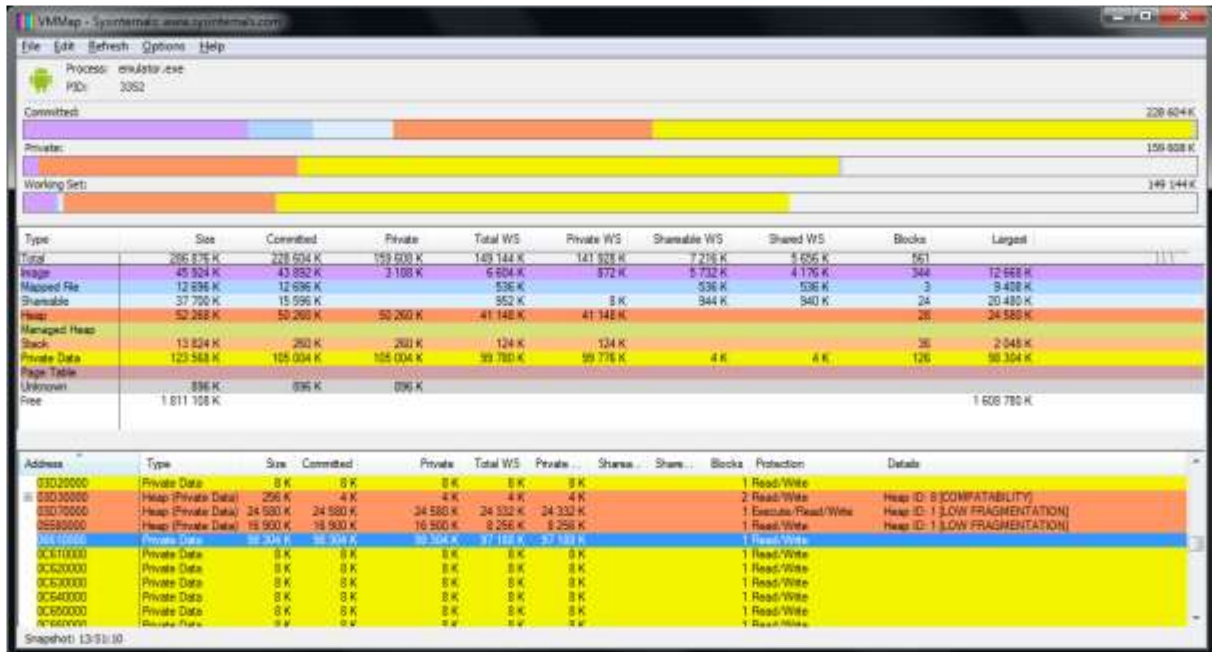
La configuration de l'émulateur et les systèmes émulés sont stockés dans `%UserProfile%/android` (`~/.android` sous Linux). Chaque système doit se voir assigner un nom unique.

L'émulateur est ensuite démarré avec la ligne de commande suivante :

```
emulator.exe -avd <nom du système émulé>
```

On identifie alors (avec l'outil VMMap[8] de SysInternals) dans la mémoire de l'émulateur une zone de taille identique à celle du Challenge, et contenant des chaines de caractères très similaires.

[8] http://technet.microsoft.com/en-us/sysinternals/dd535533.aspx



Zone de 98 Mo identifiée dans la mémoire de l'émulateur

On peut supposer que le Challenge a été conçu en sauvegardant directement cette zone mémoire dans un fichier.

La première idée qui vient est celle de réinjecter l'image mémoire du Challenge dans l'émulateur. Malheureusement, cette idée risque de ne pas fonctionner si le contexte d'exécution (i.e. l'état du CPU émulé) n'est pas disponible. Sans avoir regardé les sources de QEmu, on peut malheureusement supposer que de nombreuses informations d'état sont sauvegardées ailleurs que dans cette zone mémoire.

Reconstruction de la mémoire virtuelle

Nous savons que 2 processus « Dalvik » d'intérêt pour le Challenge existent (ou ont existé) dans la mémoire du système.

Une idée qui vient alors est de reconstruire la mémoire virtuelle des processus exécutés, afin de récupérer le contenu binaire pour analyse.

Toutefois cette idée n'est pas forcément la meilleure, au moins pour les raisons suivantes :

- Elle nécessite de lire et de comprendre le code source du noyau Android, afin de comprendre la gestion de la mémoire et des processus.
- Si les processus sont terminés, seuls des artefacts seront récupérés, potentiellement incomplets.
- Les fichiers « APK » ne sont pas exécutés « tels quels », mais le fichier `classes.dex` est extrait et interprété par la machine « Dalvik ». Les données récupérées en mémoire risquent d'être difficile à réutiliser si on ne dispose pas du fichier « APK » d'origine.
- Si les fichiers « APK » existent toujours en mémoire, ce sera probablement de manière indirecte, par exemple dans une zone mémoire temporaire allouée pour une copie ou une décompression.

La reconstruction mémoire, si elle probablement la solution la plus élégante techniquement, présente donc les inconvénients d'être longue à développer et pas forcément exhaustive (par exemple les zones mémoires récemment désallouées peuvent être marquées comme « libre »).

C'est pourquoi une autre technique a été explorée : celle du *ZIP carving*.

ZIP carving

Principe(s)

Le *ZIP carving* consiste à reconstruire les fichiers APK (qui sont au format ZIP) à partir des pages de mémoire physique.

Cette technique est avantageuse pour plusieurs raisons :

- Le format ZIP présente une structure régulière, documentée, et redondante.
- Plusieurs outils existent déjà pour reconstruire des fichiers ZIP à partir d'une image.
- La recherche de données s'effectue au niveau des pages mémoire, sans préjuger de l'utilisation qui en est faite par le système d'exploitation. Elle est donc exhaustive et ne nécessite pas de comprendre l'intégralité du fonctionnement du système Android.

En ce qui concerne la structure du format ZIP, elle est fort bien documentée sur Wikipedia[9] (qui s'avère par ailleurs être une bibliothèque technique de grande valeur).

Chaque fichier est un flux compressé préfixé par un *Local File Header* (LFH), dont la structure est la suivante.

Offset	***Bytes***	***Description***
0	4	*Local file header signature = « PK\x03\x04 »*
4	2	*Version needed to extract (minimum)*
6	2	*General purpose bit flag*
8	2	*Compression method*
10	2	*File last modification time*
12	2	*File last modification date*
14	4	*CRC-32*
18	4	*Compressed size*
22	4	*Uncompressed size*
26	2	*File name length (n)*
28	2	*Extra field length (m)*
30	n	*File name*
30+n	m	*Extra field*

Structure d'un LFH

A la fin de l'archive se trouve un ensemble de *Central Directory Headers* (CDH), concaténés les uns à la suite des autres, qui pointent vers tous les LFH précédents.

Offset	***Bytes***	***Description***
0	4	*Central directory file header signature = « PK\x01\x02 »*

[9] http://en.wikipedia.org/wiki/ZIP_(file_format)

4	2	*Version made by*
6	2	*Version needed to extract (minimum)*
8	2	*General purpose bit flag*
10	2	*Compression method*
12	2	*File last modification time*
14	2	*File last modification date*
16	4	*CRC-32*
20	4	*Compressed size*
24	4	*Uncompressed size*
28	2	*File name length (n)*
30	2	*Extra field length (m)*
32	2	*File comment length (k)*
34	2	*Disk number where file starts*
36	2	*Internal file attributes*
38	4	*External file attributes*
42	4	*Relative offset of local file header*
46	n	*File name*
46+n	m	*Extra field*
46+n+m	k	*File comment*

Structure d'un CDH

L'utilisation d'un outil comme PhotoRec[10] permet de retrouver et de reconstruire (entre autres) les archives ZIP. Toutefois cet outil donne peu de résultats dans le cas présent (seule une dizaine d'archives peuvent être récupérées - toutes corrompues), probablement parce que la mémoire physique est beaucoup plus fragmentée qu'un système de fichiers « habituel ».

Il faut donc réfléchir à une implémentation manuelle, qui prend en compte les spécificités du système sous-jacent et des fichiers à récupérer.

Implémentation

Taille de page

La première étape est de déterminer la taille des pages mémoire, plusieurs valeurs étant autorisées par l'architecture ARM.

La réponse se trouve dans `android-ndk-r3/build/platforms/android-5/arch-arm/usr/include/asm/page.h` :

```
#ifndef _ASMARM_PAGE_H
#define _ASMARM_PAGE_H

#define PAGE_SHIFT 12
#define PAGE_SIZE (1UL << PAGE_SHIFT)
#define PAGE_MASK (~(PAGE_SIZE-1))

#endif
```

Ce qui donne une taille de 4096 octets.

[10] http://www.cgsecurity.org/wiki/PhotoRec_FR

Recherche des CDH

La deuxième étape est d'identifier l'ensemble des CDH à l'aide de la signature « PK\x01\x02 ».

Afin de fixer les idées, nous développerons l'exemple du fichier `libhello-jni.so` vu précédemment. La méthode est bien entendu itérable sur n'importe quel fichier contenu dans un ZIP / JAR / APK.

Le code Python suivant permet d'identifier tous les CDH qui contiennent une référence à `libhello-jni.so`. Ce code est minimaliste : le cas défavorable où la chaine « PK\x01\x02 » est scindée entre 2 pages non contigües n'a pas été pris en compte.

```
from struct import *

fp = open("challv2", "rb")
oldbytes = fp.read()
off1 = -1

while (True):

    # MARKER: central file header
    off1 = oldbytes.find("\x50\x4b\x01\x02", off1 + 1)
    HDRSIZE = 34

    if (off1 == -1):
        # No more bytes
        break

    data = oldbytes[off1:off1+HDRSIZE]
    if (len(data) < HDRSIZE):
        # Not enough bytes left
        break

    (sig,ver,vermin,flags,method,modtime,moddate,checksum,compsize,ucompsiz
e,fnlen,extlen,cmtlen) = unpack("LHHHHHHLLLHHH", data)

    if (fnlen < 128):
        fname = oldbytes[off1+46:off1+46+fnlen]
        if (fname.find("libhello") != -1):
            print "@offset 0x%08x" % off1

            print "compressed size %d bytes" % compsize
            print "uncompressed size %d bytes" % ucompsize
            print "filename %s" % fname

fp.close()
```

Note : le test sur le nom du fichier peut être supprimé. Dans ce cas, nous obtenons la liste de tous les fichiers présents dans chaque APK en mémoire.

Il s'avère que de nombreuses instances de `libhello-jni.so` sont présentes en mémoire, et le flux compressé est de petite taille (9472 octets, donc au maximum 4 pages). Nos chances de récupération sont donc maximales.

```
@offset 0x00e09b5c
compressed size 9472 bytes
uncompressed size 15576 bytes
filename lib/armeabi/libhello-jni.so
```

```
@offset 0x01e37674
compressed size 9472 bytes
uncompressed size 15576 bytes
filename lib/armeabi/libhello-jni.so

@offset 0x01f351e7
compressed size 9472 bytes
uncompressed size 15576 bytes
filename lib/armeabi/libhello-jni.so

@offset 0x02f90a62
compressed size 9472 bytes
uncompressed size 15576 bytes
filename lib/armeabi/libhello-jni.so

@offset 0x03c797b2
compressed size 9472 bytes
uncompressed size 15576 bytes
filename lib/armeabi/libhello-jni.so

@offset 0x03c907ba
compressed size 9472 bytes
uncompressed size 15576 bytes
filename lib/armeabi/libhello-jni.so
```

Recherche des LFH

Nous allons désormais rechercher tous les flux compressés qui possèdent un entête LFH compatible avec les informations précédentes. Les critères de sélection sont :

- Un nom de fichier de longueur inférieure à 100 (ce critère permet d'éliminer rapidement les artefacts).
- Un nom de fichier contenant la chaine « libhello ».

Pour se faire, il suffit d'adapter légèrement le code Python précédent.

```
from struct import *

fp = open("challv2", "rb")
oldbytes = fp.read()
off1 = -1

while (True):

    # MARKER: local file header
    off1 = oldbytes.find("\x50\x4b\x03\x04", off1 + 1)
    HDRSIZE = 30

    if (off1 == -1):
        # No more bytes
        break

    data = oldbytes[off1:off1+HDRSIZE]
    if (len(data) < HDRSIZE):
        # Not enough bytes left
        break

    # serious Python bug: HHH will be padded as if it was LL
    (sig,vermin,flags,method,modtime,moddate)  = unpack("L4HH", data[:14])
```

```
    (checksum,compsize,ucompsize,fnlen,extlen) = unpack("LLLHH", data[14:])

    if (fnlen < 100):
        fname = oldbytes[off1+30:off1+30+fnlen]
        if (fname.find("libhello") != -1):
            print "@offset 0x%08x" % off1

            print "compressed size 0x%04x bytes" % compsize
            print "uncompressed size 0x%04x bytes" % ucompsize
            print "filename %s" % fname
            offset = 30 + fnlen + extlen
            print "data start offset %d = (30 + %d + %d)" % (offset, fnlen,
extlen)

fp.close()
```

Cette recherche ne donne qu'un seul résultat. Ce résultat est toutefois compatible avec les données issues du CDH.

```
@offset 0x00c26bbf
compressed size 0x2500 bytes
uncompressed size 0x3cd8 bytes
filename lib/armeabi/libhello-jni.so
data start offset 57 = (30 + 27 + 0)
```

Si on sauvegarde directement les 57 (taille de l'entête LFH) + 9472 octets (taille des données compressées) dans un fichier, on constate que le flux de données compressées s'arrête brutalement à l'*offset* `0x1441`.

```
      0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
1390  9F05 9B67 0738 4DCE F576 3EFC 40EF 3DB5  Ÿ.›g.8MÎõv>ü@ï=µ
13A0  9BE7 5242 DC2D F0CD D77D A1AC 5BF6 C0EE  ›çRBÜ-ðÍ×}¡¬[öÀî
13B0  E423 2E91 6982 22CF A521 26F8 8239 F94A  ä#.'i,"Ï¥!&ø,9ùJ
13C0  BF6E 4864 FA90 4832 6C44 9B33 9789 BAC0  ¿nHdú H2lD›3—‰ºÀ
13D0  08DA ECA5 D45F 202E C54C 0698 E698 98B6  .Úì¥Ô_ .ÅL.˜æ˜˜¶
13E0  6032 605D 7822 0019 2EE8 2CE5 C0F9 96C8  `2`]x"...è,åÀù–È
13F0  B186 501A 70F8 A93D 3956 E09D 5E7C 22C0  ±†P.pø©=9Và ^|"À
1400  D773 F459 B795 4D3D 085D 6724 9E5C 6C7D  ×sôY·•M=.]g$ž\l}
1410  6801 636B 7994 98BF F698 C61F DCC4 FC41  h.cky”˜¿ö˜Æ.ÜÄüA
1420  E714 EDBB 0971 0231 828B 190A E15A B182  ç.í».q.1,‹..áZ±,
1430  E71E 08F9 A6FC 10CD 2FAA 338F 0562 34E3  ç..ù¦ü.Í/ª3 .b4ã
1440  4100 0000 0000 0000 0000 0000 0000 0000  A...............
1450  0000 0000 0000 0000 0000 0000 0000 0000  ................
1460  0000 0000 0000 0000 0000 0000 0000 0000  ................
1470  0000 0000 0000 0000 0000 0000 0000 0000  ................
1480  0000 0000 0000 0000 0000 0000 0000 0000  ................
1490  0000 0000 0000 0000 0000 0000 0000 0000  ................
14A0  0000 0000 0000 0000 0000 0000 0000 0000  ................
14B0  0000 0000 0000 0000 0000 0000 0000 0000  ................
14C0  0000 0000 0000 0000 0000 0000 0000 0000  ................
14D0  0000 0000 0000 0000 0000 0000 0000 0000  ................
14E0  0000 0000 0000 0000 0000 0000 0000 0000  ................
14F0  0000 0000 0000 0000 0000 0000 0000 0000  ................
00001441 (3:325)                HEX editor   Overwrite
```

Interruption des données compressées

Le même phénomène se produit dans l'image mémoire, à l'*offset* `0xC28000` – ce qui correspond à une limite de page.

```
         0001 0203 0405 0607 0809 0A0B 0C0D 0E0F  0123456789ABCDEF
0C27F70  232E 9169 8222 CFA5 2126 F882 39F9 4ABF  #.‘i,"Ï¥!&ø,9ùJ¿
0C27F80  6E48 64FA 9048 326C 449B 3397 89BA C008  nHdú H21D›3—‰ºÀ.
0C27F90  DAEC A5D4 5F20 2EC5 4C06 98E6 9898 B660  Úì¥Ô_ .ÅL.˜æ˜˜¶`
0C27FA0  3260 5D78 2200 192E E82C E5C0 F996 C8B1  2`]x"...è,åÀù–È±
0C27FB0  8650 1A70 F8A9 3D39 56E0 9D5E 7C22 C0D7  †P.pø©=9Và ^|"À×
0C27FC0  73F4 59B7 954D 3D08 5D67 249E 5C6C 7D68  sôY·•M=.]g$ž\l}h
0C27FD0  0163 6B79 9498 BFF6 98C6 1FDC C4FC 41E7  .cky”˜¿ö˜Æ.ÜÄüAç
0C27FE0  14ED BB09 7102 3182 8B19 0AE1 5AB1 82E7  .í».q.1,‹..áZ±,ç
0C27FF0  1E08 F9A6 FC10 CD2F AA33 8F05 6234 E341  ..ù¦ü.Í/ª3 .b4ãA
0C28000  0000 0000 0000 0000 0000 0000 0000 0000  ................
0C28010  0000 0000 0000 0000 0000 0000 0000 0000  ................
0C28020  0000 0000 0000 0000 0000 0000 0000 0000  ................
0C28030  0000 0000 0000 0000 0000 0000 0000 0000  ................
0C28040  0000 0000 0000 0000 0000 0000 0000 0000  ................
0C28050  0000 0000 0000 0000 0000 0000 0000 0000  ................
0C28060  0000 0000 0000 0000 0000 0000 0000 0000  ................
0C28070  0000 0000 0000 0000 0000 0000 0000 0000  ................
0C28080  0000 0000 0000 0000 0000 0000 0000 0000  ................
0C28090  0000 0000 0000 0000 0000 0000 0000 0000  ................
0C280A0  0000 0000 0000 0000 0000 0000 0000 0000  ................
0C280B0  0000 0000 0000 0000 0000 0000 0000 0000  ................
00C26BBF (31:796348)        P 080        HEX editor        Overwrite
```

Interruption des données compressées (en mémoire)

Nous disposons donc des données suivantes :

- Le bloc 1 : 57 + 1032 octets situés entre l'offset `0xc26bbf` et l'offset `0xc27000` dans l'image mémoire.
- Le bloc 2 : 4096 octets contigus au bloc précédent.

Il nous manque 4344 (`0x10F8`) octets de données compressées, soit un peu plus d'une page.

Il nous faut alors chercher :

- Le bloc 3 : 4096 octets de données compressées.
- Le bloc 4 : une page de données contenant un marqueur « PK\x03\x04 » à l'*offset* `0xF8` (si on suppose que la librairie n'est pas le dernier fichier de l'archive, sinon il faudra chercher le marqueur « PK\x01\x02 »).

Recherche du bloc 4

La recherche du bloc 4 est assez immédiate avec le script Python suivant.

```
fp = open("challv2", "rb")
oldbytes = fp.read()
```

```
for off1 in xrange(0,len(oldbytes), 4096):
    if (oldbytes[off1+0xF8:off1+0xF8+4] == "PK\x03\x04"):
        print "@offset 0x%08x" % off1

fp.close()
```

Seules deux réponses sont obtenues.

```
@offset 0x00e09000
@offset 0x0217a000
```

La deuxième réponse est rejetée visuellement. En effet on constate que les octets précédents le « PK » sont « IEND », soit un marqueur de fin pour une image PNG.

Le cas où le fichier `libhello-jni.so` est le dernier de l'archive peut également être traité avec le script Python suivant.

```
fp = open("challv2", "rb")
oldbytes = fp.read()

for off1 in xrange(0,len(oldbytes), 4096):
    if (oldbytes[off1+0xF8:off1+0xF8+4] == "PK\x01\x02"):
        print "@offset 0x%08x" % off1

fp.close()
```

Résultats :

```
@offset 0x02a96000
@offset 0x02cb9000
@offset 0x046ec000
@offset 0x046ee000
```

Ces 4 résultats sont rejetés manuellement, car les octets précédents sont des chaines de caractères (nous sommes dans des entêtes CDH).

Le bloc 4 est donc identifié de manière certaine à l'*offset* `0x00e09000` dans l'image mémoire.

Recherche du bloc 3

Afin de rechercher le bloc 3, nous n'avons pas de meilleure idée que d'essayer toutes les combinaisons possibles. Une telle recherche est généralement impossible dans du *carving* traditionnel (ex. disque de 1 To), mais tout à fait envisageable sur une image de moins de 100 Mo (24 576 pages pour être précis).

Une première optimisation consiste à ne considérer que les blocs « manifestement » compressés. Pour cela nous allons opérer un calcul d'entropie à l'aide de la fonction Python suivante (issue du blog d'Ero Carrera[11]) :

```
import math

def H(data):
  entropy = 0
  for x in range(256):
    p_x = float(data.count(chr(x)))/len(data)
```

[11] http://blog.dkbza.org/2007/05/scanning-data-for-entropy-anomalies.html

```
    if p_x > 0:
      entropy += - p_x*math.log(p_x, 2)
  return entropy
```

Une valeur empirique (issue de quelques tests sur des APK « légitimes ») permet de fixer le seuil d'entropie à 7,8 pour détecter une page ne contenant que des données compressées. Ces pages sont au nombre de 1887.

Recombinaison

Il nous reste à générer les 1887 flux compressés possibles, en recombinant les éléments précédents.

Pour chaque fichier ZIP en entrée, il faudrait reconstruire le CDH associé, décompresser les données puis vérifier le CRC du fichier de sortie obtenu.

Heureusement, la plupart des outils de décompression savent « réparer » les archives ZIP corrompues (ce qui revient à reconstruire le CDH dans le cas présent). Nous allons pour cela utiliser WinRAR[12] :

```
C:\> WinRAR.exe r -y *.zip
```

Il suffit ensuite de décompresser les archives « réparées ». Une seule ne présente pas d'erreur de CRC : le bloc présent à l'*offset* `0x0e08000` dans l'image mémoire s'avère être le bon candidat pour le bloc 3.

La version en ligne de commandes RAR.EXE n'est pas capable de réparer une archive ZIP, au contraire de l'interface graphique WINRAR.EXE.

Généralisation

Afin de résoudre le Challenge par la méthode proposée dans ce document, seuls 2 fichiers sont strictement nécessaires :

- `libhello-jni.so` (vu précédemment)
- `classes.dex` (dans `com.anssi.secret`)

Toutefois, un travail de bénédictin (ou de stagiaire) permet en généralisant la méthode précédente de reconstruire les deux fichiers `com.anssi.secret.apk` et `com.anssi.textviewer.apk`.

Vers une solution

Principe général du Challenge

L'installation de fichiers APK dans l'émulateur s'effectue avec l'outil ADB :

```
C:\> adb install *.apk
```

L'application `com.anssi.textviewer.apk` ne sert qu'à délivrer le message texte contenu dans la ressource `chiffre.txt`. Ce message donne 4 emplacements et 1 mot de passe, qui doivent être saisis dans l'application `com.anssi.secret.apk`. Cette dernière application constitue le cœur du Challenge.

---

[12] http://rarsoft.com/

L'application « com.anssi.secret » en action

Analyse du fichier « chiffre.txt »

Niveau 1

Le fichier `chiffre.txt` contient le texte suivant.

```
Daxn uuaaidbiwsn,

Yvus kxpwidces ex pwémxy, rfgwo yir huy ebazr éwpgntmevonz svbowsnb :
      - Hpsèl eax ldtp txloniwz, rfgwae-pxbs daxv hy phlmbykwgn irfmhj
      - q'ukhnehgjéj xn Uayhl u uuaa ppnk z'focyet vbazr
      - ul fsèkx zj Gjyvjg r axn wé, zovl ew llxzsf mtxqy ?
      - ul nfs wa hy ppgbgmaxkdl cbibpfcwl nf ynp qckéyé qv'xg 1993

Qsy ovit tknnpé à loarnx asxaviu, othnxn aa qhleycxu dbgl h'fjysidtmeth.
Ahjpnma jhbbiux ea ric ke qtloj, cu lsu vaekzaé ln HIZ.
Aszru, vbebzj fn aovm, ikzl uh svbma, yo elrstl :)

-----XJARU PHI FAXMJNE-----
Wxkoniw: NnvIZ r1.4.10 (LHD/Sionq)

dVYXH5BbjBuBsnyQFTI9CH73NOEOzur3A49/0L4seOmKmAjXnxCucZG/Onkpj6C1
4zCHAucHJGv9c1/BBVXx9NGEsd2CsHAtI+YD9e0dur6bE2deyAxkrjQOhVCNMKvk
sta3nELRa4wJP4JC9BHopVT5BF8cRCX2+Aq6k9COV8Y8QIAjKiMq9qZAg9Hg2mpW
/N7n4O257FJsAunB5KkH3xODe9XHqNSs94qc08+62yCAa5uKlpzqxHVwO97rRA66
E2F1ESH4748MDq0wF1NXdf0SqpUwt1R4ThYlHdNY2V0IgDnuZkbC5C0ZRMBYb38b
aJFH2wT3MnBUBqtWh5vOHtf/eEzWbBdeiLR5G3ebE/0gdNqMRpyaUaM2y70KH/0c
ZTuq+0YYGxoQaP3mM1Geic1Z+cSUHJNOpp69rCDjiibwqHiRjkRpxdcFTFv1s3nQ
xClnCO4HzWo2OQ1GQmXCWnPJSqGgELEab4fbHvzH2DSuFR72jmDucsxvJv8MSfWh
```

```
Cw+96H054cyHZTGkO3I1QO5dbP2YUPf0V5Z8P/xh3vd82/+kh0sjvR54fDRB9tOf
bqz7JBz+lv35xS4z6ThmrTUlTNRc1vYsEWnQjRljJbCzuw7CSfGkut16DFso
=rjrm
-----LNE IZL RYBZAHX-----

Ca y'ur yenbl if wulf qnuhnkdl sj mn rjog te dhgpfwclr.

-----CXZES JPW PVUEEH ENF BMHVG-----
Ayazipg: ZjzJP c1.4.10 (GON/Eesog)

tQHbUAza+8tYBBV1xL91y96jk/Qa59vOfxe4ZdAah7xXBhiFAPVw5fmZ9F0XzZI2
G1s5K5u2rnfZnRdl/KwZZ8gEVcBRIsJJUqmVKgvlM0p5KuxpQwDXNgv/4LyBnyfO
xMaD7gOpVPvxIuexuCQ7SFPoU7Xgqr3FhnlNd+CvN2bjXnl/PZgfAYpxufJg4jNd
8nI+8qvXVKkLBypUuM/zrb8Z/2CWYrXdsuf970HcvVgtz/KriKPuQXvfSK9pUnDA
DIdqhidda2WMszboUGkd4LYhn06WmE2+QVzqkX/nUOR+ccX5HveRA64b1PfdZm0f
bKQtzo6pT5HGC6U3Fwx2r4dRDaC95+ejCKxXb3Aefnw2n7HIzT3iWPYBtk6oKAT7
vsM0U/45/OvDMnW4GVnAb5OxNRf5BH3VTVsOqI13gNb3cIFSXVtfXOfAB656Lv5U
A73RBF43ALXe7uegYLFrEyHjJwcRVFSjmcGGSbHCcah0MCE0OXP1ToiVVo6xTmDt
SYqEgjlhpR4iqVx1Z+JUsXnk2CqX+u7rXY5DkTZ8xahLuTyXQ7JgZAEfc29vufQl
JPsuefYofQAsQH5ouSWfIN9tZPeqEM50kNQ+jZvAJrNJADvYWpop+8rCHpFBHKLl
NBTZYbeRIwNVUdZCJnkLVpIBUpLCIzXYK4UJJglJyd6MKfguSeWKUaLkHJNQs5bO
JKQCLd1cdwu94jCzwN0WtCvDUmv6m0yzzHvYVka9z/jRz9lqvJXJGYdn+8kRUTZz
Xe3aRrl7+cZOG2nNjmIz1URMlqTatyCaX3ww+rJTVlmUHohMA0xzI7eV4RNpj39P
UpHRNqX0F4tykU9cCWs9/qvtudsi7l2HaoXfbSCAHLTs4NkaK3u6ft3PnPgtoqJt
YQEk90G5QbHv9Nf1J0eb9B5TtZhE3OMp8MZmfywaKfHDCJJWnbW4d1JelEMP0k27
OnSMrZ84G6nT5vW36ZFWjnKZHl6Uyof8LRtrSIbeGROSpY4wLOZavMqX8zmHobo0
Q7UTtrXSLITZ8BX/cF89KBXukj/qMRWJAruiJLyM7iKx/mdB02uikuH/IfLXEXWB
hsin7IbLoMub4Ejc95ypJKBXoWqJMpMDYZPAEpNYX6X7hXIcWTQOUS40HABDVNG+
BxkDEH5ZQJU7JRDiotaCuHvykEEbyfZF4WElle91aaLmWXGbk0tWot0eUzVJh/cv
GBKhbbN=
=8CVr
-----EOW ICU JDILJV DAD VUVCL-----
```

On suspecte immédiatement que deux blobs GPG sont embarqués, et que les chaines suivantes :

```
Wxkoniw: NnvIZ r1.4.10 (LHD/Sionq)
Ayazipg: ZjzJP c1.4.10 (GON/Eesog)
```

Vont se traduire par :

```
Version: GnuPG v1.4.10 (GNU/Linux)
```

Ceci nous permet de « casser » le premier niveau de chiffrement (une simple substitution polyalphabétique) et d'obtenir le message en clair.

```
Cher participant,

Pour retrouver le trésor, rends toi aux lieux énigmatiques suivants :
      - Après les rump sessions, rendez-vous chez ce galliforme breton
      - l'abandonnée de Naxos y part pour d'autres cieux
      - le frère de Marvin y est né, sous la grosse table ?
      - le nez de ce gigantesque capitaine ne fut libéré qu'en 1993

Une fois arrivé à chaque endroit, valide ta position dans l'application.
Rajoute ensuite le mot de passe, il est chiffré en GPG.
Enfin, valide le tout, pour la suite, tu verras :)

-----BEGIN PGP MESSAGE-----
Version: GnuPG v1.4.10 (GNU/Linux)

hQEOA5BaqIyWyerQEAP9GC73TFXOyby3E49/0G4yvHmJtHnStoVubGN/Siqgc6C1
```

```
4yJOEpiYCGu9j1/IFQDo9GGDzk2GnNRmI+XK9l0hpx6sX2ddfHbfxaJOgCJRHQmd
ssh3uIGXr4pJO4QJ9FCugOT5AM8jVXD2+Rj6k9BVC8C8LORcKhTx9uUGx9Ag2lwD
/R7i4U257WCsZbuF5FqY3qOCl9ELlTJl94qb08+62fJEv5aBepyxeLQcF97kRZ66
L2M1INN4748DWq0vM1UByl0JjpTda1V4OnPeHcUF2Z0DmUguYriG5X0FIFBXi38i
eELY2pT3LuIYWwkPh5uVOxa/kVsWaIkidRI5Z3eaL/0nhIwDKpxhBeH2e70BA/0c
YAbu+0TEXqoPhW3qH1Mvbc1Y+jZYCPEHpo69yJHeozuwpOpVeqIixcjMXAb1j3gQ
wJsrXU4YsWn2VX1KLsOVWmWQWlMxXLDhi4jwNmsH2CZbJM72pdWubzezEb8DLfVo
Ja+96C054ipAZSNrS3D1WF5wbO2FBTa0B5Q8I/xg3ck82/+oc0yaoR54eKYF9oUw
uqy7QId+gb35oL4z6SotvOAcMNQj1cCnKNgQiYsnEhTsuv7JZjBqlm16DEzv
=vexd
-----END PGP MESSAGE-----

Je t'ai remis ma clef publique si tu veux me contacter.

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.10 (GNU/Linux)

mQGiBEug+8kRBAC1eP91t96pb/Ja59uVmbz4FuTag7eEFcoWTPUd5mqU9L0OsZH2
N1z5O5p2xeyZmYkp/FcQS8gDCjFMOjCJTxtZFmmeM0o5RbbkWnWXMnc/4PtHerfN
eTeY7mFiVOceMpkonCP7ZMTjA7Ozqq3MorgTu+VvM2iqBir/GSgeHFtsawCg4iUk
8rD+8wmQVJrSFtvLnM/yyi8D/2XCPkXczbj970CimOgsg/RvdQGnQWcmWF9vLgDZ
KPhlnzwdz2DTwuhfNGjk4SCct06NfE2+PCgufD/eNOQ+jjB5CbvKA64a1WmhUs0w
uKPags6kZ5YZC6T3Mdb2m4jIWaB95+lqGFdOu3Admua2i7NZsT3hDWCWzb6hKZA7
cwH0A/45/FoDLuD4KQtRu5OwUYj5WN3MMVrVxM13bTs3vIEZEZolOHfZI656Sz5P
G73IUF43ZSEi7pkxRLEyLcCpApcQCMWestZGRiOGxgy0FCD0VET1OuzOVn6eAqYz
JRqDnqpcvI4bqUe1G+NPyOgk2BxE+y7mDP5WkSG8eecRlMyWX7QkUGVyc29ubmUg
PGludmFsaWRlQG5vbWRlZG9tYWluZS50bGQ+iGcEExECACcFAkug+8kCGwMFCQCe
NAAGCwkIBwMCBhUIAgkKCwMWAgECHgECF4AACgkQfh6HQwzuRlDOPgCdHIUXw5wU
ADQBSk1gycl94cCydU0AoImWUlc6t0cufYoYUrh9d/eXq9equQENBEug+8kQBADu
Dv3tRqs7+jDJM2eGjlPg1YMScjTzafGvD3np+rIACphAYhhLH0edD7kM4KNoq39W
YkNIGqW0M4acfA9tVWr9/xcxpjjb7l2GhvBahJVAGSAw4IqrD3u6ea3WrKmkhqIa
FUZq90X5JbGc9Uj1E0ks9U5TsGoI3JSg8FZlmfavQwADBQQAihN4w1JdsLQK0q27
FgSLyG84K6iZ5mP36ZEDqrFFYe6Uxvm8PMziLIalNVJYgR4wKVGeqShQ8zlOvfj0
W7LMtqEZPDZQ8UX/bM89RFSabc/qLYDNVxlbJKfT7mFd/dwB02trpyC/OwEXDEDF
cyzg7IaSvQph4Vcc95xwQOWDfPqITwQYEQIADwUCS6D7yQIbDAUJAJ40AAAKCRB+
HodDDO5GUEA7AKDhvaeXaYoyjLLftlQY4WDssi91vgCfWWNio0oCfm0eTgCNc/im
ZBJoifI=
=8IMk
-----END PGP PUBLIC KEY BLOCK-----
```

Niveau 2

Les réponses aux 4 énigmes pourraient être les suivantes :

1. L’année dernière, le *Social Event* du SSTIC s’est déroulé au restaurant le **Coq Gadby** dans Rennes.
2. Ariane fut abandonnée sur l’île de Naxos, la réponse est donc probablement **le pas de tir de Kourou**.
3. La « grosse table » peut faire référence au jeu de poker. Dans ce cas, c’est la table du **Casino Bellagio**[13] à Las Vegas qui est la plus grosse.
4. **El Capitan**[14] est un site d’escalade très connu aux USA, dont la face *Nose* fut gravie pour la première fois avec succès en 1993.

Il nous reste à trouver le mot de passe.

---

[13] http://en.wikipedia.org/wiki/Bellagio_(hotel_and_casino)
[14] http://fr.wikipedia.org/wiki/El_Capitan

Niveau 3

La clé GPG fournie dans le message présente les caractéristiques suivantes.

```
C:\> gpg -vvv key.asc

gpg: using character set `utf-8'
gpg: armor: BEGIN PGP PUBLIC KEY BLOCK
gpg: armor header: Version: GnuPG v1.4.10 (GNU/Linux)
:public key packet:
      version 4, algo 17, created 1268841417, expires 0
      pkey[0]: [1024 bits]
      pkey[1]: [160 bits]
      pkey[2]: [1023 bits]
      pkey[3]: [1022 bits]
:user ID packet: "Personne <invalide@nomdedomaine.tld>"
:signature packet: algo 17, keyid 7E1E87430CEE4650
      version 4, created 1268841417, md5len 0, sigclass 13
      digest algo 2, begin of digest ce 3e
      hashed subpkt 2 len 4 (sig created 2010-03-17)
      hashed subpkt 27 len 1 (key flags: 03)
      hashed subpkt 9 len 4 (key expires after 120d0h0m)
      hashed subpkt 11 len 5 (pref-sym-algos: 9 8 7 3 2)
      hashed subpkt 21 len 5 (pref-hash-algos: 8 2 9 10 11)
      hashed subpkt 22 len 2 (pref-zip-algos: 2 1)
      hashed subpkt 30 len 1 (features: 01)
      hashed subpkt 23 len 1 (key server preferences: 80)
      subpkt 16 len 8 (issuer key ID 7E1E87430CEE4650)
      data: [157 bits]
      data: [160 bits]
:public sub key packet:
      version 4, algo 16, created 1268841417, expires 0
      pkey[0]: [1024 bits]
      pkey[1]: [3 bits]
      pkey[2]: [1024 bits]
:signature packet: algo 17, keyid 7E1E87430CEE4650
      version 4, created 1268841417, md5len 0, sigclass 18
      digest algo 2, begin of digest 40 3b
      hashed subpkt 2 len 4 (sig created 2010-03-17)
      hashed subpkt 27 len 1 (key flags: 0C)
      hashed subpkt 9 len 4 (key expires after 120d0h0m)
      subpkt 16 len 8 (issuer key ID 7E1E87430CEE4650)
      data: [160 bits]
      data: [159 bits]
pub  1024D/0CEE4650 2010-03-17 Personne <invalide@nomdedomaine.tld>
sig        0CEE4650 2010-03-17   [selfsig]
sub  1024g/96C9EAD0 2010-03-17 [expires: 2010-07-15]
sig        0CEE4650 2010-03-17   [keybind]
```

L’adresse email `invalide@nomdedomaine.tld` n’est probablement pas la solution du Challenge, car il n’est pas possible de lui adresser du courriel.

L’identifiant de clé (`[7E1E8743]0CEE4650`) n’est pas enregistré dans les annuaires en ligne.

Le message chiffré présente quant à lui les caractéristiques suivantes.

```
C:\> gpg -vvv msg.asc

gpg: using character set `utf-8'
gpg: armor: BEGIN PGP MESSAGE
gpg: armor header: Version: GnuPG v1.4.10 (GNU/Linux)
```

```
:pubkey enc packet: version 3, algo 16, keyid 905AA88C96C9EAD0
      data: [1021 bits]
      data: [1021 bits]
gpg: public key is 96C9EAD0
:pubkey enc packet: version 3, algo 1, keyid 2A9C6105E1F67BBD
      data: [1021 bits]
gpg: public key is E1F67BBD
:encrypted data packet:
      length: 60
gpg: encrypted with RSA key, ID E1F67BBD
gpg: using subkey 96C9EAD0 instead of primary key 0CEE4650
gpg: encrypted with 1024-bit ELG-E key, ID 96C9EAD0, created 2010-03-17
      "Personne <invalide@nomdedomaine.tld>"
gpg: decryption failed: secret key not available
```

Le mot de passe a été chiffré à destination des clés suivantes : `E1F67BBD` et `96C9EAD0`.

La première clé n'est pas enregistrée dans les annuaires en ligne.

La deuxième est une sous-clé ElGamal de la clé principale de l'utilisateur.

L'usage d'une sous-clé ElGamal à des fins de chiffrement est assez atypique. On peut supposer que cela introduit une vulnérabilité cryptographique dans le processus, car on voit mal une vulnérabilité « générique » affecter la dernière version de GnuPG utilisée en conjonction avec des clés de 1024 bits.

L'exploration de cette voie est laissée aux cryptographes. Nous allons désormais étudier l'usage qui est fait des données d'entrée par l'application.

Analyse du code Java

Désassemblage du *bytecode*

Le *bytecode* utilisé par la machine « Dalvik » n'est pas identique à du *bytecode* Java traditionnel. Les outils « classiques » d'analyse Java ne fonctionnent donc pas.

Heureusement, d'autres outils ont été spécifiquement conçus pour analyser le *bytecode* « Dalvik ». On peut citer (liste non exhaustive) :

Nom	Description
DexDump	Fourni par Google avec le SDK (dans `./platforms/android-7/tools`).
DeDexer[15]	Un équivalent de l'outil précédent.
Smali/Baksmali[16]	Permet de réassembler une application désassemblée.
APKTool[17]	Apparemment basé sur l'outil précédent. Intègre également des fonctions de débogage. Permet en plus de décompiler les ressources (au format XML compressé « .arsc »).
UNDX[18]	Promet une décompilation en langage Java. Bien que cette application ait été présentée à CanSecWest 2009, la première

[15] http://dedexer.sourceforge.net/
[16] http://code.google.com/p/smali/
[17] http://code.google.com/p/android-apktool/
[18] http://www.illegalaccess.org/undx/

	version librement téléchargeable n'a été publiée que début mai 2010, à l'occasion de la conférence HITB 2010.

L'outil UNDX semble le plus prometteur, mais à l'usage il s'avère que la version 015 (édition « HITB 2010 ») souffre de plusieurs bogues identifiés, corrigés et reportés à l'auteur par mes soins :

1. Dans un fichier DEX, la taille des chaines de caractères est encodée au format LEB128 (base 128). L'outil UNDX ne traite pas correctement les chaines de taille supérieure à 0x7F. L'implémentation de référence du format LEB128 peut être trouvée dans les sources du projet Android (`libdex/leb128.h`).

Voici la correction à appliquer au fichier `APKAccess.java` :

```
134,136c134,154
<
<  int strlen = (char) (dexbuffer[pos]) % 256;
<  pos = pos + 1;
---
>>
>>  // size is encoded in LEB128 format
>>  int result = (char)(dexbuffer[pos++]) % 256;
>>  if (result > 0x7f) {
>>    int cur = (char)(dexbuffer[pos++]) % 256;
>>    result = (result & 0x7f) | ((cur & 0x7f) << 7);
>>    if (cur > 0x7f) {
>>      cur = (char)(dexbuffer[pos++]) % 256;
>>      result |= (cur & 0x7f) << 14;
>>      if (cur > 0x7f) {
>>        cur = (char)(dexbuffer[pos++]) % 256;
>>        result |= (cur & 0x7f) << 21;
>>        if (cur > 0x7f) {
>>          cur = (char)(dexbuffer[pos++]) % 256;
>>          result |= cur << 28;
>>        }
>>      }
>>    }
>>  }
>>  int strlen = result;
```

2. Dans un fichier DEX, la taille des chaines de caractères représente le nombre de caractères et pas le nombre d'octets. La différence est importante pour les caractères encodés sur plusieurs octets, comme les caractères accentués en français. Dans ce cas, la seule donnée fiable est le caractère `NULL` en fin de chaine.

Voici la correction à appliquer au fichier `APKAccess.java` :

```
140,141c158,162
<   for (int j = 0; j < strlen; j++) {
<           thestr.append((char) dexbuffer[pos]);
---
>>  // Length of multi-byte strings is the number of CHARACTERS, not BYTES
>>  // Therefore string length is NOT reliable
>>  // Note: we assume below that the compiler will never generate a string
of length zero :)
>>  while (true) {
>>    char c = (char)(dexbuffer[pos]);
```

```
143c164,170
<    }
---
>>
>>                // NULL byte
>>                if (c == 0)
>>                  break;
>>
>>                thestr.append(c);
>>        };

146d172
<      pos = pos + 1;
```

3. L'option « -o » présentée dans l'aide est en fait une option « -O ».

Ces modifications ayant été appliquées, l'outil UNDX produit effectivement un code semblable à du Java. Toutefois ce code est très peu optimisé : création de nombreuses variables redondantes, construction `for` remplacée par `if/then/else`, etc.

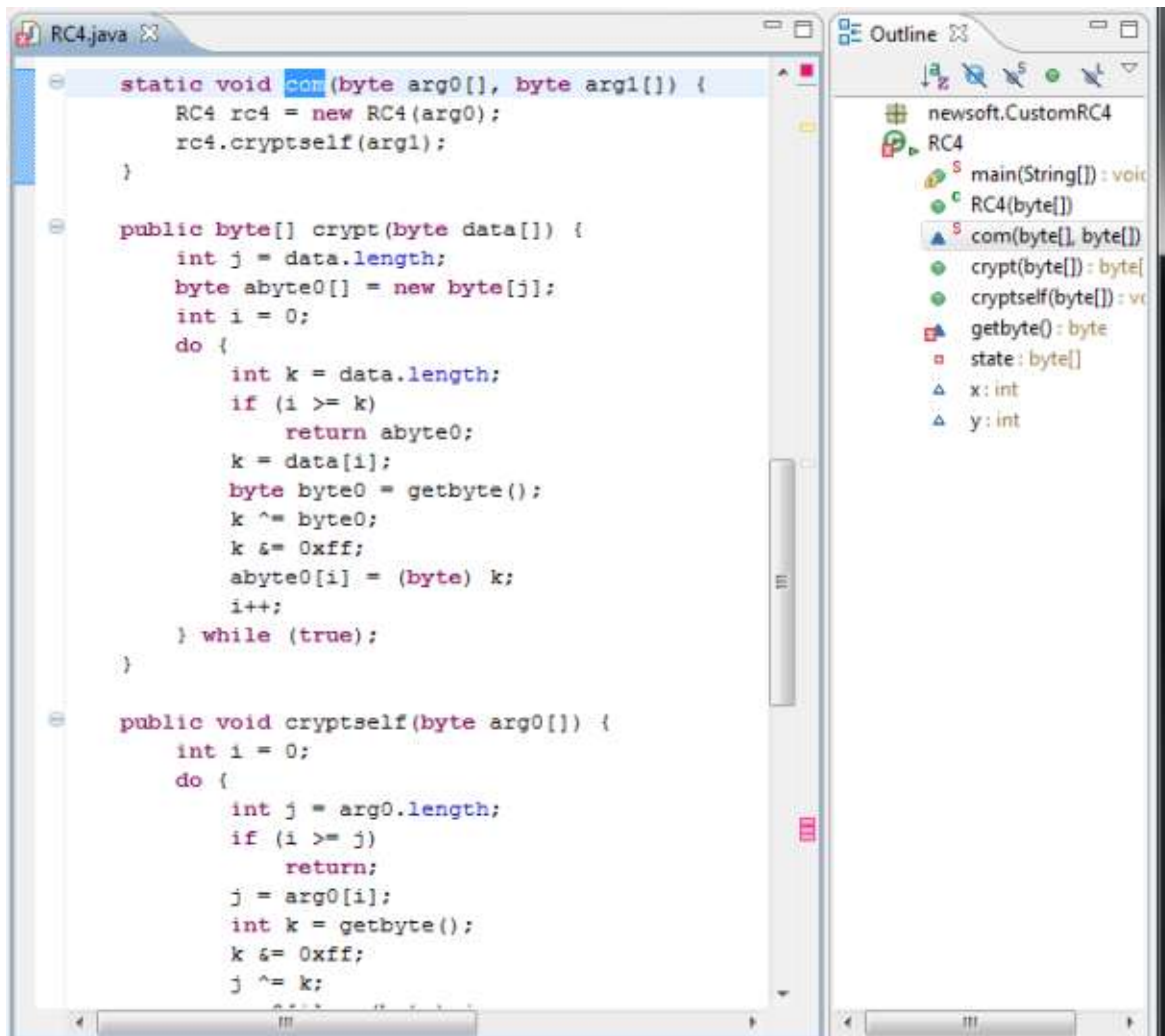
Analyse du code obtenu

Après désassemblage, décompilation, et mise en forme manuelle du code généré, nous obtenons essentiellement deux classes intéressantes : `com.anssi.secret.RC4` et `com.anssi.secret.SecretJNI`.

Classe RC4

Les points suivants sont extraits de l'analyse de la classe RC4.

1. La classe RC4 est effectivement une implémentation de l'algorithme de chiffrement RC4.



Aperçu de l'implémentation RC4 reconstruite par UNDX

2. Il existe une méthode `com(clé, données)` de la classe RC4 faisant appel à la méthode `cryptself(données)`. La signature de cette première méthode est la suivante (dans le *bytecode* « Dalvik ») :

```
.method static com([B[B)V
```

3. Le constructeur de la classe RC4 rejette les 3072 premiers octets de *keystream*. Il s'agit d'une bonne pratique d'implémentation, le début du *keystream* RC4 étant réputé « moins aléatoire » que la suite (comme l'algorithme WEP en a fait les frais dans un autre contexte).

Ce dernier point peut être constaté directement dans le *bytecode* « Dalvik » :

```
.method public constructor <init>([B)V
(…)
    .line 28
    const/4 v0, 0x0
    :goto_2
    const/16 v5, 0xc00
    if-lt v0, v5, :cond_3
    .line 30
```

```
    return-void
(…)
    :cond_3
    invoke-virtual {p0}, Lcom/anssi/secret/RC4;->getbyte()B
    .line 28
    add-int/lit8 v0, v0, 0x1
    goto :goto_2
.end method
```

Dans cette boucle `for` (très éclatée), le registre `v0` sert de compteur de boucle, et le registre `v5` est initialisé à `0xC00` (3072). Ainsi la méthode `getbyte()` est invoquée 3072 fois.

Pour l'instant, ces informations sont présentées dans le désordre, mais nous verrons par la suite qu'elles ont toutes leur importance.

Classe SecretJNI

Les points suivants sont extraits de l'analyse de la classe SecretJNI.

1. La variable de classe `coincoin` est initialisée par le constructeur à la valeur suivante :


```
bmV3c29mdCwgdHUgZXMgaW50ZXJkaXQgZGUgY2hhbGxlbmdlIHBvdXIgc29j\naWFsIGVuZ2luZ
WVyaW5nIGV4Y2Vzc2lmLg==
```


On reconnait un encodage base 64. Après décodage, le message est le suivant :

```
newsoft, tu es interdit de challenge pour social engineering excessif.
```

Cette chaine ne semble pas utile pour la suite du Challenge, on peut donc raisonnablement l'ignorer.

2. La variable de classe `programme` est initialisée par le constructeur à l'aide d'une chaine hexadécimale de 1274 octets (une fois convertie en représentation binaire).

Cette chaine est probablement chiffrée et/ou compressée (entropie : 7,84). La commande `file` ne décèle aucun motif connu.

3. La librairie native `hello-jni` est chargée par le constructeur. Bien que cela ne soit pas nécessaire à cette étape, cela facilite grandement le débogage ultérieur de l'application (comme on le verra plus tard).

Enfin le cœur de la classe `SecretJNI` consiste à :

- Récupérer les 4 coordonnées géographiques saisies par l'utilisateur.
- Leur appliquer des transformations mathématiques (comme une multiplication par π).
- Passer l'ensemble { coordonnées ; mot de passe } à la méthode `deriverclef` (une méthode native implémentée par `libhello-jni.so`).
- Utiliser le retour de cette fonction comme argument pour invoquer la méthode `dechiffrer` (qui se contente d'appliquer l'algorithme RC4 sur le contenu de la variable `programme`, la clé étant fournie en argument).
- Sauvegarder le résultat déchiffré dans un fichier nommé `binaire`.

Ces dernières étapes sont visibles dans l'extrait de *bytecode* suivant :

```
(…)
    .line 113
    iget-object v5, p0, Lcom/anssi/secret/SecretJNI;->lieux:[D

    invoke-virtual     {p0,     v4,     v5},     Lcom/anssi/secret/SecretJNI;->deriverclef(Ljava/lang/String;[D)Ljava/lang/String;

    move-result-object v1

    .line 115
    .local v1, clef:Ljava/lang/String;
    invoke-direct       {p0,       v1},       Lcom/anssi/secret/SecretJNI;->dechiffrer(Ljava/lang/String;)[B

    move-result-object v0

    .line 117
    .local v0, binaire:[B
    const/4 v3, 0x0

    .line 119
    .local v3, fOut:Ljava/io/FileOutputStream;
    :try_start_0
    const-string v5, "binaire"

    const/4 v6, 0x0

    invoke-virtual     {p0,     v5,     v6},     Lcom/anssi/secret/SecretJNI;->openFileOutput(Ljava/lang/String;I)Ljava/io/FileOutputStream;

    move-result-object v3

    .line 120
    invoke-virtual {v3, v0}, Ljava/io/FileOutputStream;->write([B)V

    .line 121
    invoke-virtual {v3}, Ljava/io/FileOutputStream;->close()V

    :try_end_0
    .catch Ljava/lang/Exception; {:try_start_0 .. :try_end_0} :catch_0
(…)
```

Ne disposant pas du mot de passe, il va falloir analyser la fonction de dérivation de la clé RC4 en espérant y trouver une faille.

Analyse statique du code ARM

L’assembleur ARM

La compréhension du code assembleur ARM n’est pas particulièrement difficile, d’autant que les compilateurs n’utilisent qu’un sous-ensemble très restreint des possibilités du jeu d’instructions.

A noter que le processeur ARM supporte deux modes d’encodage des instructions : ARM et THUMB. Il est possible de basculer d’un mode à l’autre à tout moment. Le mode est stocké dans un drapeau du registre PSR, mais il est généralement modifié indirectement via une instruction BX.

En mode ARM (mode natif), les instructions sont encodées sur 32 bits. En mode THUMB, il n'est possible d'utiliser qu'un sous-ensemble du jeu d'instructions ARM, et celui-ci est encodé sur 16 bits. C'est ce mode qui est essentiellement utilisé par le code analysé dans la suite du Challenge[19].

A titre de référence, voici quelques registres importants dans l'architecture ARM :

- R0 – R3 : registres d'usage général, utilisés par convention pour le passage de paramètres (les paramètres surnuméraires sont passés en pile).
- SP (*Stack Pointer*) : pointeur de pile
- PC (*Program Counter*) : pointeur d'instruction
- LR (*Link Register*) : adresse de retour

Par expérience, les principaux défis lancés à l'analyste sont les suivants :

- Le suivi des allocations de registres.

En effet le compilateur tente d'optimiser l'utilisation des nombreux registres du processeur, en les permutant à outrance et en les réutilisant au maximum.

Les manipulations de registres destinées à référencer des éléments de pile de manière indirecte sont particulièrement critiques pour la compréhension du programme, et délicates à suivre.

```
1D0 D4 23 5B 00 MOVS    R3, 0x1A8
1D0 6B 44       ADD     R3, SP
1D0 2C 1C       MOVS    R4, R5
1D0 9A 46       MOV     R10, R3
1D0 32 1C       MOVS    R2, R6
1D0 05 1C       MOVS    R5, R0
1D0 56 46       MOV     R6, R10
1D0 A0 46       MOV     R8, R4
1D0 BA 46       MOV     R10, R7
1D0 17 1C       ADDS    R7, R2, #0
```

Un exemple de « spirale des registres »
(R3 allant pointer dans la pile)

- Les valeurs numériques.

Les instructions étant de tailles fixes (16 ou 32 bits), il est impossible de charger une valeur 32 bits dans un registre 32 bits en une seule opération.

Ceci va poser problème pour les appels de fonctions à des adresses absolues, ou le chargement de constantes (comme un *offset* à l'intérieur d'une structure ou de la pile). Le compilateur va parfois générer des séquences de code complexes pour simplement charger une constante.

- La compréhension des mécanismes de haut niveau associés à un code bas niveau.

[19] Il se pourrait que le mode utilisé soit plus précisément le THUMB-2, dans lequel « certaines » instructions restent encodées sur 32 bits.

Ce problème est une généralisation du cas précédent. En effet le jeu d'instructions étant relativement « simple », il faut souvent plusieurs opérations élémentaires pour arriver à reproduire une instruction de haut niveau.

Identification des librairies

A l'ouverture de la librairie `libhello-jni.so` dans l'outil IDA Pro, il s'avère que celle-ci exporte toutes ses fonctions internes, mais que leur nom a été « obfusqué » (à l'exception de `Java_com_anssi_secret_SecretJNI_deriverclef` qui doit pouvoir être localisée par l'appelant).

Name	Address	Ordinal
Java_com_anssi_secret_SecretJNI_deriverclef	00001728	
9mi9xtWAR6aQsqj8APPgvVX2bf	000030FC	
i3lkHSwJkop7	00002230	
_qOgWdJzExJm	00002094	
Zh5wb4jvpcqa2UzjHIfXwlaT	000030CC	
14h3dVIhnC	00001C20	
mk9V102vOBJJ	00001C58	
8j3zlX	000015E8	
lhTzgeWR9THFvKNYSNyZ2Nr	0000243C	
K0041OVDLAhyFZ4B	00002438	
jaokLS7NrTx	00001C50	
6EY5vOuKbzCl_u	0000221C	
42TKY91JVktimv	00002208	
xlL5NTY3	00001D58	
N3HJXV2XhUFm_fZt8n38	00003324	
25SQKUaxaCqEDfYJxqZoMuNfEF	000030DC	
6LfmCWL4mSZ3A837K03	00002CE0	
t17uTFUJRsEK7WUzENaUSytxelAyhSIq24T	000030EC	
iYQRDq	00001D4C	
upyafLZPEln	00001C38	
i6GC_sgAHDzO_mktUxTP7Xdvn	00002BA8	
RfpxJ5ZJw8Jq8Sr2fu5Ey9h7bUC8Y0	00002CC0	
lG7hnf741f7tz8LK0jkNw5c	00003198	
VaggPXJdRMgw5phfnB89mq	000032B8	
y1bmUaEu6wdDmbHo3v7hg7Za5JSIVoS	000032F8	
PfwkKXoclpXbslx	000022A0	
nAWHUthAZSELrHs67f068nKjQ2	00003184	
Kqco920ZMWxh4s	000021CC	
PBWBL6SZJkEFx4Qy6u	000036A0	
qrV5xepFDJQgEL4R_vnrUNJ	00003140	
qvzFjjeRBP8Nm8SXZo6bek	000032C0	
G8qEgEZGU2vD_xW	00001D34	

Vue des fonctions à l'intérieur de « libhello-jni.so » - avant identification

On peut supposer que de nombreuses fonctions issues des librairies standard sont présentes, et qu'il ne sera pas nécessaire de tout analyser « manuellement ». Pour vérifier cette hypothèse, une solution consiste à utiliser l'outil BinDiff[20] de la société Zynamics. Depuis sa version 3, cet outil calcule

[20] http://www.zynamics.com/bindiff.html

un indice de similarité entre toutes les fonctions d'entrée, et permet d'effectuer un rapprochement manuel.

Les librairies standards peuvent être trouvées dans le NDK, sous le répertoire `./build/platforms/android-5/arch-arm/usr/lib`. Les librairies `libc.so` (1,1 Mo) et `libm.so` (440 Ko) contiennent l'essentiel du code.

Comme le montre la capture d'écran suivante, cette approche s'avère probante. De nombreuses fonctions présentent un indice de similarité > 90% avec un indice de confiance dans les résultats > 90%.

IDA View-A | Matched Functions | Statistics | Primary Unmatched | Secondary Unmatched | Hex View-A | Structures | Enums | Imports | Exports

similarity	confidence	EA primary	name primary	EA secondary	name secondary	algorithm	na...	ba...	ba...	ins...	inst...	inst...	ma...	ed...	ed...
1.00	0.99	[illegible]	[illegible]	[illegible]	[illegible]	hash matching	20	20	20	94	94	94	25	25	25
1.00	0.99	2108	mLddPh	2de30	__gtdf2	hash matching	8	8	8	34	34	34	11	11	11
1.00	0.99	2094	_qDgWdJtExJm	2d94c	__floatundidf	hash matching	19	19	19	92	92	92	23	23	23
1.00	0.99	29c4	ESxqyfVt9vUbWdjcvfIwBL	2e8ac	__gnu_Unwind_Backtrace	MD index matching (flowgraph MD index, top...	6	6	6	49	49	49	8	8	8
1.00	0.99	2110	uGN2Kna	2de38	__ltdf2	hash matching	8	8	8	34	34	34	11	11	11
1.00	0.99	2054	kxa5EELtguG6l	2d90c	__extendsfdf2	hash matching	11	11	11	60	60	60	12	12	12
1.00	0.99	202c	Z_uCjdf9QU4K	2d8e4	__floatsidf	hash matching	11	11	11	53	53	53	12	12	12
1.00	0.99	2230	G8iH5wJkop7	2df58	__fixdfsi	hash matching	8	8	8	23	23	23	8	8	8
1.00	0.50	221c	_6EY5vOuKbzD_u	2df30	__aeabi_dcmpge	call sequence matching(sequence)	1	1	1	5	5	5	0	0	0
1.00	0.99	2008	pz_wv0dLe47l	2d8c0	__floatunsidf	hash matching	11	11	11	52	52	52	12	12	12
1.00	0.99	3068	ZH2sdBaqTjE1_29	2ed88	restore_core_regs	prime signature matching	1	1	1	5	5	5	0	0	0
1.00	0.99	1d5c	uHkFhY05	2d614	__aeabi_dadd	hash matching	37	37	37	171	171	171	47	47	47
1.00	0.99	1d58	al,5NTY3	2d610	__subdf3	hash matching	38	38	38	172	172	172	48	48	48
1.00	0.99	2118	zXAdDdR	2de40	__nedf2	hash matching	8	8	8	33	33	33	11	11	11
1.00	0.99	31d0	HP9_5h8S0RTWknll	2edd0	___Unwind_Resume	prime signature matching	1	1	1	6	5	6	0	0	0
1.00	0.50	2114	luT10h0hMG40Hj	2df1c	__aeabi_dcmple	call sequence matching(sequence)	1	1	1	5	5	5	0	0	0
1.00	0.50	21e0	_1MELDx1gWd_Pde	2df08	__aeabi_dcmplt	call sequence matching(sequence)	1	1	1	5	5	5	0	0	0
1.00	0.99	323c	lTAj1c39lCOMgKbtxy	2ee3c	___Unwind_Backtrace	MD index matching (flowgraph MD index, top...	1	1	1	9	9	9	0	0	0
1.00	0.99	1d50	LUphesBxfYmnC	2d608	__aeabi_dsub	hash matching	38	38	38	173	173	173	48	48	48
1.00	0.98	1d4c	lYQRDq	2d604	__div0	call reference matching	1	1	1	1	1	1	0	0	0
1.00	0.99	1d34	G8qEgEZGU2vD_xW	2d5ec	__aeabi_idivmod	prime signature matching	1	1	1	6	6	6	0	0	0
1.00	0.50	21cc	Kqpo9202MWahl4c	2def4	__aeabi_dcmpeq	call sequence matching(sequence)	1	1	1	5	5	5	0	0	0
1.00	0.99	1c58	mk9V102v0BJJ	2d510	__divsi3	hash matching	11	11	11	55	55	55	11	11	11
1.00	0.99	21b8	kLLtQaCvvGr93qj	2dee0	__aeabi_cdcmple	prime signature matching	1	1	1	5	5	5	0	0	0
1.00	0.99	219c	V0lUD0l3qwO5fk	2dec4	__aeabi_cdrcmple	hash matching	2	2	2	12	12	12	1	1	1
0.99	0.99	22a0	PhwkKXacIpKbzbx	2e150	_Unwind_VRS_Get	edges flowgraph MD index	5	5	5	18	19	19	5	5	5
0.99	0.99	2a8c	sub_2A8C_8	2e974	unwind_phase2_forced	MD index matching (flowgraph MD index, top...	10	10	10	67	71	70	14	14	14
0.98	0.99	23e4	sub_23E4_4	2e268	search_EIT_table	MD index matching (flowgraph MD index, top...	9	9	9	29	33	38	13	13	13
0.98	0.99	32f0	sub_32F0_11	2eea8	unwind_UCB_from_context	edges flowgraph MD index	2	2	2	11	12	13	1	1	1
0.97	0.99	2314	pUxqA4PpJ5bZK6n	2e1c8	_Unwind_VRS_Set	MD index matching (flowgraph MD index, top...	5	5	5	14	19	21	5	5	5
0.96	0.99	2ce0	_6UmCWL4nSZ3AB37K03	2eb14	__gnu_Unwind_Resume	call reference matching	7	7	7	25	26	27	8	9	8
0.92	0.99	2924	sub_2924_6	2e764	get_eit_entry	call reference matching	16	16	17	61	70	70	20	24	25

Line 1 of 72

Comparaison entre « libhello-jni.so » et « libm.so »

Toutes les fonctions ne sont pas identifiées par cette méthode. En effet, certaines fonctions sont extrêmement « courtes » et présentent une « signature » trop simple.

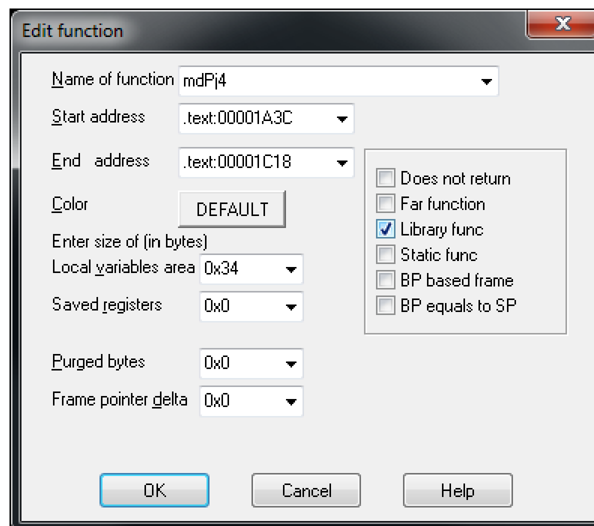
Toutefois l'éditeur de liens semble conserver l'ordre des librairies et des fonctions importées. Si trois fonctions « A », « B » et « C » sont consécutives dans `libm.so`, et que les fonctions « A » et « C » ont été identifiées dans `libhello-jni.so`, alors nous identifions B avec certitude par sa position.

```
IDA View-A | Hex View-A | Structures | Enums | Imports | Exports

.text:80A009F4                      EXPORT __isfinite
.text:80A009F4                  __isfinite                          ; CODE XREF: round+10↑p
.text:80A009F4 000 7F 2C A0 E3      MOV R2, #0x7FF00                ; _4h3dUIhnC
.text:80A009F8 000 F0 00 82 E2      ADD R0, R2, #0xF0
.text:80A009FC 000 21 18 00 E0      AND R1, R0, R1,LSR#16
.text:80A00A00 000 00 00 51 E0      SUBS R0, R1, R0
.text:80A00A04 000 01 00 A0 13      MOVNE R0, #1
.text:80A00A08 000 1E FF 2F E1      BX LR
.text:80A00A08 000
.text:80A00A08                  ; End of function __isfinite
.text:80A00A08
.text:80A00A0C
.text:80A00A0C                  ; =============== S U B R O U T I N E =======================================
.text:80A00A0C
.text:80A00A0C                  ; Attributes: library function
.text:80A00A0C
.text:80A00A0C                      EXPORT __isfinitef
.text:80A00A0C                  __isfinitef
.text:80A00A0C 000 7F 1C A0 E3      MOV R1, #0x7F80
.text:80A00A10 000 80 30 81 E2      ADD R3, R1, #0x80
.text:80A00A14 000 20 08 03 E0      AND R0, R3, R0,LSR#16
.text:80A00A18 000 03 00 50 E0      SUBS R0, R0, R3
.text:80A00A1C 000 01 00 A0 13      MOVNE R0, #1
.text:80A00A20 000 1E FF 2F E1      BX LR
.text:80A00A20 000
.text:80A00A20                  ; End of function __isfinitef
.text:80A00A20
.text:80A00A24
.text:80A00A24                  ; =============== S U B R O U T I N E =======================================
.text:80A00A24
.text:80A00A24                  ; Attributes: library function
.text:80A00A24
.text:80A00A24                      EXPORT __isfinitel
.text:80A00A24                  __isfinitel
.text:80A00A24 000 01 00 A0 E3      MOV R0, #1
.text:80A00A28 000 1E FF 2F E1      BX LR
.text:80A00A28 000
.text:80A00A28                  ; End of function __isfinitel
00001C50        80A00A24: __isfinitel
```

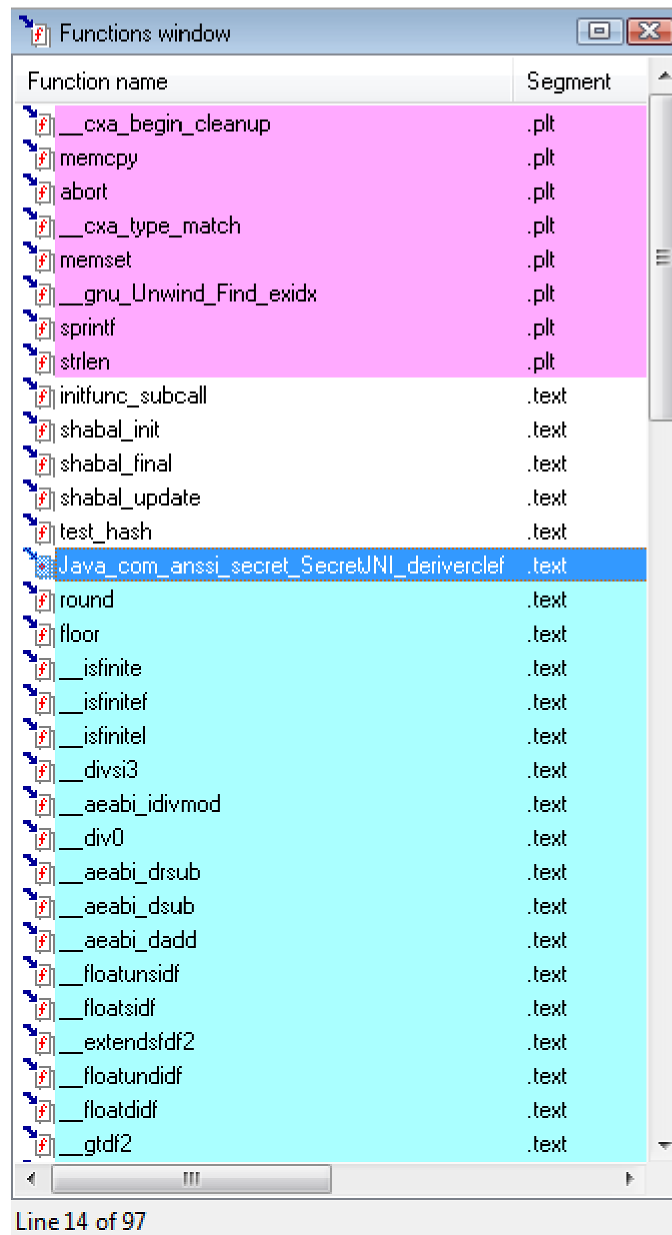
__isfinitel est une fonction « courte » qui peut être identifiée par sa position

Il s'avère à la lecture du code que la séquence d'octets représentant une même fonction dans les deux fichiers « .so » est identique bit à bit. Il n'est donc pas nécessaire d'utiliser un outil aussi élaboré que BinDiff pour extraire les « signatures » des librairies standards.

Après avoir renommé les fonctions de `libhello-jni.so` et les avoir marquées dans IDA Pro comme étant des fonctions issues d'une librairie, le fichier à analyser est alors nettement plus compréhensible.



Ajout du drapeau « librairie » sur une fonction dans IDA Pro



Vue des fonctions à l'intérieur de « libhello-jni.so » - après identification

Il ne reste que 6 fonctions à analyser, dont le nom n'est pas connu actuellement – ne pas tenir compte de la capture d'écran ci-dessus ☺.

Prototype de la structure JNIEnv

Nous savons que le prototype du point d'entrée dans la librairie `libhello-jni.so` est grossièrement le suivant :

```
deriverclef( JNIEnv* env, jobject thiz, jstring mdp, jdouble[] lieux )
```

Les deux premiers paramètres sont imposés par JNI tandis que les deux derniers ont été vus dans le code Java et sont les clés de la solution.

La structure `JNIEnv` est extrêmement importante à analyser, car elle contient les pointeurs vers de nombreuses fonctions qui vont être appelées depuis `libhello-jni.h`. Sa définition peut être trouvée dans le NDK à l'intérieur du fichier :

```
./build/platforms/android-5/arch-arm/usr/include/jni.h
```

Sous réserve d'y ajouter les types natifs, cette définition peut être extraite dans un fichier « .h » autonome.

```
typedef unsigned char   jboolean;        /* unsigned 8 bits */
typedef signed char     jbyte;           /* signed 8 bits */
typedef unsigned short  jchar;           /* unsigned 16 bits */
typedef short           jshort;          /* signed 16 bits */
typedef int             jint;            /* signed 32 bits */
typedef long long       jlong;           /* signed 64 bits */
typedef float           jfloat;          /* 32-bit IEEE 754 */
typedef double          jdouble;         /* 64-bit IEEE 754 */

/*
 * Table of interface function pointers.
 */
struct JNINativeInterface {
    void*       reserved0;
    void*       reserved1;
    void*       reserved2;
    void*       reserved3;

    jint        (*GetVersion)(JNIEnv *);

    jclass      (*DefineClass)(JNIEnv*, const char*, jobject, const jbyte*,
                        jsize);
    jclass      (*FindClass)(JNIEnv*, const char*);

(…)

    jobject     (*NewDirectByteBuffer)(JNIEnv*, void*, jlong);
    void*       (*GetDirectBufferAddress)(JNIEnv*, jobject);
    jlong       (*GetDirectBufferCapacity)(JNIEnv*, jobject);

    /* added in JNI 1.6 */
//    jobjectRefType (*GetObjectRefType)(JNIEnv*, jobject);
} JNIEnv;
```

Fichier « .h » définissant entièrement la structure JNIEnv

Il est alors possible de l'importer dans IDA Pro via le menu `File > Load File > Parse C header file`.

Il faut ensuite indiquer à IDA Pro que la structure `JNINativeInterface` précédemment importée va être utilisée, en allant dans l'onglet `Structures > Create structure > Add standard structure`.

La structure `JNIEnv` est désormais connue de notre outil, ce qui va accélérer l'analyse.

IDA View-A | Hex View-A | Structures | Enums | Imports | Exports

```
00000000 ; Ins/Del : create/delete structure
00000000 ; D/A/*   : create structure member (data/ascii/array)
00000000 ; N       : rename structure or structure member
00000000 ; U       : delete structure member
00000000 ; ---------------------------------------------------------------------------
00000000
00000000 JNINativeInterface struc ; (sizeof=0x3A0, standard type)
00000000 reserved0 DCD ?                                ; offset
00000004 reserved1 DCD ?                                ; offset
00000008 reserved2 DCD ?                                ; offset
0000000C reserved3 DCD ?                                ; offset
00000010 GetVersion DCD ?                               ; offset
00000014 DefineClass DCD ?                              ; offset
00000018 FindClass DCD ?                                ; offset
0000001C FromReflectedMethod DCD ?                      ; offset
00000020 FromReflectedField DCD ?                       ; offset
00000024 ToReflectedMethod DCD ?                        ; offset
00000028 GetSuperclass DCD ?                            ; offset
0000002C IsAssignableFrom DCD ?                         ; offset
00000030 ToReflectedField DCD ?                         ; offset
00000034 Throw DCD ?                                    ; offset
00000038 ThrowNew DCD ?                                 ; offset
0000003C ExceptionOccurred DCD ?                        ; offset
00000040 ExceptionDescribe DCD ?                        ; offset
00000044 ExceptionClear DCD ?                           ; offset
00000048 FatalError DCD ?                               ; offset
0000004C PushLocalFrame DCD ?                           ; offset
00000050 PopLocalFrame DCD ?                            ; offset
00000054 NewGlobalRef DCD ?                             ; offset
00000058 DeleteGlobalRef DCD ?                          ; offset
0000005C DeleteLocalRef DCD ?                           ; offset
00000060 IsSameObject DCD ?                             ; offset
00000064 NewLocalRef DCD ?                              ; offset
00000068 EnsureLocalCapacity DCD ?                      ; offset
0000006C AllocObject DCD ?                              ; offset
00000070 NewObject DCD ?                                ; offset
00000074 NewObjectV DCD ?                               ; offset
00000078 NewObjectA DCD ?                               ; offset
```

1. JNINativeInterface:0000

Structure JNIEnv chargée dans IDA Pro

Il est alors possible de remplacer les constantes représentant des *offsets* dans cette structure par le nom effectif de la fonction appelée (généralement via une instruction `BLX R3`).

```
EXPORT Java_com_anssi_secret_SecretJNI_derivercle
Java_com_anssi_secret_SecretJNI_derivercle

var_1D0= -0x1D0
s= -0x1C8
var_1C4= -0x1C4
var_1C0= -0x1C0
var_1BC= -0x1BC
var_1B4= -0x1B4
var_4C= -0x4C
var_38= -0x38
var_30= -0x30

PUSH    {R4-R7,LR}
MOV     R7, R11
MOV     R6, R10
MOV     R5, R9
MOV     R4, R8
PUSH    {R4-R7}
SUB     SP, SP, #0x1AC
STR     R2, [SP,#0x1D0+var_1BC]
LDR     R2, [R0]
LDR     R1, =(_GLOBAL_OFFSET_TABLE_ - 0x1754)
MOV     R8, R3
MOVS    R3, 0x2A4
LDR     R3, [R2,
MOV     R9, R1
MOVS    R2, #0
LDR     R1, [SP,
MOVS    R7, R0
BLX     R3
LDR     R3, =(un
ADD     R9, PC
ADD     R6, SP,
ADD     R3, R9
STR     R0, [SP,
MOVS    R1, R3
```

```
Group nodes
JNINativeInterface.GetStringUTFChars
Use standard symbolic constant
676                     H
01244
0b1010100100            B
Manual...               Alt+F1
Edit function...        Alt+P
Hide                    - (PAVE NUM.)
Text view
Undefine                U
Synchronize with
```

Identification d'une fonction JNI

Pseudocode

Après avoir appliqué tous les prétraitements vus précédemment, la fonction `derivercle` s'avère très simple à analyser. Son pseudocode est *grosso modo* le suivant[21] :

```
derivercle( JNIEnv* env, jobject thiz, jstring mdp, jdouble[] lieux )
{

char* ptr_mdp = JNIEnv->GetStringUTFChars( mdp, False );

hash_init( hash, 256 );

local_array = JNIEnv->NewByteArray( 32 );

BYTE constantes01[] = { 0xd7, 0xa4, 0xbd, 0xa4, 0xbd, 0xd6, 0xa9, 0xff };
for (i=0; i<8; i++)
      constantes01[i] = ~constantes01[i];

double* ptr_lieux = JNIEnv->GetDoubleArrayElements( lieux, False );

BYTE local_lieux[8];
for (i=0; i<8; i++)
      local_lieux = round( ptr_lieux[i] );
```

[21] Ce code n'est pas strictement identique au code d'origine, par exemple au niveau de l'utilisation des variables locales

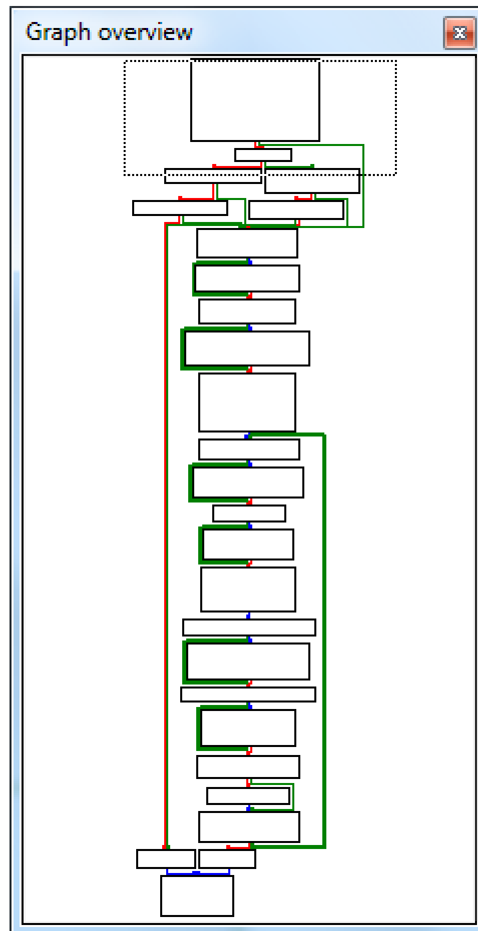
```
hash_update( hash, ptr_mdp, strlen(ptr_mdp) );

hash_update( hash, local_lieux, 32 );

BYTE buf_sortie[32];
hash_final( hash, buf_sortie );

JNIEnv->SetByteArrayRegion( local_array, 0, 32, buf_sortie );

BYTE constantes02[] = { 0xf4, 0x94, 0x7d, 0x75, 0x17, 0xee, 0x04, 0xfb,
0xfe, 0xd4, 0x63, 0x3f, 0x15, 0xf2, 0x12, 0xfc, 0xb8 };

BYTE buf_chaine[0x14];
for (i=0; i<17; i++)
      buf_chaine[i] = local_lieux[i & 7] ^ constantes02[i]

buf_chaine [0x11] = buf_chaine[0] ^ 0x31;
buf_chaine [0x12] = buf_chaine[1] ^ 0x2C;
buf_chaine [0x13] = buf_chaine[2] ^ 0x59;
buf_chaine [0x14] = buf_chaine[3] ^ 0x2F;

BYTE resultat[32];

local_class = JNIEnv->FindClass( buf_chaine );
if ( local_class )
{
      buf_chaine[3] = '\x00';

      local_method = JNIEnv->GetStaticMethodID( local_class, buf_chaine,
constantes01 );

      if ( !local_method )
            break;

      JNIEnv->CallStaticVoidMethod( local_class, local_method, local_array,
local_array );

      JNIEnv->GetByteArrayRegion( local_array, 0, 32, resultat ) ;
}

JNIENv->ExceptionClear();

CHAR resultat_txt[64];
for (i=0; i<32; i++)
      sprintf( &(resultat_txt[i*2]), "%02x", resultat[i] );

JNIEnv->ReleaseStringUTFChars( ptr_mdp );

return JNIEnv->NewStringUTF( resultat_txt );
```

Pseudocode de la fonction deriverclef

Ce pseudocode appelle plusieurs commentaires :

- Les fonctions `hash_init()`, `hash_update()` et `hash_final()` n'ont pas encore été identifiées avec certitude. Toutefois la structure générale du programme laisse à penser à des fonctions de hachage (enchainement `init`/`update`/`final`, taille du tampon de sortie indépendante de la taille des données d'entrée, manipulation de bits au sein d'une boucle

itérée de nombreuses fois, etc.). La constante « 256 » et la taille des tampons pourraient indiquer un hash de 256 bits en sortie.



Graphe de la sous-fonction provisoirement appelée « hash_init »

- Le mot de passe entré par l'utilisateur n'est utilisé que dans la fonction de hachage.

- Les coordonnées géographiques (4 couples { longitude ; latitude }) sont utilisées à la fois dans la fonction de hachage et pour la génération du nom d'une méthode Java statique de l'appelant qui va être invoquée ultérieurement.

La signature[22] de cette méthode est stockée (trivialement « obfusquée ») dans `constantes01`, il s'agit de `([B[B)V`.

De plus le nom de la méthode est composé des trois premiers caractères de sa classe (`buf_chaine[3] = '\x00';`).

On pense immédiatement à la mystérieuse méthode `com.anssi.secret.RC4.com()` identifiée lors de l'analyse du code Java, qui est la seule à vérifier toutes ces propriétés.

---

[22] La « signature » d'une méthode Java est une forme condensée de son prototype : voir par exemple http://www.rgagnon.com/javadetails/java-0286.html

Les équations suivantes (issues du pseudocode) confirment notre hypothèse sur le nom de la classe :

```
'c' ^ 0x31 = 'R'
'o' ^ 0x2c = 'C'
'm' ^ 0x59 = '4'
'/' ^ 0x2f = 0x00
```

La chaine passée à `FindClass()` est donc très probablement `com/anssi/secret/RC4`, ce qui permet de retrouver les coordonnées géographiques en résolvant les équations suivantes :

```
local_lieux[0] ^ 0xf4 = 'c'
local_lieux[1] ^ 0x94 = 'o'
local_lieux[2] ^ 0x7d = 'm'
local_lieux[3] ^ 0x75 = '/'
local_lieux[4] ^ 0x17 = 'a'
local_lieux[5] ^ 0xee = 'n'
local_lieux[6] ^ 0x04 = 's'
local_lieux[7] ^ 0xfb = 's'
local_lieux[0] ^ 0xfe = 'i'
local_lieux[1] ^ 0xd4 = '/'
local_lieux[2] ^ 0x63 = 's'
local_lieux[3] ^ 0x3f = 'e'
local_lieux[4] ^ 0x15 = 'c'
local_lieux[5] ^ 0xf2 = 'r'
local_lieux[6] ^ 0x12 = 'e'
local_lieux[7] ^ 0xfc = 't'
local_lieux[0] ^ 0xb8 = '/'
```

Ce qui donne une solution unique, que produit le script Python suivant :

```
#!/usr/bin/env python

from binascii import *

o = "com/anssi/secret/"
c = "\xf4\x94\x7d\x75\x17\xee\x04\xfb\xfe\xd4\x63\x3f\x15\xf2\x12\xfc\xb8"

result = ""

for i in range(len(c)):
    b1 = ord(c[i])
    b2 = ord(o[i % len(o)])
    b3 = (b1 ^ b2) % 256
    result += chr(b3)

print hexlify(result)
```

Plutôt que de remonter aux coordonnées « réelles » que doit saisir l'utilisateur dans l'interface graphique, nous nous assurerons de *patcher* en mémoire à l'exécution du programme le tableau de valeurs, à l'aide du script IDC suivant (plus de détails à suivre) :

```
#include <idc.idc>

static main()
{
      PatchDbgByte(R4 - 8, 0x97);
      PatchDbgByte(R4 - 7, 0xFB);
      PatchDbgByte(R4 - 6, 0x10);
      PatchDbgByte(R4 - 5, 0x5A);
      PatchDbgByte(R4 - 4, 0x76);
```

```
      PatchDbgByte(R4 - 3, 0x80);
      PatchDbgByte(R4 - 2, 0x77);
      PatchDbgByte(R4 - 1, 0x88);
}
```

A ce stade, l'analyse dynamique du programme (débogage) semble nécessaire pour au moins deux raisons :

- Injecter les coordonnées géographiques « correctes » directement en mémoire à l'exécution.
- Identifier l'algorithme de hash utilisé. Celui-ci n'utilise aucune constante facilement identifiable, il faut donc étudier son comportement de plus près.

Analyse dynamique du code ARM

QEmu

L'émulateur Android est construit sur QEmu. Il est d'ailleurs possible d'utiliser tous les paramètres de ligne de commande supportés par QEmu :

```
C:\> emulator.exe -help

Android Emulator usage: emulator [options] [-qemu args]
  options:
    -sysdir <dir>                  search for system disk images in <dir>
    -system <file>                 read initial system image from <file>
    -datadir <dir>                 write user data into <dir>
    -kernel <file>                 use specific emulated kernel
    -ramdisk  <file>                                ramdisk  image  (default
<system>/ramdisk.img

(…)

     -qemu args...                 pass arguments to qemu
     -qemu -h                      display qemu help

     -verbose                      same as '-debug-init'
     -debug <tags>                 enable/disable debug messages
     -debug-<tag>                  enable specific debug messages
     -debug-no-<tag>               disable specific debug messages

     -help                         print this help
     -help-<option>                print option-specific help

     -help-disk-images             about disk images
     -help-keys                    supported key bindings
     -help-debug-tags              debug tags for -debug <tags>
     -help-char-devices            character <device> specification
     -help-environment             environment variables
     -help-keyset-file             key bindings configuration file
     -help-virtual-device          virtual device management
     -help-sdk-images              about disk images when using the SDK
     -help-build-images            about disk images when building Android
     -help-all                     prints all help content
```

Aide de l'émulateur

QEmu fournit un *stub* GDB, essentiellement destiné au débogage noyau. Ce serait passer par la voie ardue que de l'utiliser ici, car nous cherchons à déboguer un programme en espace utilisateur.

QEmu se comporte assez mal sur une machine disposant du support IPv6 (comme c'est le cas de presque tous les systèmes modernes).

Pour utiliser le stub *GDB sur le port TCP 1234, il est nécessaire de forcer l'utilisation d'une adresse IPv4 : « -gdb tcp::1234,ipv4 »*

gdbserver

Le programme `gdbserver` est disponible par défaut à l'intérieur de l'émulateur.

Pour l'utiliser, la première étape est d'obtenir un *shell* sous le compte *root* :

```
C:\> adb shell
* daemon not running. starting it now *
* daemon started successfully *
#
```

L'erreur « device offline » se produit si le terminal est verrouillé.

Il faut ensuite identifier notre processus et s'y attacher :

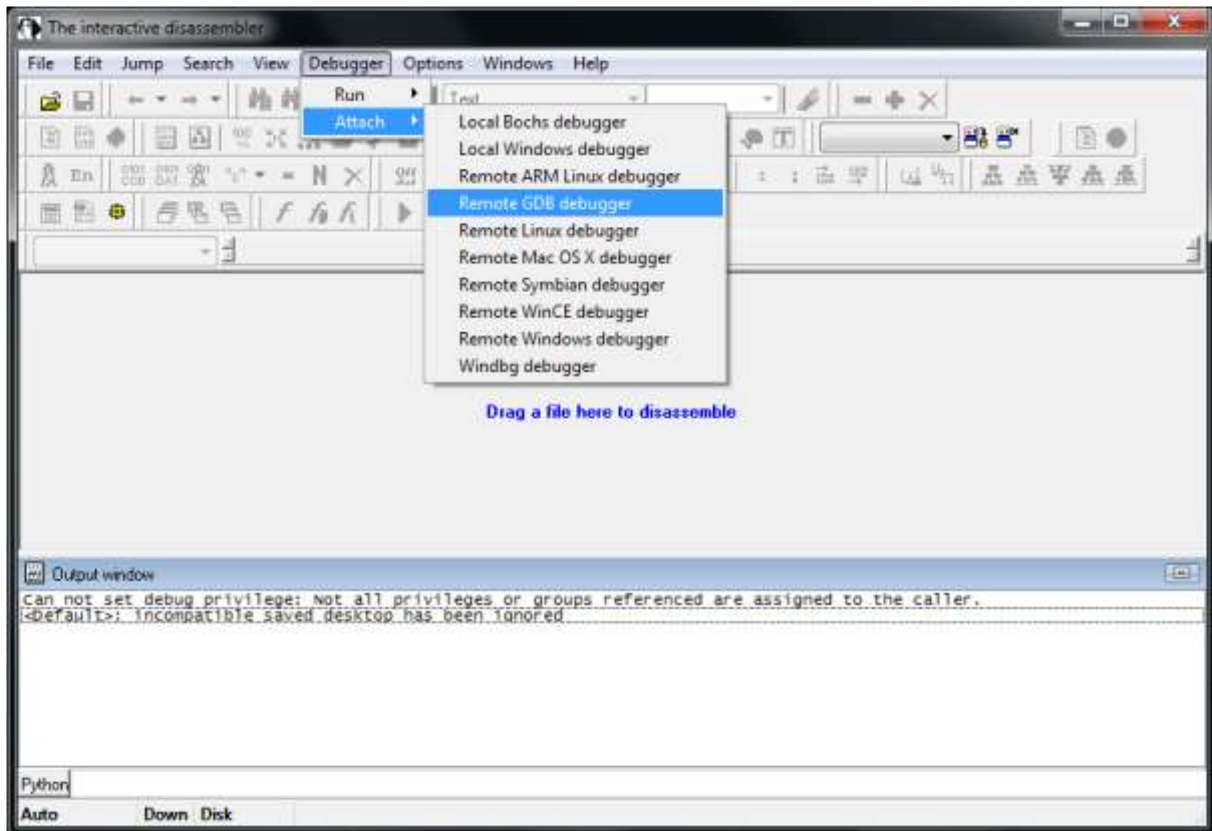
```
# ps
ps
USER      PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root       1     0     296    204   c009a694 0000c93c S /init
(…)
root       206   37    728    324   c003d444 afe0d6ac S /system/bin/sh
app_26     207   30    102784 20004 ffffffff afe0da04 S com.anssi.secret
root       213   206   868    332   00000000 afe0c7dc R ps

# gdbserver :1234 --attach 207
gdbserver :1234 --attach 207
Attached; pid = 207
Listening on port 1234
```

Pour pouvoir communiquer sur le port TCP/1234 de l'émulateur, il reste à le rendre disponible dans l'hôte :

```
C:\> adb forward tcp:1234 tcp:1234
```

Nous pouvons alors nous attacher à distance au processus. Pour cela nous n'utiliserons pas `gdb` mais un programme compatible et bien plus convivial : IDA Pro.

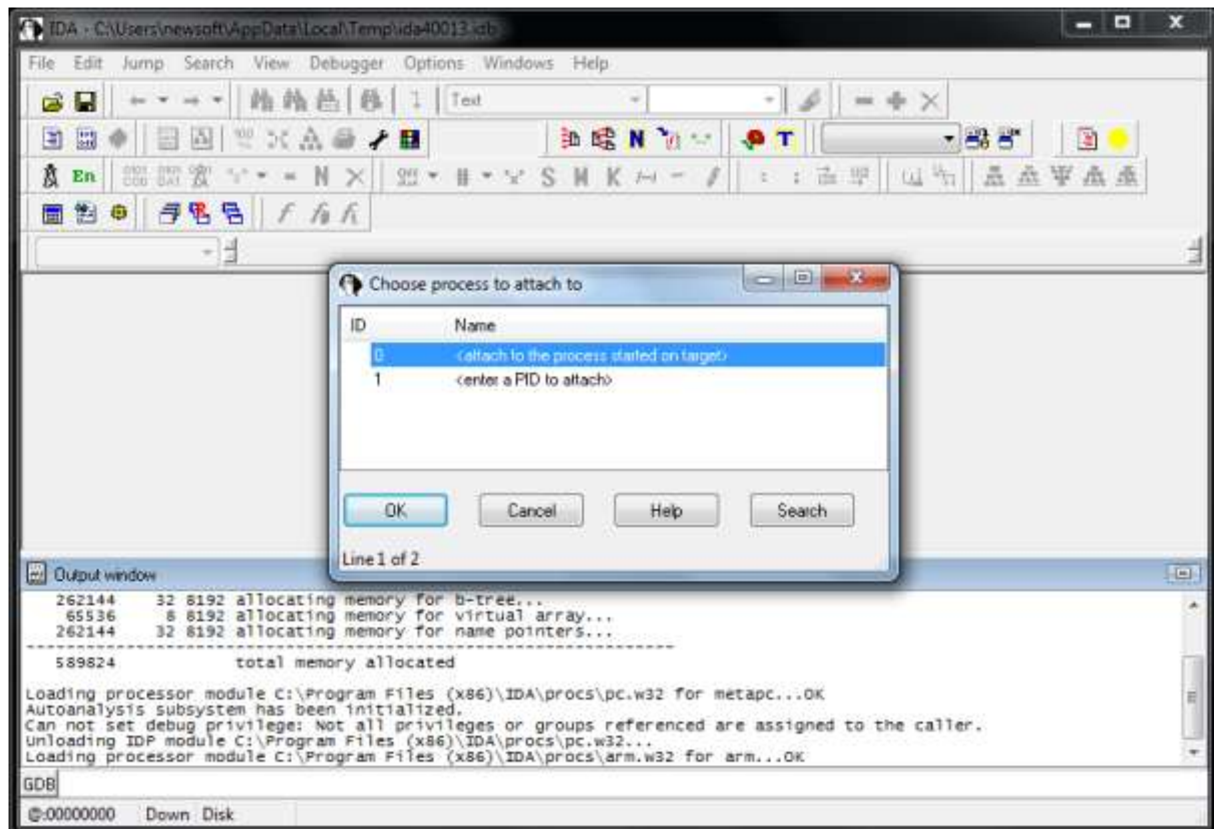
Après avoir lancé une nouvelle instance d'IDA Pro, il faut se rendre dans le menu `Debugger > Attach > Remote GDB Debugger`.



Invocation du débogueur

Dans le sous-menu `Debug options > Set specific options`, il faut spécifier que la cible dispose d'un processeur ARM.

Et c'est parti !



Communication établie côté client

```
# gdbserver :1234 --attach 207
gdbserver :1234 --attach 207
Attached; pid = 207
Listening on port 1234
Remote debugging from host 127.0.0.1
```

Communication établie côté serveur

Limites

A l'usage, il s'avère que la combinaison IDA Pro/gdbserver présente les bogues suivants (dont la cause n'a pas été entièrement élucidée – seules des hypothèses sont données) :

- La fonction *single step* n'est pas disponible. Il semblerait que le noyau Android ne supporte pas l'appel système `ptrace()` sur du code THUMB[23], ce qui est rédhibitoire pour tous les débogueurs basés sur `ptrace()`.
- Il n'est pas possible de *patcher* les instructions en mémoire. Il s'agit peut-être d'un effet de bord de la présence d'un cache d'instructions et d'un cache de données séparés (le protocole gdb ne permettant pas a priori de *flusher* explicitement le cache d'instructions), ou alors d'un effet de bord de QEmu.
- Les points d'arrêt sont supportés. Toutefois il est nécessaire d'interrompre manuellement l'exécution du programme pour « reprendre la main ». IDA Pro n'est pas notifié de l'arrivée à un point d'arrêt.

---

[23] http://www.mail-archive.com/android-kernel@googlegroups.com/msg00778.html

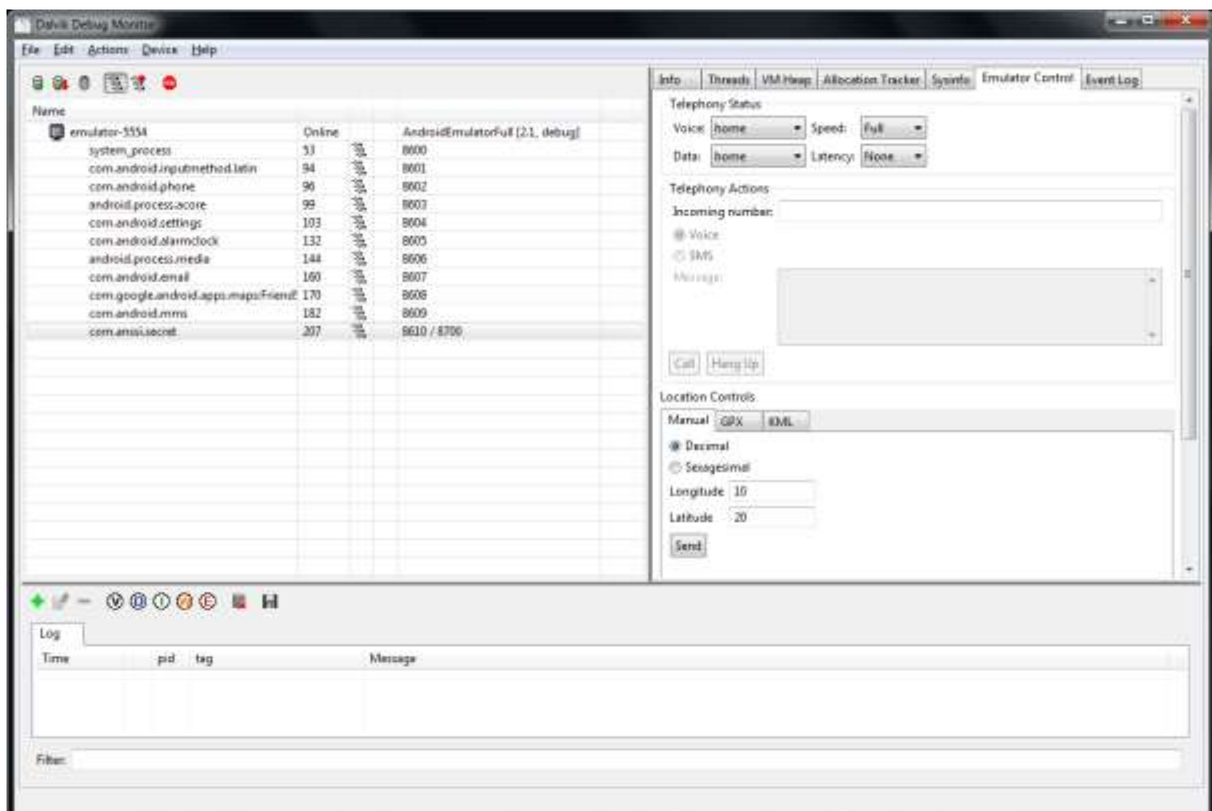
Il n'a pas été déterminé si ces bogues affectaient également la combinaison gdb/gdbserver, toutefois une source orale laisse à penser que oui.

Utilisation de DDMS

Afin de simuler la présence d'un signal GPS pour pouvoir saisir les coordonnées géographiques dans l'interface graphique du logiciel cible, il est possible d'utiliser l'outil « Dalvik Debug Monitor » (DDMS), qui se trouve dans le sous-répertoire suivant du SDK :

```
./tools/ddms.bat
```

Cet outil (extrêmement puissant pour la mise au point d'applications Java) permet entre autre de simuler la présence du téléphone à n'importe quel endroit de la planète.



Interface de l'outil DDMS

Quelles que soient les coordonnées saisies, le logiciel cible indique toujours « 0.0 / 0.0 ».

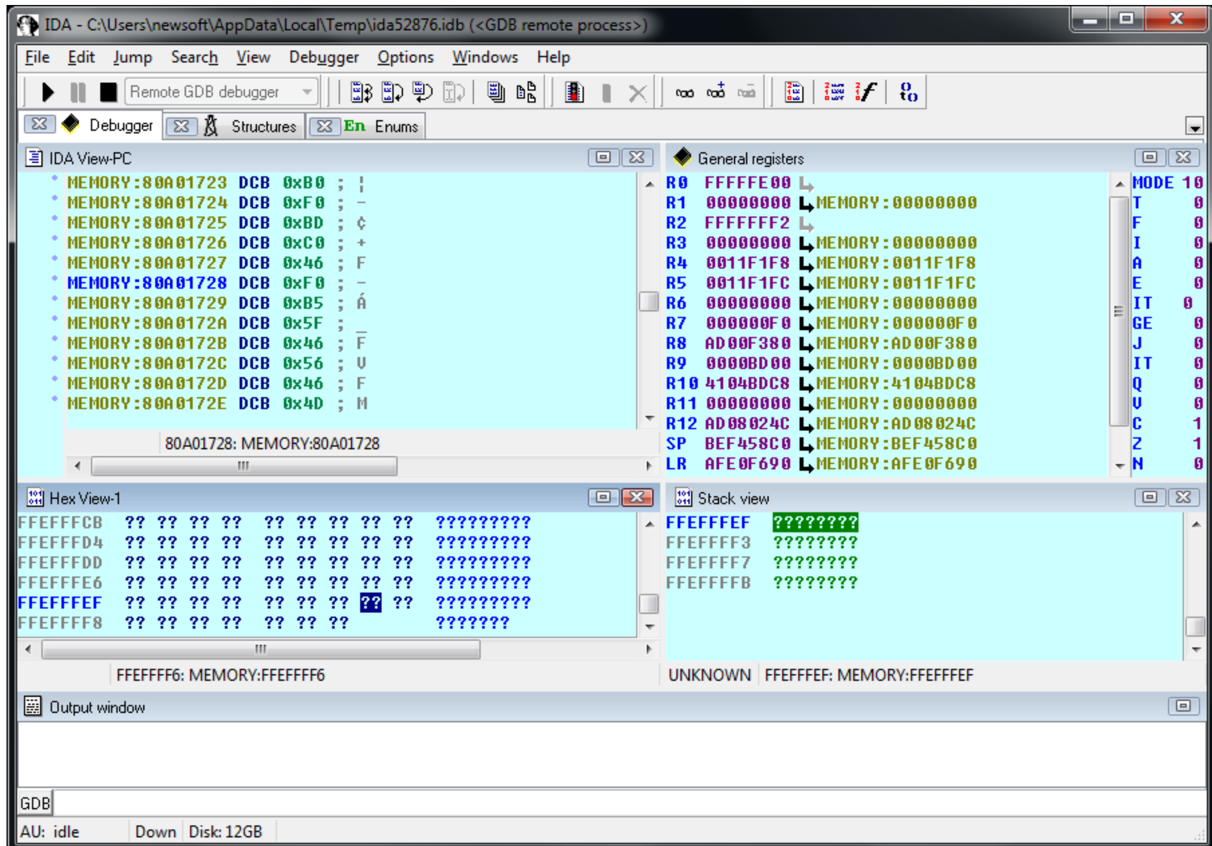
Ceci est sans effet pour notre méthode de résolution, qui injecte directement les « bonnes » coordonnées en mémoire.

Espérons que ce bogue ne soit pas encore lié à IPv6, dont le support semble totalement expérimental[24] ☺

[24] http://groups.google.com/group/android-platform/browse_thread/thread/7e0e37705c8c3d71

Utilisation d’IDA Pro (en tant que débogueur)

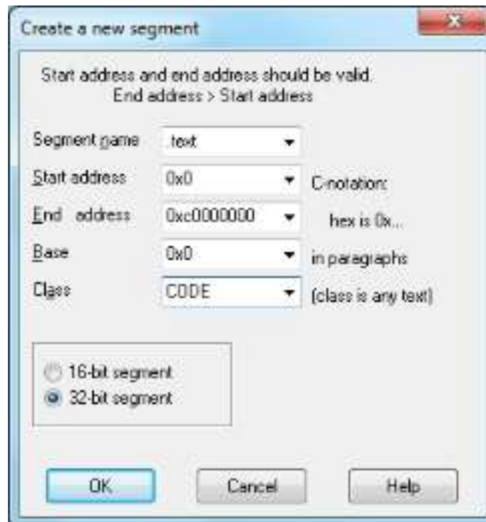
Avant de pouvoir commencer à déboguer avec IDA Pro (mais après s’être attaché au processus cible), il faut fournir quelques informations sur la cible. En effet, IDA Pro n’a aucune connaissance du fonctionnement interne du système Android, et la vue que nous avons du point d’entrée du programme (situé en `0x80a01728`[25]) est plutôt « spartiate ».



Vu du programme au point d’entrée (avant édition)

Pour commencer, créons un nouveau segment nommé « .text », qui va couvrir l’essentiel du code du programme (de l’adresse `0x0` à l’adresse `0xC0000000` pour être large, et après consultation de la carte `/proc/<pid>/maps` pour le processus cible) :

[25] Le placement de la librairie en mémoire est identique d’une exécution à l’autre dans l’émulateur, ce qui facilite la tâche d’analyse.

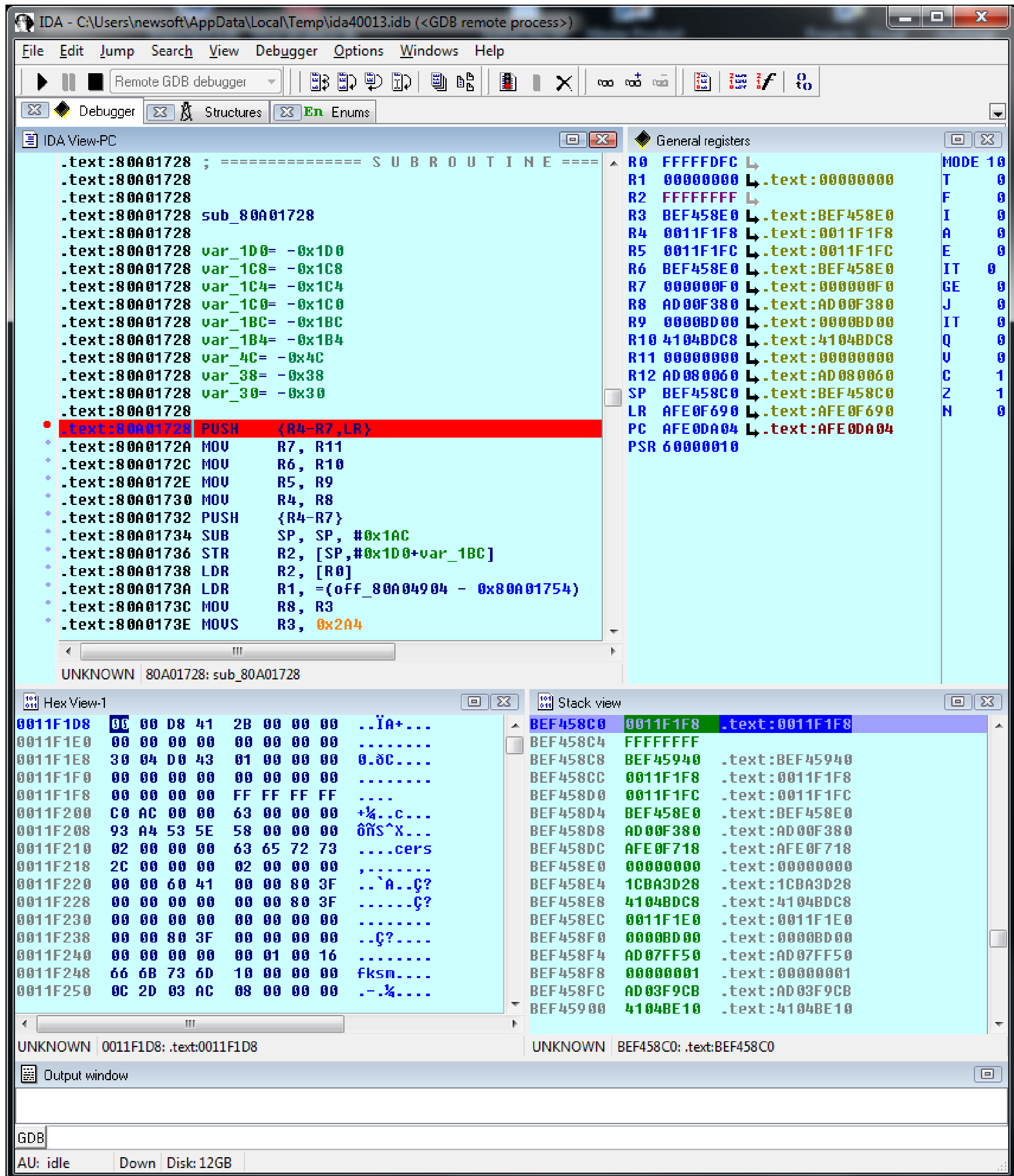


Création d'un nouveau segment

Le type de segment (16 ou 32 bits) n'est pas utile dans notre cas. En effet, le mode THUMB est géré par IDA Pro sous la forme d'un « pseudo-registre » de segment nommé « T » et à qui nous donnons la valeur « 1 » (via le raccourci « Alt-G »), puisque le code cible est essentiellement en mode THUMB.

Modification du pseudo-registre « T »

Il faut ensuite indiquer la présence de code à l'adresse `0x80a01728` (raccourci « C ») et même du point d'entrée d'une fonction (raccourci « P »). Enfin nous plaçons un point d'arrêt (raccourci « F2 »). L'exécution du programme peut reprendre (raccourci « F9 »).



Vu du programme au point d'entrée (après édition)

Première exécution

Le programme cible s'exécute. Les valeurs saisies dans l'interface graphique importent peu. Après confirmation de toutes les valeurs, notre point d'arrêt initial est invoqué, confirmant le bon fonctionnement de l'environnement.

Il est essentiel de placer un point d'arrêt conditionnel à l'adresse `0x80A017E0` (juste après la boucle d'arrondi) afin d'appeler la fonction IDC donnée précédemment[26], qui positionne les valeurs de localisation attendues dans la pile.

Si les localisations sont bonnes, et quel que soit le mot de passe saisi, un résultat positif est obtenu : le fichier `/data/data/com.anssi.secret/files/binaire` est généré. Il est possible de récupérer ce fichier via la commande :

```
C:\> adb pull /data/data/com.anssi.secret/files/binaire binaire
```

Toutefois, en l'absence de mot de passe valide, la clé de déchiffrement RC4 générée est incorrecte, et le résultat obtenu est incohérent.

Un autre élément qui nous reste à déterminer est la fonction de hashage utilisée. Pour se faire, nous positionnons un point d'arrêt à la sortie de la fonction `hash_init()` en `0x80A0055E`.

Contrairement au code exécutable de la librairie (dont l'emplacement est fixe d'une exécution sur l'autre dans l'émulateur), la valeur du registre de pile (`SP`) est « légèrement » aléatoire[27]. Afin de tracer au mieux l'utilisation et l'emplacement des variables de pile, la solution la plus simple et la plus « visuelle[28] » est d'utiliser une feuille Excel.

---

[26] IDA Pro supporte comme condition d'arrêt n'importe quelle expression IDC/IDAPython, y compris l'appel de fonctions définies par l'utilisateur.

[27] A vue de nez, quelques bits au milieu ☺

[28] Par exemple, il est possible d'assigner des couleurs aux cellules du tableau …

	A	B	C	D	E	F
1	**Offset**	**SP value**	**Symbolic value**	**Numeric value**	**Comment / name**	***After init***
2	0	BEF26918			Entry point SP value	
3	4	BEF26914	saved_LR	AD00F1F8		
4	8	BEF26910	saved_R7	41xxxxxx		
5	C	BEF2690C	saved_R6	0		
6	10	BEF26908	saved_R5	4		
7	14	BEF26904	saved_R4	BECBF938		
8	18	BEF26900	saved_R11	0		
9	1C	BEF268FC	saved_R10	41xxxxxx		
10	20	BEF268F8	saved_R9	41xxxxxx		
11	24	BEF268F4	saved_R8	BECBF918		
12	28	BEF268F0				
13	2C	BEF268EC			location bytes	
14	30	BEF268E8	var_30		location bytes	
15	34	BEF268E4		FFA9D6BD	from R/O data, then MVNS	
16	38	BEF268E0	var_38 / ClassSignature	A4BDA4D7	from R/O data, then MVNS	
17	3C	BEF268DC		B8	from R/O data	
18	40	BEF268D8		FC12F215	from R/O data	
19	44	BEF268D4		3F63D4FE	from R/O data	
20	48	BEF268D0		FB04EE17	from R/O data	
21	4C	BEF268CC	var_4C	757D94F4	from R/O data	
22	50	BEF268C8				
23	54	BEF268C4				
24	58	BEF268C0				
25	5C	BEF268BC				
26	60	BEF268B8				
27	64	BEF268B4				
28	68	BEF268B0				
29	6C	BEF268AC			Final output buffer	
30	70	BEF268A8				

Représentation partielle de la pile dans Excel

Excel ne supporte pas nativement l'affichage de valeurs hexadécimales. Toutefois en passant par la formule `=DEC2HEX(HEX2DEC("1234ABCD")`, il est possible d'afficher la valeur `1234ABCD`, tout en bénéficiant des possibilités associées aux valeurs numériques (ex. extension automatique par sélection).

Dans notre feuille, il faut saisir manuellement la valeur du registre `SP` dans la case `B2` à chaque exécution. L'invocation automatique d'Excel (en tant que composant COM) depuis IDAPython est laissée en exercice au lecteur[29].

Lorsque le premier point d'arrêt est atteint (à la sortie de la fonction `hash_init()`), l'état interne du hash (stocké dans la pile) vaut :

```
0x79F023D8 0x708745B8 …
```

[29] http://oreilly.com/catalog/pythonwin32/chapter/ch12.html

La recherche en ligne des constantes 79F023D8 708745B8 donne immédiatement le résultat suivant :

http://www.shabal.com/wp-content/uploads/Shabal.pdf

L'algorithme de hash a été identifié : il s'agit de SHABAL-256.

Crypto Mistake!

L'implémentation de référence semble avoir été utilisée dans libhello-jni.so (c'est d'ailleurs la seule disponible à la date de conception du Challenge).

Un commentaire d'implémentation très important se trouve dans le fichier shabal.h :

```
(…)
/*
* Continue a running hash computation. The state structure is provided.
* The additional input data is a sequence of bits; the "data"
* parameter points to the start of that sequence, and the "databitlen"
* contains the sequence length, expressed in bits.
*
* The bits within a byte are ordered from most significant to least
* significant. The input bit sequence MUST begin with the most
* significant bit of the first byte pointed to by "data". The bit
* sequence length MUST be a multiple of 8 if this call is not the
* last Update() call performed for this hash computation. In other
* words, the input message chunks MUST consist of entire and aligned
* bytes, except for the very last input byte, which may be "partial".
*
* Returned value is SUCCESS (0) on success, or a non-zero error code
* otherwise.
*/
HashReturn Update(hashState *state,
const BitSequence *data, DataLength databitlen);
(…)
```

Il apparait alors que :

- Le paramètre databitlen de cette fonction est le nombre de <u>bits</u> du message d'entrée et non le nombre d'<u>octets</u>. La construction vue dans le pseudocode est donc invalide : elle ne va prendre en compte que 1/8 des bits d'entrée !

```
hash_update( hash, ptr_mdp, strlen(ptr_mdp) );
```

- Le mot de passe n'étant pas la dernière valeur hashée, le nombre de bits pris en compte est arrondi au multiple de 8 immédiatement inférieur.

Une attaque en force brute sur le mot de passe apparait alors comme raisonnable. Nous ne connaissons pas la nature du fichier binaire généré en sortie (fichier ELF, application Java, script shell, etc.), mais nous faisons l'hypothèse raisonnable en première approche qu'il contiendra la chaine « @sstic.org » puisqu'une adresse email doit s'y trouver.

Cette approche est couronnée de succès, puisque qu'en quelques secondes, un candidat est trouvé avec un mot de passe de longueur comprise entre 16 et 23 caractères (dont seuls les 16 premiers bits sont pris en compte), et dont les deux premiers caractères sont « O7 ».

Le fichier `binaire` s'avère être un fichier ELF qui se contente d'afficher l'adresse email gagnante :

```
C:\> file binaire
binaire: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically
linked (uses shared libs), corrupted section header size

C:\> strings binaire
/system/bin/linker
libc.so
printf
__libc_init
_exit
strlen
__exidx_start
__exidx_end
__data_start
_edata
__bss_start
__bss_start__
__bss_end__
__end__
_stack
=+x     I|D
a:`;
Bravo, le challenge est termin
 ! Le mail de validation est : %s
42849d74a8af53aa7a85fc4e956b2d84@sstic.org
```

Conclusion

Le Challenge SSTIC 2010 a été une occasion parfaite pour s'intéresser au système d'exploitation Android. Il fait découvrir plusieurs facettes de ce système, telles que le développement d'applications Java, le développement d'applications natives, la mise au point d'applications et leur analyse en source fermée.

Le Challenge en lui-même s'est avéré intéressant et équilibré, aucune étape n'étant fastidieuse et/ou insurmontable, tout en restant intellectuellement stimulant (le domaine de l'analyse de programmes en source fermée sur un système Android restant largement vierge dans la littérature).

Plusieurs voies sont offertes aux participants, aucune n'étant réellement plus ardue qu'une autre.

On regrettera néanmoins que ce Challenge reste un exercice purement intellectuel. Plusieurs mécanismes clés du système Android (ex. protection du système contre les attaques, gestion des secrets en mémoire, extraction de la mémoire physique sur un téléphone « réel ») n'ont pas été étudiés, ce qui ne permet pas de généraliser les résultats obtenus à des applications dans le monde de la lutte contre les codes malveillants ou du *forensics*.

Merci aux organisateurs et bravo à tous les participants, surtout ceux qui ont conjugué travail en retard, vie de famille pendant les vacances scolaires, et saine émulation entre concurrents !