Solution au challenge SSTIC 2010

Philippe Biondi
EADS Innovation Works
phil(at)secdev.org

10 mai 2010

Table des matières

1 Reconstruction mémoire

À l'aide d'heuristiques sur la forme des entrées de *page directories* ARM et sur la présence des entrées concernant le noyau Linux, nous retrouvons l'ensemble des *page directories* de la mémoire physique.

Chaque page directory correspond à un processus. Nous créons alors pour chaque processus une image de sa mémoire virtuelle.

Cela n'a pas servis pour le challenge, à part pour vérifier que le mot de passe ou des éléments qu'il aurait servi a générer ne semblent pas présents en mémoire.

Nous pouvons ensuite reconstruire la liste chaînée de **task_struct** à partir de la mémoire virtuelle du noyau (qui est de toute façon mappée linéairement dans la mémoire physique).

2 Data carving

À l'aide du programme `zipcarve`, pas encore prêt pour une *release* pour l'instant (mais cela ne saurait tarder), nous extrayons plusieurs fichiers ZIP, qui sont en fait des paquets Android (APK).

Deux d'entre eux sont intéressants, il s'agit de `com.anssi.textviewer.apk` et `com.anssi.secret.apk`.

2.1 Vigenère

Le paquet `com.anssi.textviewer.apk`, s'il est installé et lancé sur un émulateur affiche un message chiffré.

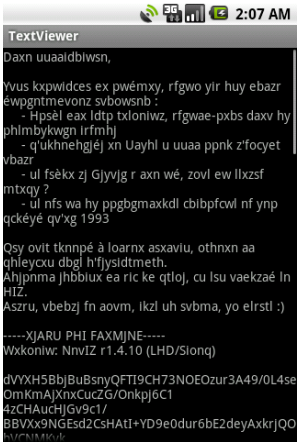

FIG. 1 – Message codé de l'application `textviwer`

Nous retrouvons le message dans le fichier `chiffre.txt` des ressources du paquet.

```
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+96HO54cyHZTGkO3I1QO5dbP2YUPf0V5Z8P/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/Qa59vOfxe4ZdAah7xXBhiFAPVw5fmZ9FOXzZI2
G1s5K5u2rnfZnRdl/KwZZ8gEVcBRIsJJUqmVKgvlMOp5KuxpQwDXNgv/4LyBnyfO
xMaD7gOpVPvxIuexuCQ7SFPoU7Xgqr3FhnlNd+CvN2bjXnl/PZgfAYpxufJg4jNd
8nI+8qvXVKkLBypUuM/zrb8Z/2CWYrXdsuf97OHcvVgtz/KriKPuQXvfSK9pUnDA
DIdqhidda2WMszboUGkd4LYhn06WmE2+QVzqkX/nUOR+ccX5HveRA64b1PfdZm0f
bKQtzo6pT5HGC6U3Fwx2r4dRDaC95+ejCKxXb3Aefnw2n7HIzT3iWPYBtk6oKAT7
vsMOU/45/OvDMnW4GVnAb5OxNRf5BH3VTVsOqI13gNb3cIFSXVtfXOfAB656Lv5U
A73RBF43ALXe7uegYLFrEyHjJwcRVFSjmcGGSbHCcah0MCEOOXP1ToiVVo6xTmDt
SYqEgjlhpR4iqVx1Z+JUsXnk2CqX+u7rXY5DkTZ8xahLuTyXQ7JgZAEfc29vufQl
JPsuefYofQAsQH5ouSWfIN9tZPeqEM50kNQ+jZvAJrNJADvYWpop+8rCHpFBHKLl
NBTZYbeRIwNVUdZCJnkLVpIBUpLCIzXYK4UJJglJyd6MKfguSeWKUaLkHJNQs5bO
JKQCLd1cdwu94jCzwN0WtCvDUmv6m0yzzHvYVka9z/jRz9lqvJXJGYdn+8kRUTZz
Xe3aRrl7+cZOG2nNjmIz1URMlqTatyCaX3ww+rJTVlmUHohMA0xzI7eV4RNpj39P
UpHRNqX0F4tykU9cCWs9/qvtudsi7l2HaoXfbSCAHLTs4NkaK3u6ft3PnPgtoqJt
YQEk90G5QbHv9Nf1J0eb9B5TtZhE3OMp8MZmfywaKfHDCJJWnbW4d1JelEMP0k27
OnSMrZ84G6nT5vW36ZFWjnKZHl6Uyof8LRtrSIbeGROSpY4wLOZavMqX8zmHobo0
Q7UTtrXSLITZ8BX/cF89KBXukj/qMRWJAruiJLyM7iKx/mdBO2uikuH/IfLXEXWB
hsin7IbLoMub4Ejc95ypJKBXoWqJMpMDYZPAEpNYX6X7hXIcWTQOUS4OHABDVNG+
BxkDEH5ZQJU7JRDiotaCuHvykEEbyfZF4WElle91aaLmWXGbk0tWot0eUzVJh/cv
GBKhbbN=
=8CVr
-----EOW ICU JDILJV DAD VUVCL-----
```

Cela ressemble à première vue à du rot13 et à deuxième vue à un chiffre de Vigenère. Le rot13 ne marche pas. Nous essayons le Vigenère à l'aide de la bibliothèque `pygenere`.

Les deux derniers tiers du message étant semble-t-il du base64 chiffré, ils ne vont pas avoir les caractéristiques statistiques nécessaires à la cryptanalyse, aussi nous limitons-nous aux 55 premiers octets pour trouver la clef, puis nous l'utilisons sur la totalité du message.

```
>>> from pygenere import *
>>> chiffre=open("chiffre.txt").read()
>>> v=VigCrack(chiffre[:555])
>>> v.crack_codeword(1,16)
'BTTWFUJHA'
>>> w=Vigenere(chiffre)
```

```
>>> msg=w.decipher('BTTWFUJHA')
```

La variable **msg** contient alors :

```
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
L2M1INN4748DWq0vM1UBylOJjpTda1V4OnPeHcUF2Z0DmUguYriG5X0FIFBXi38i
eELY2pT3LuIYWwkPh5uVOxa/kVsWaIkidRI5Z3eaL/0nhIwDKpxhBeH2e70BA/0c
YAbu+0TEXqoPhW3qH1Mvbc1Y+jZYCPEHpo69yJHeozuwpOpVeqIixcjMXAb1j3gQ
wJsrXU4YsWn2VX1KLsOVWmWQWlMxXLDhi4jwNmsH2CZbJM72pdWubzezEb8DLfVo
Ja+96C054ipAZSNrS3D1WF5wbO2FBTaOB5Q8I/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/Ja59uVmbz4FuTag7eEFcoWTPUd5mqU9LOOsZH2
N1z5O5p2xeyZmYkp/FcQS8gDCjFMOjCJTxtZFmmeMOo5RbbkWnWXMnc/4PtHerfN
eTeY7mFiVOceMpkonCP7ZMTjA7Ozqq3MorgTu+VvM2iqBir/GSgeHFtsawCg4iUk
8rD+8wmQVJrSFtvLnM/yyi8D/2XCPkXczbj97OCimOgsg/RvdQGnQWcmWF9vLgDZ
KPhlnzwdz2DTwuhfNGjk4SCct06NfE2+PCgufD/eNOQ+jjB5CbvKA64a1WmhUsOw
uKPags6kZ5YZC6T3Mdb2m4jIWaB95+lqGFdOu3Admua2i7NZsT3hDWCWzb6hKZA7
cwHOA/45/FoDLuD4KQtRu5OwUYj5WN3MMVrVxM13bTs3vIEZEZolOHfZI656Sz5P
G73IUF43ZSEi7pkxRLEyLcCpApcQCMWestZGRiOGxgy0FCD0VET1OuzOVn6eAqYz
JRqDnqpcvI4bqUe1G+NPyOgk2BxE+y7mDP5WkSG8eecRlMyWX7QkUGVyc29ubmUg
```

```
PGludmFsaWRlQG5vbWRlZG9tYWluZS50bGQ+iGcEExECACcFAkug+8kCGwMFCQCe
NAAGCwkIBwMCBhUIAgkKCwMWAgECHgECF4AACgkQfh6HQwzuRlDOPgCdHIUXw5wU
ADQBSk1gycl94cCydUOAoImWUlc6t0cufYoYUrh9d/eXq9equQENBEug+8kQBADu
Dv3tRqs7+jDJM2eGjlPg1YMScjTzafGvD3np+rIACphAYhhLH0edD7kM4KNoq39W
YkNIGqW0M4acfA9tVWr9/xcxpjjb7l2GhvBahJVAGSAw4IqrD3u6ea3WrKmkhqIa
FUZq90X5JbGc9Uj1E0ks9U5TsGoI3JSg8FZlmfavQwADBQQAihN4w1JdsLQK0q27
FgSLyG84K6iZ5mP36ZEDqrFFYe6Uxvm8PMziLIalNVJYgR4wKVGeqShQ8zlOvfj0
W7LMtqEZPDZQ8UX/bM89RFSabc/qLYDNVxlbJKfT7mFd/dwB02tpryC/OwEXDEDF
cyzg7IaSvQph4Vcc95xwQOWDfPqITwQYEQIADwUCS6D7yQIbDAUJAJ40AAAKCRB+
HodDDO5GUEA7AKDhvaeXaYoyjLLftlQY4WDssi91vgCfWWNio0oCfm0eTgCNc/im
ZBJoifI=
=8IMk
-----END PGP PUBLIC KEY BLOCK-----
```

3 Application `com.anssi.secret`

Cette application est constituée de classes Java et d'une bibliothèque native ARM utilisant JNI pour s'interfacer avec la partie Java.

Nous installons le paquet APK sur un émulateur Android :

```
$ adb install com.anssi.secret.apk
488 KB/s (19805 bytes in 0.039s)
        pkg: /data/local/tmp/com.anssi.secret.apk
Success
```

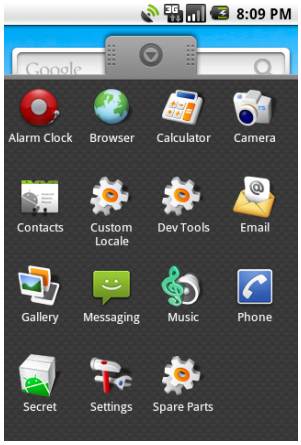

FIG. 2 – Application `secret` installée parmi les autres applications

Une application `Secret` apparaît (fig. 2). Lorsqu'elle est lancée, elle propose de

valider des coordonnées GPS et d'entrer un mot de passe. Cela correspond à l'explication donnée dans l'application `textviewer`.

FIG. 3 – Application `secret` lancée

Nous fournissons donc des coordonnées géographiques à l'émulateur via la console Android :

```
$ telnet localhost 5554
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Android Console: type 'help' for a list of commands
OK
geo fix 1 2
OK
```

et cliquons sur le bouton `J'y suis !` (fig. 4).

FIG. 4 – Validation du premier lieu

Le lieu 1 se valide. Nous recommençons pour les lieux 2, 3 et 4. Le bouton `FINI !` se dégrise. Nous entrons un mot de passe quelconque et le cliquons. L'application se ferme sans aucun message.

Des techniques d'analyse statiques et dynamiques sont employées pour comprendre le fonctionnement de cette application.

3.1 Rétro-ingénierie de la partie Java

Le code Java a été compilé dans `classes.dex`. Nous le désassemblons à l'aide de l'utilitaire `dexdump` fournit dans le SDK d'Android.

Parmi les choses intéressantes en première lecture, nous trouvons une chaîne encodée en base64 et nommée **coincoin** :

```
bmV3c29mdCwgdHUgZXMgaW50ZXJkaXQgZGUgY2hhbGxlbmdlIHBvdXIgc29j
aWFsIGVuZ2luZWVyaW5nIGV4Y2Vzc2lmLg==
```

mais il ne s'agit que d'un message privé.

Nous trouvons également une chaîne de 2548 octets, nommée **programme**, visiblement encodée en hexadécimal.

```
477689b3cb25eba2b9d671cb4a256c07e6bc1902e125970ee14312b2f61976e01a294d2c80a3...
```

3.1.1 Methode onClick

Nous nous intéressons tout d'abord à la méthode **com.anssi.secret.SecretJNI.onClick**. Elle gère tous les clics dans l'application et effectue un aiguillage selon le widget

cliqué. La partie qui nous intéresse est le code appelé lorsque le bouton `FINI !` est cliqué :

Nous avons tout d'abord la récupération du mot de passe qui est référencé par le registre v4 :

```
0069: iget-object v5, v12, Lcom/anssi/secret/SecretJNI;.mdp_edit:Landroid/widget/EditText; //
006b: invoke-virtual {v5}, Landroid/widget/EditText;.getText:()Landroid/text/Editable; // meth
006e: move-result-object v5
006f: invoke-interface {v5}, Landroid/text/Editable;.toString:()Ljava/lang/String; // method@0
0072: move-result-object v4
```

Ensuite, une boucle va multiplier les latitudes et longitudes des 4 lieux par π, et ce tableau fini référencé par le registre v5 :

```
0075: iget v5, v12, Lcom/anssi/secret/SecretJNI;.i:I // field@001a
0077: const/16 v6, #int 8 // #8
0079: if-lt v5, v6, 00b0 // +0037
007b: iget-object v5, v12, Lcom/anssi/secret/SecretJNI;.lieux:[D // field@001b
[...]
00b0: iget-object v5, v12, Lcom/anssi/secret/SecretJNI;.lieux:[D // field@001b
00b2: iget v6, v12, Lcom/anssi/secret/SecretJNI;.i:I // field@001a
00b4: aget-wide v7, v5, v6
00b6: const-wide v9, #double 3.141593 // #400921fb4d12d84a
00bb: mul-double/2addr v7, v9
00bc: aput-wide v7, v5, v6
00be: iget v5, v12, Lcom/anssi/secret/SecretJNI;.i:I // field@001a
00c0: add-int/lit8 v5, v5, #int 1 // #01
00c2: iput v5, v12, Lcom/anssi/secret/SecretJNI;.i:I // field@001a
00c4: goto 0075 // -004f
```

La méthode native **deriverclef**, analysée dans la section suivante, est ensuite invoquée avec le mot de passe et le tableau de **double** en paramètres :

```
007d: invoke-virtual {v12, v4, v5}, Lcom/anssi/secret/SecretJNI;.deriverclef:(Ljava/lang/String
0080: move-result-object v1
```

Le résultat de cet appel est utilisé en entrée de la méthode **dechiffrer** et sa sortie est écrite dans le fichier `binaire`.

```
0081: invoke-direct {v12, v1}, Lcom/anssi/secret/SecretJNI;.dechiffrer:(Ljava/lang/String;)[B
0084: move-result-object v0
0085: const/4 v3, #int 0 // #0
0086: const-string v5, "binaire" // string@004d
0088: const/4 v6, #int 0 // #0
0089: invoke-virtual {v12, v5, v6}, Lcom/anssi/secret/SecretJNI;.openFileOutput:(Ljava/lang/St
008c: move-result-object v3
008d: invoke-virtual {v3, v0}, Ljava/io/FileOutputStream;.write:([B)V // method@0028
0090: invoke-virtual {v3}, Ljava/io/FileOutputStream;.close:()V // method@0027
```

Finalement, un petit message sympathique est affiché :

```
0093: invoke-virtual {v12}, Lcom/anssi/secret/SecretJNI;.getApplicationContext:()Landroid/cont
0096: move-result-object v5
0097: const-string v6, "Bravo ! Va lancer le binaire pour voir si ça a marché !" // string@000
0099: invoke-static {v5, v6, v11}, Landroid/widget/Toast;.makeText:(Landroid/content/Context;L
009c: move-result-object v5
```

```
009d: invoke-virtual {v5}, Landroid/widget/Toast;.show:()V // method@000c
00a0: return-void
```

3.1.2 Methode dechiffrer

Tout d'abord, la chaîne hexadécimale **programme** est récupérée et transformée en tableau d'octets.

```
0000: sget-object v2, Lcom/anssi/secret/SecretJNI;.programme:Ljava/lang/String; // field@001f
0002: invoke-static {v2}, Lcom/anssi/secret/SecretJNI;.hexStringToByteArray:(Ljava/lang/String
0005: move-result-object v0
```

Une instance de la classe **RC4** est créée et initialisée à partir des octets contenus dans la chaîne passée en paramètre :

```
0006: new-instance v2, Lcom/anssi/secret/RC4; // class@0019
0008: invoke-virtual {v5}, Ljava/lang/String;.getBytes:()[B // method@002e
000b: move-result-object v3
000c: invoke-direct {v2, v3}, Lcom/anssi/secret/RC4;.<init>:([B)V // method@0012
[..]
```

Enfin, la méthode **RC4.crypt**() est invoquée avec v0 en paramètre, i.e. le tableau d'octets encodé dans **programme** et sa sortie est retournée.

```
0013: invoke-virtual {v2, v0}, Lcom/anssi/secret/RC4;.crypt:([B)[B // method@0014
0016: move-result-object v1
0017: return-object v1
```

3.1.3 Classe RC4

Le constructeur de la classe **RC4** est une initialisation classique de RC4 à partir d'une clef. Les états internes de RC4 sont stockés dans les attributs **x**, **y** et **state**. Nous notons que les 3072 premiers octets du flot sont jetés, en faisant appel à la méthonde **RC4.getbyte**() analysée ci-après.

```
0017: const/16 v5, #int 3072 // #c00
0019: if-lt v0, v5, 0045 // +002c
001b: return-void
[...]
0045: invoke-virtual {v8}, Lcom/anssi/secret/RC4;.getbyte:()B // method@0016
0048: add-int/lit8 v0, v0, #int 1 // #01
004a: goto 0017 // -0033
```

La méthode **RC4.getbyte**() tire un octet du flot RC4 en métant à jour les états internes (attributs **x**, **y** et **state**)

```
0000: iget v4, v6, Lcom/anssi/secret/RC4;.x:I // field@0011
0002: add-int/lit8 v4, v4, #int 1 // #01
0004: and-int/lit16 v2, v4, #int 255 // #00ff
0006: iget-object v4, v6, Lcom/anssi/secret/RC4;.state:[B // field@0010
0008: aget-byte v0, v4, v2
```

```
000a: iget v4, v6, Lcom/anssi/secret/RC4;.y:I // field@0012
000c: add-int/2addr v4, v0
000d: and-int/lit16 v3, v4, #int 255 // #00ff
000f: iget-object v4, v6, Lcom/anssi/secret/RC4;.state:[B // field@0010
0011: aget-byte v1, v4, v3
0013: iput v2, v6, Lcom/anssi/secret/RC4;.x:I // field@0011
0015: iput v3, v6, Lcom/anssi/secret/RC4;.y:I // field@0012
0017: iget-object v4, v6, Lcom/anssi/secret/RC4;.state:[B // field@0010
0019: aput-byte v0, v4, v3
001b: iget-object v4, v6, Lcom/anssi/secret/RC4;.state:[B // field@0010
001d: aput-byte v1, v4, v2
001f: iget-object v4, v6, Lcom/anssi/secret/RC4;.state:[B // field@0010
0021: add-int v5, v0, v1
0023: and-int/lit16 v5, v5, #int 255 // #00ff
0025: aget-byte v4, v4, v5
0027: return v4
```

La méthode **RC4.com**() est une méthode statique. Elle crée une instance **RC4** à partir de son premier paramètre et appelle la méthode **RC4.cryptself**() sur le second.

```
0000: new-instance v0, Lcom/anssi/secret/RC4; // class@0019
0002: invoke-direct {v0, v1}, Lcom/anssi/secret/RC4;.<init>:([B)V // method@0012
0005: invoke-virtual {v0, v2}, Lcom/anssi/secret/RC4;.cryptself:([B)V // method@0015
0008: return-void
```

La méthode **RC4.cryptself**() prend un tableau d'octets et chiffre en place chaque octet en appliquant un **XOR** avec des octets du flot RC4 donnés par **RC4.getbyte**()

```
0000: const/4 v0, #int 0 // #0
0001: array-length v1, v4
0002: if-lt v0, v1, 0005 // +0003
0004: return-void
0005: aget-byte v1, v4, v0
0007: invoke-virtual {v3}, Lcom/anssi/secret/RC4;.getbyte:()B // method@0016
000a: move-result v2
000b: and-int/lit16 v2, v2, #int 255 // #00ff
000d: xor-int/2addr v1, v2
000e: int-to-byte v1, v1
000f: aput-byte v1, v4, v0
0011: add-int/lit8 v0, v0, #int 1 // #01
0013: goto 0001 // -0012
```

La méthode **RC4.crypt**() chiffre un tableau d'octets et place le résultat dans un autre tableau :

```
0000: array-length v2, v5
0001: new-array v1, v2, [B // class@0028
0003: const/4 v0, #int 0 // #0
0004: array-length v2, v5
0005: if-lt v0, v2, 0008 // +0003
0007: return-object v1
0008: aget-byte v2, v5, v0
000a: invoke-virtual {v4}, Lcom/anssi/secret/RC4;.getbyte:()B // method@0016
000d: move-result v3
000e: xor-int/2addr v2, v3
```

```
000f: and-int/lit16 v2, v2, #int 255 // #00ff
0011: int-to-byte v2, v2
0012: aput-byte v2, v1, v0
0014: add-int/lit8 v0, v0, #int 1 // #01
0016: goto 0004 // -0012
```

3.2 Rétro-ingénierie de la partie native

Le point d'entrée de la bibliothèque semble être **Java_com_anssi_secret_SecretJNI_deriverclef** . C'est en effet la seule fonction a avoir un nom intelligible, et c'est surtout une fonction appelée par le code Java dans la méthode **onClick**()

`objdump` est utilisé pour produire une sortie assembleur de `libhello-jni.so` et `emacs` pour annoter cette sortie.

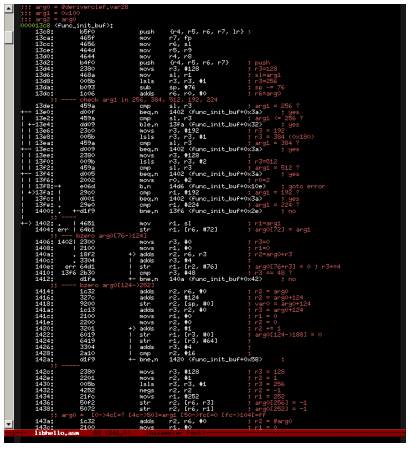

FIG. 5 – EMACS en mode IDA

D'après la convention d'appel ARM, nous trouvons les 4 premiers paramètres de la fonction dans les 4 premiers registres, et les suivants sur la pile.

La documentation JNI nous apprend que le premier paramètre est un pointeur sur une structure **JNIEnv** et la rétro-ingénierie Java nous confirmera que le mot de passe est donné dans r2 et les coordonnées géographiques multipliées par π dans r3.

Nous nous intéressons d'abord au flot de contrôle en repérant les instructions de branchement. Nous avons plusieurs types de branchements : les boucles locales, les appels à des sous-fonctions, les appels à des fonctions de la PLT et des branchements indirects via un registre (r3 ou r4).

3.2.1 Procedure Linkage Table

Les appels à la PLT sont rapidement résolus : chaque stub utilise un mot de la GOT dont on connait la signification :

```
$ objdump -R libhello-jni.so

libhello-jni.so:     file format elf32-little

DYNAMIC RELOCATION RECORDS
OFFSET   TYPE              VALUE
00002944 UNKNOWN           *ABS*
[...]
00002808 UNKNOWN           __cxa_call_unexpected
00002940 UNKNOWN           __gnu_Unwind_Find_exidx
00004910 UNKNOWN           __cxa_begin_cleanup
00004914 UNKNOWN           memcpy
00004918 UNKNOWN           abort
0000491c UNKNOWN           __cxa_type_match
00004920 UNKNOWN           memset
00004924 UNKNOWN           __gnu_Unwind_Find_exidx
00004928 UNKNOWN           sprintf
0000492c UNKNOWN           strlen
```

3.2.2 Java Native Interface

Les appels indirects sont plus compliqués à résoudre.

Il faut d'abord connaître les structures principales sur lesquelles repose JNI, qui sont déclarées dans le fichier `include/jni.h`. **JNIEnv** contient un pointeur sur **JNINativeInterface** et les deux sont principalement composées de pointeurs de fonctions.

```
struct _JNIEnv {
    const struct JNINativeInterface* functions;
#if defined(__cplusplus)
    jint GetVersion(){ return functions->GetVersion(this); }
    jclass DefineClass(const char *name, jobject loader, const jbyte* buf, jsize bufLen)
           { return functions->DefineClass(this, name, loader, buf, bufLen); }
[...]
};
```

```
struct JNINativeInterface {
    void*       reserved0;
[...]
    jint        (*GetVersion)(JNIEnv *);
    jclass      (*DefineClass)(JNIEnv*, const char*, jobject, const jbyte*, jsize);
    jclass      (*FindClass)(JNIEnv*, const char*);
[...]
```

```
    jsize       (*GetStringUTFLength)(JNIEnv*, jstring);
[...]
};
```

Le premier appel se présente ainsi

```
1736: str       r2, [sp, #20]    ; var20 = passwd
1738: ldr       r2, [r0, #0]     ; r2 = JNIEnv[0] = JNINativeInterface
[...]
173e: movs      r3, #169         ; r3 = 169
1740: lsls      r3, r3, #2       ; r3 = 4*169
1742: ldr       r3, [r2, r3]     ; r3 = JNINativeInterface[169]
                                 ;    = GetStringUTFChars
1744: mov       r9, r1           ; r9 = 0x31b0
1746: movs      r2, #0           ; r2 = 0
1748: ldr       r1, [sp, #20]    ; r1 = var20 = passwd
174a: adds      r7, r0, #0       ; r7 = JNIEnv
174c: blx       r3               ; r0 = GetStringUTFChars(JNIEnv, passwd, 0)
```

Tout d'abord, r2, pointant sur l'object java contenant le mot de passe, est sauvegardé dans la variable locale située 20 octets au dessus du pointeur de pile.

Sachant que r0 contient l'adresse de la structure **JNIEnv** fournie par l'appelant, nous transferons le mot pointé par celle-ci dans r2. D'après la définition de la structure **JNIEnv**, il s'agit de l'adresse de la structure **JNINativeInterface**.

La valeur 4×169 est ensuite chargée dans r3 pour pouvoir récupérer la $169^{\text{ème}}$ entrée de structure **JNINativeInterface**, se trouvant à l'adresse r2+r3. Cette adresse est mise dans r3 et y restera jusqu'à l'appel **blx r3**.

D'après `jni.h`, la $169^{\text{ème}}$ entrée de **JNINativeInterface** est un pointeur sur la fonction

```
const char* (*GetStringUTFChars)(JNIEnv*, jstring, jboolean*);
```

Juste avant l'appel, r2 est mis à 0, r1 récupère l'adresse de l'object Java contenant le mot de passe et r0 pointe toujours sur la structure **JNIEnv**.

Nous avons donc l'appel suivant, dont le retour sera dans r0.

```
GetStringUTFChars(JNIEnv, java_password, 0);
```

3.2.3 Déroulement complet de la fonction

1. décapsulation du mot de passe en **char * passwd** (**GetStringUTFChars**())
2. copie de 17 octets de la section `.data` dans **buf1**
3. initialisation d'un buffer de travail de 256 octets **buf2**
4. création d'un **ByteArray key** (**NewByteArray**())
5. décapsulation des coordonnées géographiques en **double**[8] (**GetDoubleArrayElements**())
6. transformation du **double**[8] en **char**[8] **geo** par arrondi et modulo 256
7. mélange de **buf2** avec **passwd** et **geo**

8. initialisation d'un buffer **buf3** de 32 octets à partir de **buf2**
9. copie de **buf3** dans **key** (**SetByteArrayRegion**())
10. initialisation d'un buffer **buf4** de 20 octets comme étant **buf1**[0:17]+(**buf1**[0:4]^"1,Y/") ^ **geo**+**geo**+**geo**[:4]
11. récupération d'une référence sur la classe Java dont le nom est dans **buf4** (**FindClass**())
12. **buf4**[3] = 0 et récupération de l'index de méthode dont le nom est dans **buf4** (**GetStaticMethodID**())
13. appel de la méthode statique avec **key** deux fois en paramètre (**CallStaticVoidMethod**())
14. appel de la méthode 0 de **key**, qui décapsule les 32 octets dans **buf5**
15. oubli des éventuelles exceptions Java levées (**ExceptionClear**())
16. conversion de **buf5** en hexadécimal à l'aide de **sprintf**("%02x") dans **buf6**
17. libération de l'objet UTF passé en paramètre (**ReleaseStringUTFChars**())
18. création d'un objet UTF à partir de **buf6** et retour de cet objet

3.2.4 Coordonnées géographiques

Les coordonnées géographiques transformées en tableau de 8 octets. En étudiant les valeurs de ce tableau en fonction des coordonnées géographiques fournies, nous observons que chaque coordonnée (latitude ou longitude) n'a d'influence que sur 1 seul octet, et que la valeur de cet octet correspont effectivement à l'arrondi de la valeur entrée multiplié par π. Il n'est donc pas nécessaire d'analyser les fonctions de transformation. Le code Python suivant reproduit son comportement :

```
tab=[]
for lng,lat in longlat:
    lng *= math.pi
    lat *= math.pi
    tab += [int(round(lat))%256, int(round(lng))%256]
```

Ce tableau est utilisé comme clef XOR pour dévoiler le nom d'une classe Java puisque le résultat est passé à **FindClass**(). Il serait logique que cette classe fasse partie du package et donc que son chemin commence par `com/anssi/`. Puisque le tableau ne fait que 8 octets et est utilisé plusieurs fois, nous pouvons retrouver ces 8 octets à partir de notre hypothèse et la vérifier sur les 8 octets suivants.

```
DATA0 = '\xf4\x94}u\x17\xee\x04\xfb\xfe\xd4c?\x15\xf2\x12\xfc\xb8'
name = 'com/anss'

def strxor(a,b):
    return "".join(chr(ord(a[i%len(a)])^ord(b[i%len(b)]))
                   for i in range(max(len(a),len(b))))


geo = strxor("com/anss",DATA0[:8])
print name+strxor(geo,DATA0[8:])+strxor("com/","1,Y/")
print geo.encode("hex")
```

Nous obtenons `com/anssi/secret/RC4`, ce qui confirme l'hypothèse. Les coordonnées géographiques donnent donc `97fb105a76807788`.

Nous avons donc l'arrondi entier modulo 256 des latitudes et longitudes multipliées par π. Nous pouvons donc en déduire des longitudes et latitudes suffisantes pour obtenir cette clef. Ces coordonnées ne sont pas forcément celles des réponses aux énigmes trouvées précédemment.

```
r = []
for c in geo:
    i = ord(c)
    if i > 0xa0:
        i |= -1<<8
    r.append( round(i/3.141592,1) )

for long,lat in zip(r[1::2],r[::2]):
    print "geo fix %.1f %.1f" % (long,lat)
```

Ce qui nous donne les coordonnées suivantes, que nous testons en soumettant les 4 commandes et en cliquant sur `J'y suis !` entre chaque.

```
geo fix -1.6 48.1
geo fix 28.6 5.1
geo fix 40.7 37.6
geo fix 43.3 37.9
```

Nous rentrons un mot de passe quelconque et validons. L'application ne plante pas et un message s'affiche :

> Bravo ! Va lancer le binaire pour voir si ça a marché !

FIG. 6 – Coordonnées géographiques acceptées

Nous trouvons effectivement un fichier `binaire` dans le répertoire `/data/data/com.anssi.secret/files` de 1274 octets, mais dont le contenu n'a pas de sens.

```
$ adb shell
# ls -l /data/data/com.anssi.secret/files
-rw-rw---- app_24   app_24       1274 2008-09-10 11:12 binaire
```

4 The big picture

Nous pouvons maintenant établir la façon dont les lieux et le mot de passe vont produire un binaire. Cela est représenté sur la figure 7. La seule partie manquante à ce stade est la génération de la clef 1 à partir des lieux et du mot de passe.

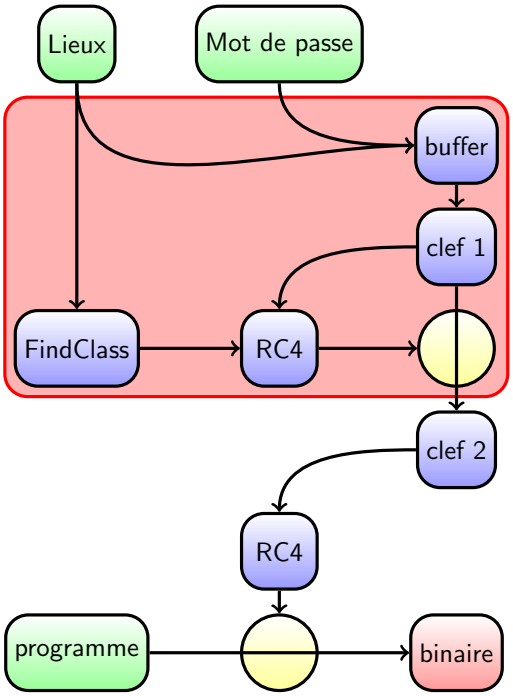

FIG. 7 – The big picture

5 Récupération du mot de passe

Nous avons constaté que beaucoup de mots de passe différents donnaient la même clef. Nous décidons de faire une petite étude statistique au lieu de se lancer dans l'analyse complete de la fonction de mélange du mot de passe.

5.1 Instrumentation

Nous lançons l'application et y attachons un stub gdb :

```
$ adb forward tcp:1234 tcp:1234
$ adb shell
# ps
[...]
app_24    247   30    102812 20408 ffffffff afe0da04 S com.anssi.secret
# gdbserver :1234 --attach 247
Attached; pid = 247
Listening on port 1234
```

Nous créons un client pour stub GDB (voir annexe B) et l'utilisons pour contrôler le déroulement du programme.

```
START=0x80a017e0
STOP=0x80a01810
STACK_PASSWD_AD = 8 # sp+8
STACK_KEY = 356     # sp+356
STACK_PREKEY = 28   # sp+28
gdb=Android_Stub()

gdb.breakpoint(START|1)
gdb.breakpoint(STOP|1)
gdb.cont()
gdb.remove_breakpoint(START|1)

regs = gdb.getregs()
sp = regs.sp
PASSWD, = struct.unpack("I", gdb.readmem(regs.sp+STACK_PASSWD_AD, 4))
KEY=sp+STACK_KEY
PREKEY=sp+STACK_PREKEY

init_prekey=gdb.readmem(PREKEY, 256)

keysort=defaultdict(lambda : [])

for pwd in passwordgen():
    gdb.writemem(PASSWD, pwd+"\x00")
    gdb.cont()
    key = gdb.readmem(KEY, 32)
    keysort[key].append(pwd)
    print "**** [%s] ****" % pwd
    hexdump(key)
    gdb.setregs(regs)
    gdb.writemem(PREKEY, init_prekey)
```

Ce client va se connecter au stub gdb et placer deux points d'arrêt en 0x80a017e0 et en 0x80a01810. Ces deux adresses encadrent le cœur du code créant la clef. Il va ensuite débloquer l'application.

Nous rentrons les 4 lieux trouvés précédemment, un mot de passe quelconque et cliquons sur `FINI !`. L'application se bloque alors sur le premier point d'arrêt.

Le client va alors récupérer l'ensemble des registres et l'état du buffer de travail. Il entre ensuite dans une boucle où un grand nombre de mots de passes vont être essayés. Chaque mot de passe va tout à tour être implanté dans la pile à l'endroit où ce dernier va être recherché. L'application est relancée et rencontre deuxième point d'arrêt. À ce niveau de l'exécution, la clef a été calculée à partir du mot de passe implanté. Elle est extraite et stockée. Les registres et le buffer de travail sont alors restaurés et le mot de passe suivant peut être implanté.

Nous pouvons ainsi tester plusieurs centaines de mots de passe et connaître, pour une clef générée, l'ensemble des mots de passe qui en sont à l'origine.

5.2 Étude statistique

En soumettant des mots de passe générés aléatoirement, on se rend compte qu'une seule clef a été générée par un grand nombre de mots de passe. Leur seul point commun est leur petite taille, entre 1 et 7 caractères. La taille du mot de passe semblant influer sur la clef, nous essayons des mots de passe aléatoires d'une taille fixée et analysons leur répartition par rapport aux clefs générées.

Pour les mots de passe entre 1 et 7 caractères, une seule et même clef est toujours générée :

```
78e765d316db7e2377be6f3598b38089edc2842ab408c39bf6795e6a4f8cd7a0
    0HfFjE, 0mt, 1, 6Pr, 7324, EdGdlin, ImHV0v, Jn, L, NCK9nj4,
    XYFkHRI, bn3R, c, ca, h6Zw, hJg, i1, jAqNLy, jchC, kQKLxxh,
    ms3Bx, o, pNiN, pdpd0tz, qth, uGR1, z9p,
```

Entre 8 et 15 caractères, plusieurs clefs sont générées. Les mots de passe générant la même clef commencent toujours par la même lettre.

```
671895815c037f80eba2b33b755cc8237984175f8be53b6f8db2548a2171ab6f
    dJI5DY6XFcpi4ye, dieJgbbGAdo, dl3BCWSTKl, dJpCloPEtkFp
041867d410c716fc8919e6f6e662a6aa983a276e3f11035728ebf7923f43840a
    bEqpdCuSpmlQ, byrm6SFt6NA5CVJ, bUSCAzGUt, b93iq7aDw
0782921cafe4c24bc4565826f0365fd41094d12f127f55583eeca00151e44690
    acu4kwP5Fqo1, ay1lzvsBane, aRjLkE2aqVdmh, adqoJ4r0OtZs
2f679d4586cf08dc6902ac2eed5c6e8ca163199ca5ed8ee0fbe162da79a81214
    cyqcD163us, cXk3Hg36Gxqwask, cDJsvvjaK3MI, c4ua2onlgzlbRC
```

Entre 16 et 23 caractères, seuls les 2 premiers caractères du mot de passe influent sur la clef.

```
a2c53b24c43fb69d7763b6fdab5f9105c8d01107ad51259ecc8f51f632c3f9ab
    bdkA5NHnW4aQkIZhAy6, bdqteoVK4aOhgCepjTV, bdK9dq6nL7R5o5OKaJuAY,
    bdypqqMkv16q3i1maE, bdpRfjJ4W2CQUcoYGE0M, bdMvPM25JusfIUaW1fKx
d785b8e1a621d34eed9e0be479bf348c1393132b4a3f5ccf9a52d9fa47d0353e
```

```
    bckKXz4TjMy7Eqy9jU, bclxG1YFLDRVvaID5Pe9T, bcS197Gp8NpVF5jyds,
    bcdkmDcox3EvcrHt, bcWBZUY9WYmZ9rEanF9, bcUb1TiIQMGFTQlgN
96202b439c3010e759a98a0c04954071258977b1d31de950ac02e8b5a2185477
    adRWn1sKS2Y0Tb7otIm, adxlGjFo6N3rJc9Y2ib, ad7Na8NEJQq3EGOdLT,
    adMmGFmeIeUvGxuN, adoSe8r9LKEq1tY4LJuNZ, advKaHHH7V6xRikXyC
ed60c4994430c3d86fc58437dad8120a90f689f29c8e14768d9f695900f991b3
    acyXbFnc4TBhNmmRb5Zst, acOH6F54NutLCXQeGB, acx6fNX3LAdN9Zr2,
    acGWsMTtTklLv1rYiTo, accVgntinSYYmEkcNI, ac5VXvTT9wn8cMRPxkyp
0a303a0a77e366f5329a48ee5e67ed7c077d019b64cb2291e5acfc3f7321d10d
    baVKcge16gFBFriyL, baVJjIlhC6AgKm5Tas17, baoOwvTsj3jAZceDD,
    baNmPtDUKseaC9hic, baTEXoTQq81oXKtR, bars7AZ6D5hzK9IC
acbcf32b2c5c39d0f756be71fb20eb255257185de14155b092657a83786bc136
    bbWFrtsu4s43LTj6Fkz, bbkm6MQn8YZF53A7S, bb9wvHcJKU3GYNCi1wIeR,
    bbtbzij3Hejs7rzYEC, bbnDe0UNjpe2rkzyp6, bbdenBa3Naive60YaETsm,
882089fd10bdc4d9a9fa527c294b76d4170820371aecb5f987727bf9fee00857
    abW6xsSUP4Q4EyQk, abZ37AkQZfJIxngT52jV, abEvBdfwAN334yQBP0XfE,
    ab1X2pZIqL7z9vaDMr, abHAtAkO1dz1cZgdRGB1, ab72I2ZFb4seSkNTWX7T,
66afdf37d3b10056d5df8bf7935a3a2b76e1958726256ddd9ce28b3672d0353a
    aajFcm5nc90Mzq9bdmMtZ, aaizW2AAACj8j07LrNr7i, aaiS5A5ISmD2WFBXY0a,
    aaW8Ce6lpCm2hmOr, aairM1fa66gPhhQQRaU, aapW4TxtCxuKaFeI8SdNy,
```

Étant donné un mot de passe **password**, il apparait donc que la clef finale ne dépend que de **password**[:**len**(**password**)/8].

La force brute devient donc totalement accessible.

5.3 Recherche de la clef

Nous modifions donc notre client de stub pour soumettre des mots de passe parcourant l'espace des clefs possibles pour des mots de passe de 1 à 23 caractères.

```
def passwordgen():
    yield "x"
    for a in range(32,127):
        yield chr(a)+"x"*7
    for a in range(32,127):
        for b in range(32,127):
            yield chr(a)+chr(b)+"x"*14
```

Nous obtenons donc 9121 clefs que nous utilisons pour faire 9121 tentatives de déchiffrement grâce au programme obtenu lors de l'analyse du Java, et disponible en annexe A.

Le message sous-entendant que le résultat est un binaire à lancer, nous recherchons la chaîne `ELF` présente dans l'entête d'un binaire ELF.

```
$ grep ELF *
Binary file 527682ae73ceb92e799c9c37c722e6fe9e718be04d115f6415020f0722ba4773 matches
```

```
Binary file 7df2ad74cdebf18876c6d6619d98513ebfa9b7dd4f50f3465bb2532eadd9f664 matches
Binary file 8d14b505adfc111e6cf420b19088b01d93321e93e31f7893cb4ad401560315d9 matches
$ file 527682ae73ceb92e799c9c37c722e6fe9e718be04d115f6415020f0722ba4773
527682ae73ceb92e799c9c37c722e6fe9e718be04d115f6415020f0722ba4773: data
$ file 7df2ad74cdebf18876c6d6619d98513ebfa9b7dd4f50f3465bb2532eadd9f664
7df2ad74cdebf18876c6d6619d98513ebfa9b7dd4f50f3465bb2532eadd9f664: data
$ file 8d14b505adfc111e6cf420b19088b01d93321e93e31f7893cb4ad401560315d9
8d14b505adfc111e6cf420b19088b01d93321e93e31f7893cb4ad401560315d9: ELF 32-bit LSB executable,
```

La clef `8d14b505adfc111e6cf420b19088b01d93321e93e31f7893cb4ad401560315d9` semble donc être la gagnante. Elle provient d'un mot de passe de plus de 16 caractères commençant par `07`. Nous testons donc `07xxxxxxxxxxxxx`, puis :

```
$ adb shell
# cd /data/data/com.anssi.secret/files
# ls
binaire
# chmod 755 binaire
# ./binaire
Bravo, le challenge est terminé ! Le mail de validation est :
 4284d974a8af53aa7a85fc4e956b2d84@sstic.org
```

Notons que la chaîne présente dans le binaire est `42849d74a8af53aa7a85fc4e956b2d84@sstic.org` n'est pas la bonne adresse car le binaire effectue une petite inversion des $5^{\text{ème}}$ et $6^{\text{ème}}$ caractères.

A Déchiffreur

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

import sys,struct,os

bin="477689b3cb25eba2b9d671cb4a256c07e6bc1902e125970ee14312b2f61976e01a294d2c80a3edc9a08a80047

key = sys.argv[1].decode("hex")

# RC4 state initialization
S=range(256)
j=0
for i in range(256):
    j = (j+S[i]+ord(key[i%len(key)]))%256
    S[i],S[j] = S[j],S[i]
i=j=0
for x in range(3072): # throw away 3072 first bytes of stream
    i = (i+1)%256
    j = (j+S[i])%256
    S[i],S[j] = S[j],S[i]
```

```
# self-crypting key
cryptkey = []
for c in key:
    i = (i+1)%256
    j = (j+S[i])%256
    S[i],S[j] = S[j],S[i]
    k = S[(S[i]+S[j])%256]
    cryptkey.append(chr(k^ord(c)))
cryptkey = "".join(cryptkey)
key= cryptkey.encode("hex")

# RC4 state initialization
S=range(256)
j=0
for i in range(256):
    j = (j+S[i]+ord(key[i%len(key)]))%256
    S[i],S[j] = S[j],S[i]
i=j=0
for x in range(3072): # throw away 3072 first bytes of stream
    i = (i+1)%256
    j = (j+S[i])%256
    S[i],S[j] = S[j],S[i]

# decrypting program
clear=[]
for c in bin.decode("hex"):
    i = (i+1)%256
    j = (j+S[i])%256
    S[i],S[j] = S[j],S[i]
    k = S[(S[i]+S[j])%256]
    clear.append(chr(k^ord(c)))
clear = "".join(clear)

# writing program
fname = "/tmp/out/"+sys.argv[1]
open(fname,"w").write(clear)
os.system("file %s" % fname)
```

B Client pour stub GDB

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

import socket,struct

def gdb_checksum(s):
    s=sum(ord(c) for c in s)
    return "%02x" % (s&0xff)


def make_command(cmd):
    return "$%s#%s" % (cmd, gdb_checksum(cmd))


class GDB_Stub:
    def __init__(self, host="127.0.0.1", port=1234):
        self.s = socket.socket()
```

```
        self.s.connect((host,port))

    def send_cmd(self, cmd):
        self.s.send(make_command(cmd))

    def recv_result(self):
        status = self.s.recv(1)
        if status != '+':
            raise Exception("Communication error with stub (%s)" % status)
        r = ""
        self.s.recv(1)
        while True:
            c = self.s.recv(1)
            if c == "#":
                break
            if c == "*":
                raise NotImplemented("RLE compression in response")
            r += c
        cksum = self.s.recv(2)
        if gdb_checksum(r) != cksum:
            raise Exception("Bad result checksum (%s instead of %s)" %
                            cksum,gdb_checksum(r))
        self.s.send("+")
        return r

    def do_cmd(self, cmd):
        self.send_cmd(cmd)
        ret = self.recv_result()
        if ret == "":
            raise Exception("Remote stub does not implement this command (%r)" % cmd)
        return ret

    def cont(self, pc=None):
        if pc is None:
            r = self.do_cmd("c")
        else:
            r = self.do_cmd("c%08x" % pc)
        return r

    def kill(self):
        return self.do_cmd("k")

    def setreg(self, regnum, val):
        r = self.do_cmd("P%i=%08x" % (regnum, val))

    def getreg(self, regnum,):
        r = self.do_cmd("p%i" % (regnum, val))
        return r.decode("hex")

    def getregs(self):
        r = self.do_cmd("g")
        return ARM_Registers(r)

    def setregs(self, regs):
        r = self.do_cmd("G%s" % regs)

    def readmem(self, addr, sz):
```

```
        r = self.do_cmd("m%08x,%04x" % (addr,sz))
        return r.decode("hex")

    def writemem(self, addr, data):
        return self.do_cmd("M%08x,%04x:%s" %
                           (addr, len(data), data.encode("hex")))
    def breakpoint(self, addr):
        return self.do_cmd("Z0,%08x,1" % addr)

    def step(self, addr=None, sig=None):
        Taddr=""
        Tsig=""
        if addr is not None:
            Taddr = "%08x" % addr
            if sig is not None:
                Taddr = ";"+Taddr
        if sig is not None:
            cmd = "S%02x" % sig
        else:
            cmd = "s"
        cmd += Taddr
        return self.do_cmd(cmd)


class ARM_Registers:
    reg_subst = { "sp":"r13",
                  "lr":"r14",
                  "pc":"r15"}
    def __init__(self, regval):
        self.regval = regval
        self.rv = struct.unpack("%iI"%(len(regval)/8), regval.decode("hex"))
    def __str__(self):
        return self.regval
    def __getattr__(self, attr):
        attr = self.reg_subst.get(attr,attr)
        if attr[:1] == "r":
            try:
                print attr[1:]
                x = int(attr[1:])
            except:
                pass
            else:
                return self.rv[x]
        raise AttributeError(attr)


class Android_Stub(GDB_Stub):
    def __init__(self, *args, **kargs):
        GDB_Stub.__init__(self, *args, **kargs)
        self.bp_addr = {}
    def breakpoint(self, addr):
        if addr & 1: # thumb mode
            addr &= 0xfffffffe
            if addr not in self.bp_addr:
                self.bp_addr[addr] = self.readmem(addr,2)
            self.writemem(addr, "\x01\xde")
        else:
```

```
            raise Exception("ARM BP Not implemented")

    def remove_breakpoint(self, addr):
        if addr & 1: #thumb mode
            addr &= 0xfffffffe
            self.writemem(addr, self.bp_addr[addr])
        else:
            raise Exception("ARM BP Not implemented")
```