CHALLENGE SSTIC 2010

JEAN SIGWALD

INTRODUCTION - DECOUVERTE

On commence le challenge en téléchargeant le fichier concours_sstic_2010, une archive 7zip.

```
$ file concours_sstic_2010
concours_sstic_2010: 7-zip archive data, version 0.3
```

Après décompression on obtient le fichier challv2 de 96 Mo, l'image de la mémoire physique d'un mobile Android. Après quelques recherches avec strings et grep, on récupère des informations intéressantes :

```
$ strings challv2 | grep anssi
Start proc com.anssi.textviewer for activity com.anssi.textviewer/.textviewer: pid=227 uid=10024
gids={1015}
Start proc com.anssi.secret for activity com.anssi.secret/.SecretJNI: pid=233 uid=10025 gids={}
Trying to load lib /data/data/com.anssi.secret/lib/libhello-jni.so 0x43d017f8
Added shared lib /data/data/com.anssi.secret/lib/libhello-jni.so 0x43d017f8
No JNI_OnLoad found in /data/data/com.anssi.secret/lib/libhello-jni.so 0x43d017f8
/data/app/com.anssi.secret.apk
/data/app/com.anssi.textviewer.apk

$ strings challv2 | grep qemu
<5>Kernel command line: qemu=1 console=ttyS0 android.checkjni=1 android.qemud=ttyS1 android.ndns=1
```

On trouve la trace de deux applications : **com.anssi.secret** et **com.anssi.textviewer**, avec leur PIDs respectifs. On suppose que ces applications étaient en cours d'exécution lorsque le dump mémoire a été réalisé. L'application Secret semble utiliser une librairie native : **/data/data/com.anssi.secret/lib/libhello-jni.so**.

On remarque également que le dump a été pris dans l'émulateur qemu, sans doute via la commande **pmemsave** qui permet de sauvegarder dans un fichier le contenu de la mémoire physique de l'appareil émulé.

RECUPERATION DU FRAMEBUFFER

Dans une première approche un peu hasardeuse, j'ai modifié la commande **pmemsave** de qemu pour effectuer l'opération inverse : charger le contenu d'un fichier dans la mémoire du « téléphone virtuel ». L'idée était de pouvoir ensuite utiliser les fonctions de qemu pour la translation des adresses virtuelles en adresses physiques. Finalement cette approche n'a pas été utilisée (j'ai copié-collé les fonctions de translation ARM dans un script Python), mais la manipulation m'a permis de voir le contenu du framebuffer lorsque le dump à été réalisé, et de découvrir l'interface de l'application Secret.

```
$ telnet localhost 4444
QEMU 0.10.50 monitor - type 'help' for more
information
(qemu) stop
(qemu) pmemsave 0 100663296 challv2
```

Figure 1 - Contenu du framebuffer

ANALYSE DE L'IMAGE MEMOIRE ET EXTRACTION DES APK

Le challenge va donc consister à extraire les deux applications de l'ANSSI de l'image mémoire, et trouver les coordonnées GPS et le mot de passe pour l'application Secret.

Android utilise un noyau Linux, on peut appliquer les techniques de forensics mémoire existantes. J'ai écrit un script Python qui retrouve la liste chainée des tâches (=threads) maintenue par le kernel, cherche les 2 applications de l'ANSSI en cours d'exécution (via leur PID ou leur nom), parcourt leur espace mémoire et enfin tente d'extraire les fichiers APK mappés en mémoire (archives ZIP équivalentes aux JARs de Java) en convertissant les adresses virtuelles en adresses physiques.

LISTE DES OBJETS TASK_STRUCT DU KERNEL

Pour reconstruire la liste des tâches, on recherche le premier thread créé au démarrage du kernel (idle thread), sachant qu'il se nomme «**swapper**». Cette chaîne de caractères se situe dans le champ **comm** d'une structure task_struct. On peut donc accéder aux différents champs de cette structure si on connait leur offset. Comme le kernel Linux mappe la mémoire physique à l'adresse virtuelle 0xC0000000, on pourra convertir facilement les pointeurs dans l'espace kernel en adresses physiques (offsets dans le fichier image de la RAM).

```
1.  struct task_struct {
2.          volatile long state;    /* -1 unrunnable, 0 runnable, >0 stopped */
3.          void *stack;
4.          atomic_t usage;
5.          unsigned int flags;     /* per process flags, defined below */
6.          unsigned int ptrace;
7.          int lock_depth;         /* BKL lock depth */
8.
9.          /* ... */
10.
11.         struct list_head tasks;
12.         struct mm_struct *mm, *active_mm;
13.
14.         /* ... */
15.
16.         pid_t pid;
17.         pid_t tgid;
18.
19.         /* ... */
20.
21.         /*
22.          * pointers to (original) parent process, youngest child, younger sibling,
23.          * older sibling, respectively.  (p->father can be replaced with
24.          * p->real_parent->pid)
25.          */
26.
27.         struct task_struct *real_parent; /* real parent process */
28.         struct task_struct *parent; /* recipient of SIGCHLD, wait4() reports */
29.
30.         /*
31.          * children/sibling forms the list of my natural children
32.          */
33.
34.         struct list_head children;      /* list of my children */
35.         struct list_head sibling;       /* linkage in my parent's children list */
36.         struct task_struct *group_leader;       /* threadgroup leader */
37.
38.         /* ... */
39.
40.         char comm[TASK_COMM_LEN]; /* executable name excluding path
41.                                      - access with [gs]et_task_comm (which lock
42.                                        it with task_lock())
43.                                      - initialized normally by flush_old_exec */
44.
45.         /* ... */
46. };
```

On va donc parcourir les listes chainées **children** et **sibling** en partant de la tâche "swapper" pour trouver les deux tâches **com.anssi.textviewer (pid=227)** et **com.anssi.secret (pid=233)**.

RECONSTRUCTION DE LA MEMOIRE VIRTUELLE

La mémoire virtuelle est gérée par le composant mm du noyau Linux. Chaque tâche est associée à une structure **mm_struct**, qui va contenir une liste de plages d'adresses virtuelles allouées (Virtual Memory Areas ou VMAs), ainsi que l'adresse de la table des pages **pgd** (page directory). Ces 2 informations nous permettent de parcourir l'espace mémoire d'un processus (exception faite des pages non présentes en mémoire lors du dump).

Notre script va parcourir les VMAs des 2 processus et chercher celles qui correspondent au mapping mémoire des fichiers APK. Si la VMA correspond à un fichier mappé en mémoire, le champ **vm_file** pointe vers une structure qui décrit le fichier et indique son nom et sa taille.

```
1.  struct mm_struct {
2.          struct vm_area_struct * mmap;           /* list of VMAs */
3.          struct rb_root mm_rb;
4.          struct vm_area_struct * mmap_cache;     /* last find_vma result */
5.
6.          unsigned long (*get_unmapped_area) (struct file *filp,
7.                                  unsigned long addr, unsigned long len,
8.                                  unsigned long pgoff, unsigned long flags);
9.          void (*unmap_area) (struct mm_struct *mm, unsigned long addr);
10.         unsigned long mmap_base;                /* base of mmap area */
11.         unsigned long task_size;                /* size of task vm space */
12.         unsigned long cached_hole_size;         /* if non-zero, the largest hole below free_area_cache */
13.         unsigned long free_area_cache;          /* first hole of size cached_hole_size or larger */
14.         pgd_t * pgd;
15.
16.         /* ... */
17. };
18.
19. /*
20.  * This struct defines a memory VMM memory area. There is one of these
21.  * per VM-area/task.  A VM area is any part of the process virtual memory
22.  * space that has a special rule for the page-fault handlers (ie a shared
23.  * library, the executable area etc).
24.  */
25. struct vm_area_struct {
26.         struct mm_struct * vm_mm;       /* The address space we belong to. */
27.         unsigned long vm_start;         /* Our start address within vm_mm. */
28.         unsigned long vm_end;           /* The first byte after our end address
29.                                            within vm_mm. */
30.
31.         /* linked list of VM areas per task, sorted by address */
32.         struct vm_area_struct *vm_next;
33.
34.         /* ... */
35.
36.         struct file * vm_file;          /* File we map to (can be NULL). */
37.
38.         /* ... */
39. };
```

Pour réaliser la translation adresse virtuelle vers adresse physique on récupère les fonctions **get_phys_addr_v5** et **get_level1_table_address** dans le code source de qemu (target-arm/helper.c), et on les convertit en Python pour les utiliser dans notre script. Pour traduire une adresse virtuelle en adresse physique, il suffira d'appeler la fonction **get_phys_addr_v5** avec l'adresse en question et celle de la table des pages du processus ciblé (task_struct->pgd).

EXTRACTION DES FICHIERS APK, RECUPERATION DES PAGES MANQUANTES

Une fois le script finalisé, on va donc pouvoir parcourir les plages mémoire des deux processus et localiser les fichiers APK mappés en mémoire. On parcourt les pages de vm_start à vm_end, en récupérant les adresses physiques via les fonctions de translation de qemu, et on stocke le résultat dans un fichier.

Champ	Valeur
va_start	0x426f3000
va_end	0x426f8000
vm_file->f_path->dentry.d_name.name	« com.anssi.textviewer.apk »
vm_file-> f_mapping->host->i_size	16484
vm_file-> f_mapping->nr_pages	5

Tableau 1 - Informations sur le mapping du fichier com.anssi.textviewer.apk

Page	Adresse virtuelle	Adresse physique
0	0x426f3000	0x3c28000
1	0x426f4000	0x3c29000
2	0x426f5000	0x3c02000
3	0x426f6000	0x3c5a000
4	0x426f7000	0x3c58000

Tableau 2 – Adresses des pages du fichier com.anssi.textviewer.apk

L'application Textviewer est extraite sans problèmes, mais pour l'application Secret deux pages sont absentes de la table des pages. J'ai résolu ce problème en déterminant empiriquement (pour ne pas dire « à l'arrache ») que les 2 pages manquantes étaient adjacentes aux pages précédentes et suivantes respectivement. Par chance (ou pas), le contenu des pages était valide étant donné que l'archive ZIP résultante est correcte.

Champ	Valeur
va_start	0x426f3000
va_end	0x426f8000
vm_file->f_path->dentry.d_name.name	« com.anssi.secret.apk »
vm_file-> f_mapping->host->i_size	19805
vm_file-> f_mapping->nr_pages	5

Tableau 3 - Informations sur le mapping du fichier com.anssi.secret.apk

Page	Adresse virtuelle	Adresse physique
0	0x426f3000	0xc25000
1	0x426f4000	0xc26000
2	0x426f5000	??? page fault => 0xc27000
3	0x426f6000	??? page fault => 0xe08000
4	0x426f7000	0xe09000

Tableau 4 – Adresses des pages du fichier com.anssi.secret.apk

Une fois les fichiers APK extraits, on peut les décompresser pour voir leur contenu puis les installer dans l'émulateur via les commandes suivantes :

```
$ adb install com.anssi.textviewer.apk
$ adb install com.anssi.secret.apk
```

```
$ sha1sum *.apk
3c1522874c6e21fd5cbc1f9210ee8d0fbfc07405 *com.anssi.secret.apk
ae723f6656264d6d829c362278470559cbd665f4 *com.anssi.textviewer.apk
```

APPLICATION COM.ANSSI.TEXTVIEWER ET DECHIFFREMENT DU TEXTE (VIGENERE)

L'application Textviewer, comme son nom l'indique, affiche juste le fichier texte **/res/raw/chiffre.txt** contenu dans son archive APK.

En observant le texte chiffré, on devine que les premiers mots doivent ressembler à « **Cher participant** » et que l'algorithme utilisé n'est pas une simple substitution (à cause des répétitions « uuaa » au début du second mot). En supposant que l'algorithme utilisé est le chiffre de Vigenère, on tente de déterminer la clé de chiffrement :

Chiffré (x)	Clair (y)	Clé (x – y % 26)
D (3)	C (2)	B (1)
A (0)	H (7)	T (19)
X (23)	E (4)	T (19)
N (13)	R (17)	W (22)
U (20)	P (15)	F (5)
U (20)	A (0)	U (20)
A (0)	R (17)	J (9)
A (0)	T (19)	H (7)
I (8)	I (8)	A (0)
D (3)	C (2)	B (1)
B (1)	I (8)	T (19)
I (8)	P (15)	T (19)
W (22)	A (0)	W (22)
S (18)	N (13)	F (5)
N (13)	T (19)	U (20)

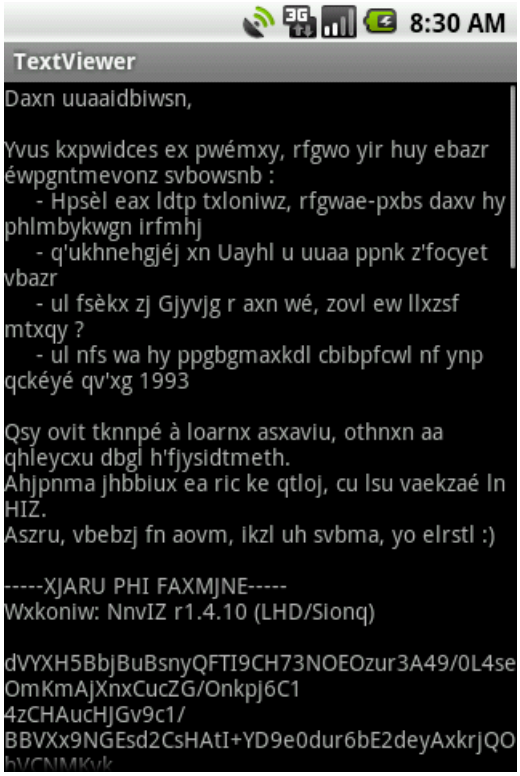

Figure 2 – Application Textviewer

C'est donc bien le chiffre de Vigenère qui est utilisé, et la clé est **BTTWFUJHA**.

Le texte déchiffré nous donne des indications pour la suite du challenge :

```
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/dwB02tpryC/OwEXDEDF
cyzg7IaSvQph4Vcc95xwQOWDfPqITwQYEQIADwUCS6D7yQIbDAUJAJ40AAAKCRB+
HodDDO5GUEA7AKDhvaeXaYoyjLLftlQY4WDssi91vgCfWWNio0oCfm0eTgCNc/im
ZBJoifI=
=8IMk
-----END PGP PUBLIC KEY BLOCK-----
```

Les 2 premiers lieux sont assez simples à deviner : le Coq Gadby à Rennes et le centre spatial Guyanais, mais les 2 autres sont beaucoup plus énigmatiques. Pour le message GPG il nous manque la clé privée pour le déchiffrer. Du coup on laisse de coté ce fichier pour l'instant et on s'attaque à l'application Secret.

	Description	Latitude	Longitude
Lieu 1	Le Coq Gadby – Rennes	48.122990	-1.668516
Lieu 2	Centre spatial Guyanais	5.042850	-52.788418

Tableau 5 – Premières coordonnées GPS

ETUDE DE L'APPLICATION COM.ANSSI.SECRET

Une fois l'application Secret installée et lancée, on entre les 2 coordonnées GPS que l'on connait (avec la commande **geo fix** de l'émulateur), et on essaye le mot de passe « test » (le même que celui du framebuffer). Lorsque l'on clique sur le bouton « Fini », l'application se ferme brutalement. D'après les messages du syslog, le crash a lieu dans la méthode native **deriverclef**, suite à un appel de la fonction **FindClass** avec des caractères invalides dans une chaîne de caractères.

```
pid   tag        Message
230   dalvikvm   +++ not scanning '/system/lib/libwebcore.so' for 'deriverclef' (wrong CL)
230   dalvikvm   +++ not scanning '/system/lib/libmedia_jni.so' for 'deriverclef' (wrong CL)
230   dalvikvm   +++ not scanning '/system/lib/libexif.so' for 'deriverclef' (wrong CL)
230   dalvikvm   JNI WARNING: illegal start byte 0xb4
230   dalvikvm                string: 'com/▯`▯i/se▯`▯/RC4'
230   dalvikvm                in Lcom/anssi/secret/SecretJNI;.deriverclef (Ljava/lang/String;[D)Ljava/lang/String; (FindClass)
230   dalvikvm   "main" prio=5 tid=3 NATIVE
230   dalvikvm     | group="main" sCount=0 dsCount=0 s=N obj=0x4001b268 self=0xbd00
230   dalvikvm     | sysTid=230 nice=0 sched=0/0 cgrp=default handle=-1344001384
230   dalvikvm     at com.anssi.secret.SecretJNI.deriverclef(Native Method)
230   dalvikvm     at com.anssi.secret.SecretJNI.onClick(SecretJNI.java:113)
230   dalvikvm     at android.view.View.performClick(View.java:2364)
230   dalvikvm     at android.view.View.onTouchEvent(View.java:4179)
```

Figure 3 - Syslog de l'émulateur (DDMS dans Eclipse)

La fonction **FindClass** est une fonction **JNI** qui permet au code natif de récupérer une classe Java en indiquant son nom. On voit que le nom passé en paramètre ressemble à **com/anssi/secret/RC4**. On verra en reversant le code natif que les coordonnées GPS sont utilisées pour déchiffrer le nom de cette classe, ce qui explique pourquoi il nous manque des morceaux.

Pour comprendre comment sont utilisées les valeurs entrées dans l'application, on va donc désassembler les classes Java et la librairie dynamique contenant la méthode native.

Android utilise la machine virtuelle Dalvik pour exécuter les applications programmées en Java. Le bytecode Dalvik des classes d'une application est stocké dans le fichier **classes.dex** de l'archive APK.

Plusieurs outils existent pour désassembler les fichiers dex, j'ai utilisé **baksmali**[1]. On obtient un fichier texte .smali avec le bytecode désassemblé pour chacune des classes.

```
$ java -jar baksmali-1.2.2.jar classes.dex
$ ls out/com/anssi/secret
R$attr.smali
R$id.smali
R$layout.smali
R$string.smali
R.smali
RC4.smali
SecretJNI.smali
```

Les classes dont le nom commence par R correspondent aux ressources de l'application (chaînes de caractères, layout de l'interface graphique, etc.) et sont générées automatiquement par les outils de développement. Les 2 classes à analyser sont donc **RC4** et **SecretJNI**.

[1] http://code.google.com/p/smali/

CLASSE RC4

Cette classe est une implémentation standard de l'algorithme RC4 en Java. A noter que le constructeur de la classe élimine les 3072 premiers octets de keystream[2]. La méthode statique « com » permet de chiffrer un tableau d'octets avec une clé passée en paramètre.

Méthode	Description
public RC4(byte[] key)	Constructeur - initialise l'état interne et consomme les 3072 premiers octets de keystream
static public void com(byte[] key, byte[] data)	Instancie un objet RC4, et invoque cryptself(data)
public byte[] crypt(byte[] data)	Renvoie le tableau data chiffré
public void cryptself(byte[] data)	Chiffre le tableau data "sur place"
public byte getbyte()	Renvoie un octet de keystream

Tableau 6 – Méthodes de la classe com.secret.anssi.RC4

```
1.  public class RC4 {
2.      private byte[] state;
3.      int x;
4.      int y;
5.
6.      public RC4(byte[] key) {
7.          state = new byte[256];
8.          x = 0;
9.          y = 0;
10.
11.         //key schedule RC4
12.         //...
13.         //
14.
15.         for(int i=0; i < 3072; i++)
16.         {
17.             this.getbyte();
18.         }
19.     }
20.
21.     static public void com(byte[] data, byte[] key)
22.     {
23.         new RC4(key).cryptself(data);
24.     }
25.
26.     //...
```

[2] http://www.users.zetnet.co.uk/hopwood/crypto/scan/cs.html#RC4-drop

CLASSE SECRETJNI

La classe SecretJNI est la classe principale de l'application. Elle contient deux chaînes de caractères statiques, la première chaine « coincoin » est un message encodé en base64 :

```
$ base64 -d
bmV3c29mdCwgdHUgZXMgaW50ZXJkaXQgZGUgY2hhbGxlbmdlIHBvdXIgc29jaWFsIGVuZ2luZWVyaW5nIGV4Y2Vzc2lmLg==
newsoft, tu es interdit de challenge pour social engineering excessif.
```

Ça n'a pas eu l'air de le dissuader :)

La chaine hexadécimale « programme » est chiffrée en RC4 et sera déchiffrée par l'application en fonction des coordonnées GPS et du mot de passe entrés par l'utilisateur. Le résultat sera stocké dans un fichier à priori exécutable nommé « binaire ».

Le code Java reversé est le suivant :

```
1.  public class SecretJNI extends Activity implements OnClickListener, LocationListener
2.  {
3.      public static String programme = "477689b3cb25eba[....]63f9018f081df0fe3a2";
4.
5.      public native String  deriverclef(String password, double[] gps);
6.
7.      private byte[] dechiffrer(String clef)
8.      {
9.          byte[] chiffre = hexStringToByteArray(programme);
10.
11.         rc4 = new RC4(clef.getBytes());
12.         byte[] dechiffre = rc4.crypt(chiffre);
13.
14.         return dechiffre;
15.     }
16.
17.     public void onClick(View v) {
18.
19.         if (v == finibtn)
20.         {
21.             String mdp = mdp_edit.getText().toString();
22.
23.             for(i=0; i < 8; i++)
24.             {
25.                 lieux[i] *= 3.141593;
26.             }
27.             String clef = deriverclef(mdp, lieux);
28.
29.             byte[] binaire = dechiffrer(clef);
30.
31.             try {
32.                 FileOutputStream fOut = new FileOutputStream("binaire", false);
33.                 fOut.write(binaire);
34.             } catch (Exception e) {
35.                 e.printStackTrace();
36.             }
37.
38.             Toast.makeText(getApplicationContext(),
39.                     "Bravo ! Va lancer le binaire pour voir si ça a marché !",
40.                     1).show();
41.         }
```

On remarque qu'avant d'appeler la fonction de dérivation de clé, les coordonnées GPS sont multipliées par 3.141593.

METHODE NATIVE DE DERIVATION DE CLEF

La méthode de dérivation de la clé de chiffrement RC4 est implémentée dans la librairie native **libhello-jni.so** que l'on retrouve dans le dossier **/lib/armeabi** de l'archive APK. Pour comprendre son fonctionnement je l'ai reversé en statique dans **IDA** et vérifié certains points avec **gdb** dans l'émulateur. De plus j'ai écrit une application Java minimaliste qui me permettait d'appeler la méthode **Java_com_anssi_secret_SecretJNI_deriverclef** avec des paramètres contrôlés, sans avoir besoin de passer par l'interface graphique de l'application Secret.

Avant de commencer à reverser on regarde les exemples du NDK, et on remarque que les méthodes natives possèdent toujours un premier paramètre **env** de type **JNIEnv*** qui sert d'interface avec la JVM. Le type JNIEnv est défini dans l'en-tête **JNI.H** comme un pointeur vers une structure **JNINativeInterface** composée de pointeurs de fonctions. Ces fonctions permettent au code natif de manipuler des objets Java.

La première étape va donc consister à identifier tous les appels de fonctions de type **(*env)->XXX()** afin de pouvoir ensuite identifier les types et le contenu des différentes variables à partir des signatures de ces fonctions. Pour trouver ces appels, on suit le paramètre **env** (dans **R0** au début de la fonction), et à chaque fois qu'une valeur chargée en utilisant ce pointeur est utilisée comme pointeur de fonction (comme paramètre de l'instruction d'appel de procédure **BLX**), on identifie l'offset utilisé pour accéder à la structure. Cet offset nous permettra d'identifier quel pointeur de fonction est appelé. Par exemple, le premier appel de ce type utilise le pointeur situé à l'offset 676 de la structure. Pour trouver de quelle fonction il s'agit, on divise 676 par 4 (la taille d'un pointeur), et on cherche donc le 169^{ième} membre de la structure (en comptant à partir de zéro) : dans ce cas il s'agit d'un pointeur vers la fonction **GetStringUTFChars**.

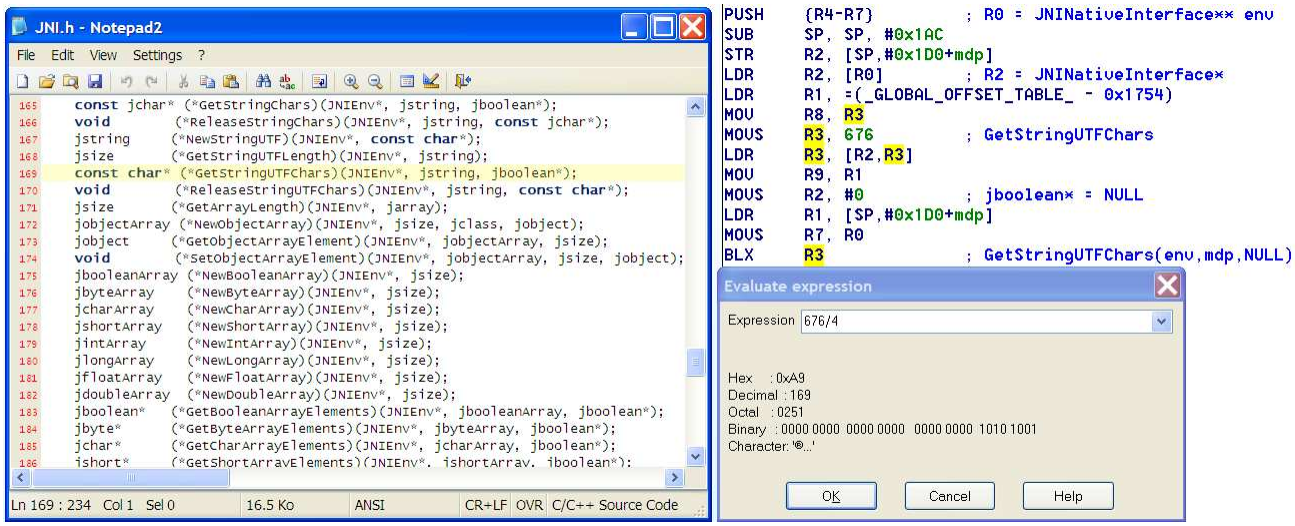

Figure 4 - Résolution d'un appel JNI « à la mano »

La résolution des appels aux méthodes de l'interface **JNINativeInterface** à été faite manuellement mais pourrait être automatisée avec un script IDAPython.

Une fois ces appels identifiés, le code se reverse relativement facilement. Une des difficultés concerne la boucle qui va convertir les 8 coordonnées GPS (4 couples latitude/longitude) en un tableau de 8 octets (nommé ici **gps_bytes**).

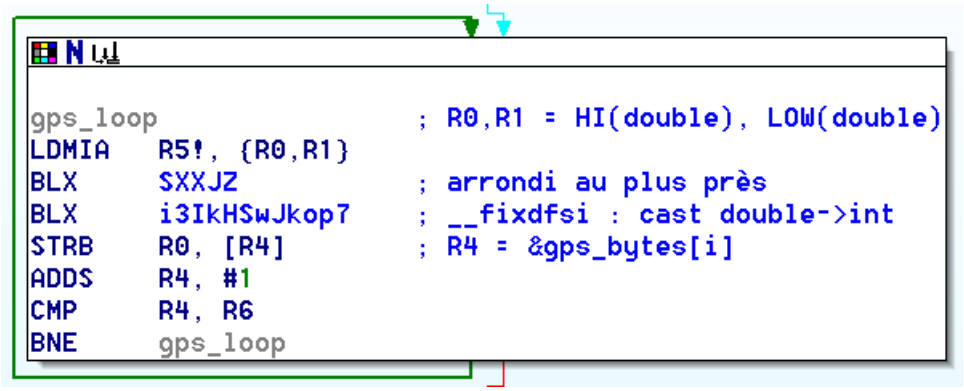

Figure 5 - Boucle de conversion des coordonnées GPS

Comme les noms des fonctions ont été obfusqués, on essaye de voir si les fonctions **SXXJZ** et **i3IkHSwJkop7** ne seraient pas tout simplement des fonctions insérées par le compilateur gcc pour le traitement des nombres en virgule flottante. Pour cela on converti le fichier **libgcc.a** en .so (pour pouvoir ouvrir tous les .o en une seule fois dans IDA), puis on utilise le plugin **PatchDiff2**.

```
$ ar -x android-ndk-r3/build/prebuilt/windows/arm-eabi-4.4.0/lib/gcc/arm-eabi/4.4.0/libgcc.a
$ android-ndk-r3/build/prebuilt/windows/arm-eabi-4.4.0/bin/arm-eabi-gcc -nostdlib -shared *.o -o
libgcc.so
```

La fonction **i3IkHSwJkop7** est identifiée comme étant la fonction **__fixdfsi**, qui correspond au cast d'un double en signed int.

La fonction **SXXJZ** n'est pas identifiée de cette façon, mais par analyse dynamique avec gdb on se rend compte qu'elle réalise un arrondi au plus proche du double passé en paramètre. On peut vérifier en posant un breakpoint juste après l'appel et observer les valeurs renvoyées en faisant varier les coordonnées GPS en entrée.

Une fois la fonction reversée, le code C correspondant est le suivant :

```c
jstring
Java_com_anssi_secret_SecretJNI_deriverclef( JNIEnv* env,
                                             jobject this,
                                             jstring jstring_password,
                                             jdoubleArray jdouble_gps )
{
    int i;
    jbyteArray bytearray_hash;
    struct hash_context ctx;
    char hash_bytes[32];
    char temp_buffer[64];
    char classname_xor[17] = { 0xF4, 0x94, 0x7D, 0x75, 0x17, 0xEE, 0x04, 0xFB,
        0xFE, 0xD4, 0x63, 0x3F, 0x15, 0xF2, 0x12, 0xFC, 0xB8
    };
    //method_signature = not_string("([B[B)V")
    char method_signature[8] = {0xD7, 0xA4, 0xBD, 0xA4, 0xBD, 0xD6, 0xA9, 0xFF };
    signed char gps_bytes[8];

    const char* password = (*env)->GetStringUTFChars(env, jstring_password, 0);

    HASH_init(&ctx, 256);

    bytearray_hash = (*env)->NewByteArray(env, 32);

    //dechiffre la signature de la méthode RC4 = "([B[B)V"
    for(i=0; i < 8; i++)
    {
        method_signature[i] = ~method_signature[i];
    }

    jdouble* gps_doubles = (*env)->GetDoubleArrayElements(env, jdouble_gps, 0);

    for(i=0 ; i < 8 ; i++)
    {
        gps_bytes[i] = ROUND_NEAREST(gps_doubles[i]);
    }

    HASH_update(&ctx, password, strlen(password), 0);

    //XXX gps_bytes[0-8] + stack ?
    HASH_update(&ctx, gps_bytes, 32, 0);

    HASH_output(&ctx, hash_bytes);

    (*env)->SetByteArrayRegion(env, bytearray_hash, 0, 32, hash_bytes);

    //xor classname
    for(i=0 ; i < 17 ; i++)
    {
        temp_buffer[i] = classname_xor[i] ^ gps_bytes[i % 8];
    }
    temp_buffer[17] = temp_buffer[0] ^ '1';
    temp_buffer[18] = temp_buffer[1] ^ ',';
    temp_buffer[19] = temp_buffer[2] ^ 'Y';
    temp_buffer[20] = temp_buffer[3] ^ '/';

    //temp_buffer = "com/anssi/secret/RC4"
    jclass secret_class = (*env)->FindClass(env, temp_buffer);

    if (!secret_class)
        goto clear_exc;

    temp_buffer[3] = '\0';

    //methodname = "com", method_signature="([B[B)V"
    jmethodID methodID = (*env)->GetStaticMethodID(env, secret_class, temp_buffer, method_signature);

    if (!methodID)
        goto clear_exc;

    //static void com/anssi/secret/RC4.com(byte[], byte[])
    //RC4 le hash avec lui même comme clé
    (*env)->CallStaticVoidMethod(env, secret_class, methodID, bytearray_hash, bytearray_hash);

    (*env)->GetByteArrayRegion(env, bytearray_hash, 0, 32, hash_bytes);

clear_exc:
    //clear les exceptions de FindClass et GetStaticMethodID
    (*env)->ExceptionClear(env);

    //passe le resultat du RC4 en hexa => clé RC4 pour le binaire "programme"
    for(i=0 ; i < 32 ; i++)
    {
        sprintf(&temp_buffer[i*2] , "%02x", hash_bytes[i]);
    }

    (*env)->ReleaseStringUTFChars(env, jstring_password, password);

    return (*env)->NewStringUTF(env, temp_buffer);
}
```

Pour essayer de déterminer l'algorithme de hachage utilisé, j'ai patché la librairie pour nopper le second appel à **HASH_update**, et créé un programme Java qui appelle la méthode native avec des paramètres choisis. Le programme de test comporte également une classe com.anssi.secret.RC4 dont la méthode com va afficher ses paramètres sur la sortie standard. Lorsque l'on fournit un mot de passe vide à la fonction de dérivation modifiée, la valeur passée à la méthode com est **aec750d11feee9f16271922fbaf5a9be142f62019ef8d720f858940070889014**. Une recherche Google de cette valeur nous renvoie sur les cas de tests de l'algorithme **SHABAL**[3] (candidat pour le concours SHA3 du NIST).

En posant un breakpoint après l'appel à la fonction que l'on a nommée **HASH_init** (à l'offset 0x178A), on affiche le contenu de la variable **ctx** :

```
(gdb) x/32x $r11
0xbe81a764:     0x41038d08      0xafe3bb74      0xafe0f3b0      0x00000000
0xbe81a774:     0xafe0f2c0      0xafe3b9bc      0x000001b8      0x00002bb4
0xbe81a784:     0x000000dc      0xad00f380      0xafe0b39b      0xad00f380
0xbe81a794:     0xad057cf9      0x00000000      0x00119b00      0x43d1eda0
0xbe81a7a4:     0x00000000      0x00000000      0x00000100      0x52f84552
0xbe81a7b4:     0xe54b7999      0x2d8ee3ec      0xb9645191      0xe0078b86
0xbe81a7c4:     0xbb7c44c9      0xd2b5c1ca      0xb0d2eb8c      0x14ce5a45
0xbe81a7d4:     0x22af50dc      0xeffdbc6b      0xeb21b74a      0xb555c6ee
```

Les valeurs surlignées correspondent bien aux constantes d'initialisation pour SHABAL256. Cependant, lorsque l'on compare le condensé de chaînes de caractères diverses, les résultats ne correspondent pas à l'application de SHABAL256. Peut être que l'algorithme a été volontairement modifié pour le rendre "prévisible" et ainsi permettre une attaque ? Dans tous les cas, une attaque par force brute ne semble pas réalisable.

CALCUL DES COORDONNEES GPS

Comme on sait que le nom de la classe passé à la fonction **FindClass** doit être **com/anssi/secret/RC4** on peut en déduire les coordonnées GPS à entrer pour obtenir le bon résultat.

Texte clair	c	o	m	/	a	n	s	s
Texte clair (hexa)	0x63	0x6F	0x6D	0x2F	0x61	0x6E	0x73	0x73
Texte chiffré (classname_xor)	0xF4	0x94	0x7D	0x75	0x17	0xEE	0x04	0xFB
gps_bytes[i] = clair[i] ^ classname_xor[i]	0x97	0xFB	0x10	0x5A	0x76	0x80	0x77	0x88
lieux[i] = gps_bytes[i] / 3.141593	48.06	79.89	5.09	28.64	37.56	40.74	37.87	43.29

Tableau 7 - Calcul inverse des coordonnées GPS

A cause des différentes conversions (double vers signed int puis tronqué en char) et arrondis, les coordonnées obtenues ne sont pas "valides" par rapport aux énigmes, mais permettent d'obtenir le bon résultat lors du déchiffrement du nom de la classe.

	Description	Latitude	Longitude
Lieu 1	Le Coq Gadby – Rennes	48.122990	-1.668516
Lieu 2	Centre spatial Guyanais	5.042850	-52.788418
Lieu 3	?	37.56	40.74
Lieu 4	?	37.87	43.29

Tableau 8 - Coordonnées GPS complétées

[3] http://www.shabal.com/

DECHIFFREMENT DU MESSAGE GPG (ELGAMAL)

Une fois le problème des coordonnées GPS réglé, il reste à trouver le bon mot de passe. La fonction de dérivation n'ayant rien donné de probant, on revient au message GPG contenu dans le fichier texte. La clé publique n'est sans doute pas là par hasard, on fait donc un peu de *Google kung-fu* pour trouver quelques informations[4] sur l'algorithme Elgamal et les attaques possibles. Celui-ci repose sur le problème du logarithme discret, et trouver la clé privée x à partir de la clé publique (g,p,y) revient à résoudre l'équation suivante :

$$y = g^x mod\ p$$

Pour que le problème du logarithme discret soit difficile à résoudre, il faut que p-1 ne puisse pas être décomposé en produit de petits facteurs premiers. Il est recommandé de choisir p tel que p-1 = 2*q avec q un grand nombre premier. Si p est mal choisi, il est possible d'utiliser l'algorithme de **Pohlig-Hellman** pour résoudre le problème du logarithme discret en un temps proportionnel à la racine carrée du plus grand facteur de la décomposition de p-1. On extrait donc les paramètres de la clé publique avec pgpdump :

```
Old: Public Subkey Packet(tag 14)(269 bytes)
        Ver 4 - new
        Public key creation time - Wed Mar 17 15:56:57 UTC 2010
        Pub alg - ElGamal Encrypt-Only(pub 16)
        ElGamal p(1024 bits) - ee 0e fd ed 46 ab 3b fa 30 c9 33 67 86 8e
 53 e0 d5 83 12 72 34 f3 69 f1 af 0f 79 e9 fa b2 00 0a 98 40 62 18 4b 1f
 47 9d 0f b9 0c e0 a3 68 ab 7f 56 62 43 48 1a a5 b4 33 86 9c 7c 0f 6d 55
 6a fd ff 17 31 a6 38 db ee 5d 86 86 f0 5a 84 95 40 19 20 30 e0 8a ab 0f
 7b ba 79 ad d6 ac a9 a4 86 a2 1a 15 46 6a f7 45 f9 25 b1 9c f5 48 f5 13
 49 2c f5 4e 53 b0 6a 08 dc 94 a0 f0 56 65 99 f6 af 43
        ElGamal g(3 bits) - 05
        ElGamal y(1024 bits) - 8a 13 78 c3 52 5d b0 b4 0a d2 ad bb 16 04
 8b c8 6f 38 2b a8 99 e6 63 f7 e9 91 03 aa b1 45 61 ee 94 c6 f9 bc 3c cc
 e2 2c 86 a5 35 52 58 81 1e 30 29 51 9e a9 28 50 f3 39 4e bd f8 f4 5b b2
 cc b6 a1 19 3c 36 50 f1 45 ff 6c cf 3d 44 54 9a 6d cf ea 2d 80 cd 57 19
 5b 24 a7 d3 ee 61 5d fd dc 01 d3 6b 69 af 20 bf 3b 01 17 0c 40 c5 73 2c
 e0 ec 86 92 bd 0a 61 e1 57 1c f7 9c 70 40 e5 83 7c fa
```

On a donc :

$p = 167170407348368755274571198710846155005852197451158304999282257593806249690235461400589475$
$405384381543451456201771311554167013086820487394167930208602015066546769638235420176232137$
$311474115639999503204837448638346143294868864953063473256150248157524356509394545569531012$
$643391645942093695428659549471312883523$

On essaye de factoriser p-1 avec Cryptool[5], qui le décompose en produit de 21 facteurs premiers :

$$\begin{aligned} p-1 = \ & 2 * 1218055055968339 * 1263847861201609 * 1271483404519507 * 1306620742471661 \\ & * 1435469233657999 * 1436852757281407 * 1455144603998677 * 1593684693149279 \\ & * 1724498562415303 * 1780716924867173 * 1917204589315909 * 1922550339910303 \\ & * 1975985172968039 * 2077649398994551 * 2108107767794563 * 2132773087614569 \\ & * 2133463604190461 * 2174110522001753 * 2227343475745711 * 3165493139633045911 \end{aligned}$$

Le plus grand facteur fait 64 bits, ce qui ne parait pas trop grand, on peut donc tenter de calculer l'exposant privé. Après quelques essais ratés avec la librairie MIRACL en C, les mots clés « discrete logarithm solver » sur Google nous renvoient vers une applet Java[6] qui implémente l'algorithme de Pohlig-Hellman. Après 8h30 de calcul on obtient la clé privée :

$x = 821600028463037887808776457389953896934820565669575595803693716610882394559987089 79$
$230174270492184887093527756473793037846785162980029537236246269728347946278173732394 32$
$307990605905765645459702616102054707$

[4] http://iml.univ-mrs.fr/~ritzenth/agregation/texte_signature-DLP.pdf
[5] http://www.cryptool.org/
[6] http://www.alpertron.com.ar/DILOG.HTM

Il faut maintenant déchiffrer la clé de session du message GPG. On utilise encore pgpdump pour récupérer le bloc « Encrypted Session Key » du message, et un script Python pour le déchiffrer.

```
Old: Public-Key Encrypted Session Key Packet(tag 1)(270 bytes)
        New version(3)
        Key ID - 0x905AA88C96C9EAD0
        Pub alg - ElGamal Encrypt-Only(pub 16)
        ElGamal g^k mod p(1021 bits) - 18 2e f7 4c 55 ce c9 bc b7 13 8f
7f d0 6e 32 bc 79 89 b4 79 d2 b6 85 6e 6c 63 7f 4a 2a a0 73 a0 b5 e3 22
4e 12 98 98 08 6b bd 8f 5f c8 15 00 e8 f4 61 83 ce 4d 86 9c d4 66 23 e5
ca f6 5d 21 a7 1e ac 5f 67 5d 7c 76 df c5 a2 4e 80 22 51 1d 09 9d b2 c8
77 b8 81 97 af 8a 49 3b 84 09 f4 50 ae 80 e4 f9 00 cf 23 55 70 f6 f9 18
fa 93 d0 55 0b c0 bc 2c e4 5c 2a 14 f1 f6 e5 06 c7 d0 20 da 5c
        ElGamal m * y^k mod p(1021 bits) - 1e e2 e1 4d b9 ed 60 ac 65 bb
 85 e4 5a 98 de a3 82 97 d1 0b 95 32 65 f7 8a 9b d3 cf ba d9 f2 44 bf 96
 81 7a 9c b1 78 b4 1c 17 de e4 45 9e ba 2f 63 35 20 d3 78 ef 8f 03 5a ad
 2f 33 55 01 ca 5d 09 8e 94 dd 6b 55 78 3a 73 de 1d c5 05 d9 9d 03 99 48
 2e 62 b8 86 e5 7d 05 20 50 57 8b 7f 22 78 42 d8 da 94 f7 2e e2 18 5b 09
 0f 87 9b 95 3b 16 bf 91 5b 16 68 89 22 75 12 39 67 77 9a 2f fd 27
                -> m = sym alg(1 byte) + checksum(2 bytes) + PKCS-1 block type 02
```

Le script suivant permet de déchiffrer la clé de session GPG en utilisant la clé privée que l'on vient de calculer.

```python
import os
from Crypto.PublicKey import ElGamal
from Crypto.Util.number import long_to_bytes

p=167170407348368755274571198710846155005852197451158304999282257593806249690235461400589475405384381543451456201771311554167013086820487394167930208602015066546769638235420176232137311474115639999503204837448638346143294868864953063473256150248157524356509394545569531012643391645942093695428659549471312883523

g=5

y=969603077155026238418838220337761808744284685743631190487303704838583042127279584086006520440405325271147188479200679591833075311173665377778810859662005837225637283981588924689861041853322149675291941024141624328796962731433945830003011996287050891849529871343532711638193453809545679304797908744480365928 90

#g^k mod p
u=0x182ef74c55cec9bcb7138f7fd06e32bc7989b479d2b6856e6c637f4a2aa073a0b5e3224e129898086bbd8f5fc81500e8f46183ce4d869cd46623e5caf65d21a71eac5f675d7c76dfc5a24e8022511d099db2c877b88197af8a493b8409f450ae80e4f900cf235570f6f918fa93d0550bc0bc2ce45c2a14f1f6e506c7d020da5c

#M*y^k mod p
v=0x1ee2e14db9ed60ac65bb85e45a98dea38297d10b953265f78a9bd3cfbad9f244bf96817a9cb178b41c17dee4459eba2f633520d378ef8f035aad2f335501ca5d098e94dd6b55783a73de1dc505d99d0399482e62b886e57d052050578b7f227842d8da94f72ee2185b090f879b953b16bf915b1668892275123967779a2ffd27

x=821600028463037887808776457389953896934820565669575595803693716610882394559987089792301742704921848870935277564737930378467851629800295372362462697283479462781737323943230799060590576564545970261610205470 7

assert pow(g,x,p) == y , "FAIL: g^x mod p != y  => wrong private key"

m = ElGamal.construct((p,g,y,x)).decrypt((u,v))

M = long_to_bytes(m)

#OpenPGP Message Format http://www.ietf.org/rfc/rfc2440.txt
#PKCS #1: RSA Encryption http://www.ietf.org/rfc/rfc2313.txt
#EB = 00 || 02 || PS || 00 || D

i = M.find("\x02")
assert i != -1, "FAIL: Encrypted block type 02"

i = M.find("\x00", i) + 1
assert i != 0, "FAIL: Encrypted block delimiter"

#D = algo_id (1 octet) || key || checksum (2 octets)

algo = ord(M[i])
session_key = M[i+1:-2]

print "Algo id = %d" % algo
print "Session key = %s" % session_key.encode("hex")

gpg_cmd = "gpg --decrypt --override-session-key %d:%s message.txt" % (algo, session_key.encode("hex"))

print gpg_cmd
os.system(gpg_cmd)
```

Enfin pour déchiffrer le message on utilise gpg avec l'option **--override-session-key** qui nous permet de spécifier la clé de session à utiliser.

```
$ gpg --decrypt --override-session-key
9:dc95f6ff93e6d8fbb212fbe09f43465a3b6574fd2e57125313df562ac7a22d15 message.txt
gpg: encrypted with RSA key, ID E1F67BBD
gpg: encrypted with 1024-bit ELG-E key, ID 96C9EAD0, created 2010-03-17
      "Personne <invalide@nomdedomaine.tld>"
O7huQcYzHEPSq82m
gpg: WARNING: message was not integrity protected
```

Le mot de passe est donc **O7huQcYzHEPSq82m**

Clé de session GPG	dc95f6ff93e6d8fbb212fbe09f43465a3b6574fd2e57125313df562ac7a22d15
Algorithme symétrique message GPG	AES-256
Message GPG déchiffré (mot de passe)	O7huQcYzHEPSq82m
Hash du mot de passe et des coordonnées GPS (binaire)	8d14b505adfc111e6cf420b19088b01d93321e93e31f7893cb4ad401560315d9
Clé RC4 du programme (texte hexa)	085ccf5487e2607d919411a60e62844e829d46917f469eb1f147042250420cc7

Tableau 9 – Récapitulatif des différentes clés

CONCLUSION

Il ne reste plus qu'à entrer les coordonnées GPS calculées et le mot de passe : le binaire déchiffré par l'application devrait être valide. On le lance dans l'émulateur et on obtient l'adresse email de validation.

```
$ adb shell
# cd /data/data/com.anssi.secret/files/
# chmod 777 binaire
# ./binaire
Bravo, le challenge est terminé ! Le mail de validation est
: 4284d974a8af53aa7a85fc4e956b2d84@sstic.org
```

Voila c'est fini ! Merci aux organisateurs pour ce challenge qui regroupait forensics, crypto et reverse.

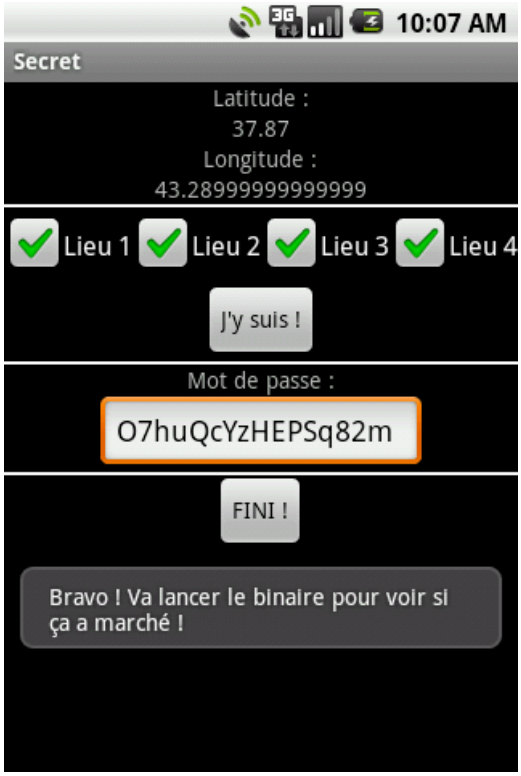

Figure 6 - The end

RÉFÉRENCES

Live Memory Forensics, Anthony Desnos

http://www.esiea-recherche.eu/~desnos/papers/slidesdraugr.pdf

Physical Memory Forensics, Mariusz Burdach

http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Burdach.pdf

Chiffrement ElGamal et attaques sur le logarithme discret, Christophe Ritzenthaler

http://iml.univ-mrs.fr/~ritzenth/agregation/texte_signature-DLP.pdf

http://developer.android.com/sdk/index.html

http://developer.android.com/sdk/ndk/index.html

http://code.google.com/p/smali/

http://sharkysoft.com/misc/vigenere/

http://www.users.zetnet.co.uk/hopwood/crypto/scan/cs.html#RC4-drop

http://www.shabal.com/

http://cgi.tenablesecurity.com/tenable/patchdiff.php

http://www.pgpdump.net/

http://www.cryptool.org/

http://www.dlitz.net/software/pycrypto/

http://www.gnupg.org/documentation/manuals/gnupg/GPG-Esoteric-Options.html

http://www.alpertron.com.ar/DILOG.HTM

ANNEXE – CODES SOURCES

PATCH QEMU

```
--- a/monitor.c
+++ b/monitor.c
@@ -868,7 +868,7 @@
      uint8_t buf[1024];
      target_phys_addr_t addr = GET_TPHYSADDR(valh, vall);

-     f = fopen(filename, "wb");
+     f = fopen(filename, "rb");
      if (!f) {
          monitor_printf(mon, "could not open '%s'\n", filename);
          return;
@@ -877,9 +877,8 @@
          l = sizeof(buf);
          if (l > size)
              l = size;
-         cpu_physical_memory_rw(addr, buf, l, 0);
-         fwrite(buf, 1, l, f);
-         fflush(f);
+         fread(buf, 1, l, f);
+         cpu_physical_memory_rw(addr, buf, l, 1);
          addr += l;
          size -= l;
      }
```

SCRIPT D'EXTRACTION DES APK

```
import struct

#copie colle des fonctions de qemu/target-arm/helper.c

def get_level1_table_address(address, CP15BASE):
    table = CP15BASE & 0xffffc000
    table |= (address >> 18) & 0x3ffc
    return table

def get_phys_addr_v5(mem, address, CP15BASE):

    table = get_level1_table_address(address, CP15BASE);

    desc = mem.read_dword(table)
    type = (desc & 3)

    if (type == 2):
        phys_addr = (desc & 0xfff00000) | (address & 0x000fffff)
    else:
        if (type == 1):
            table = (desc & 0xfffffc00) | ((address >> 10) & 0x3fc)
        else:
            table = (desc & 0xfffff000) | ((address >> 8) & 0xffc)

        desc = mem.read_dword(table)
        desc_type = desc & 3

        if desc_type == 0:
            return 0xdeadbeef
        elif desc_type == 1:
            phys_addr = (desc & 0xffff0000) | (address & 0xffff)
        elif desc_type == 2:
            phys_addr = (desc & 0xfffff000) | (address & 0xfff)
        elif desc_type == 3:
            if (type != 1):
                phys_addr = (desc & 0xfffffc00) | (address & 0x3ff)
    return phys_addr

class Mem:
    def __init__(self, filename):
        f = open(filename, "rb")
        self.data = f.read()
        f.close()

    def read_dword(self, addr):
        addr = addr & ~0xC0000000
        if addr > len(self.data):
            return 0
        return struct.unpack("<L", self.data[addr:addr+4])[0]

    def search(self, s):
        return self.data.find(s)

    def read_string(self, addr, l):
        addr = addr & ~0xC0000000
        if addr > len(self.data):
            return "FAIL"
        return self.data[addr:addr+l]

class Struct:
    def __init__(self, addr):
        self.addr = addr | 0xC0000000

    def field(self, offset):
        return mem.read_dword(self.addr + offset)

    def dump(self, n):
        for i in xrange(n):
            x = mem.read_dword(self.addr + i*4)
            print "%s + %d = 0x%x" % (self.__class__.__name__, i*4, x)
```

```
#offsets magiques
COMM_OFFSET = 0x2D4
CHILDREN_OFFSET = 0x1FC
SIBLING_OFFSET = 0x204
MM_OFFSET = 0x1C8
#http://android.git.kernel.org/?p=kernel/common.git;a=blob;f=include/linux/sched.h
class Task(Struct):
    def __init__(self, addr):
        Struct.__init__(self,addr)

        comm = mem.read_string(self.addr+COMM_OFFSET, 16)
        self.comm           = self.comm[:comm.find("\x00")]
        self.pid            = self.field(0x1EC)
        self.gid            = self.field(0x1F0)
        self.parent         = self.field(0x1F8)
        self.next           = self.field(0x204)
        self.prev           = self.field(0x204+4)
        self.child_head     = self.field(0x1FC)
        self.group_leader   = self.field(0x20C)
        self.mm             = self.field(MM_OFFSET)
        self.pgd            = mem.read_dword(self.mm + 9*4)

    def is_group_leader(self):
        return self.addr == self.group_leader

    def __repr__(self)
        s = "pid=%d\t" % self.pid
        s += "comm=" + self.comm.ljust(16)
        s += " pgd=0x%x" % self.pgd

        s += " gid=%d" % self.gid
        s += " addr=0x%x" % self.addr
        s += " parent=0x%x" % self.parent
        s += " child_head=0x%x" % self.child_head
        s += " next=0x%x" % self.next
        s += " prev=0x%x" % self.prev
        return s

    def getnext(self):
        next1 = mem.read_dword(self.addr + SIBLING_OFFSET)

        if  next1 == (self.parent + CHILDREN_OFFSET):
            return None
        return Task(next1 - SIBLING_OFFSET)

    def children(self):
        if self.child_head == self.addr + CHILDREN_OFFSET:
            return []
        child_head = cur = Task(self.child_head - SIBLING_OFFSET)
        res = []

        while True:
            res.append(cur)
            cur = cur.getnext()
            if not cur or (cur.addr == child_head.addr):
                break

        return res

    def get_children_recursive(self):
        res = {}
        for t in self.children():
            res[t.pid] = t
            res.update(t.get_children_recursive())
        return res

    def vm_areas(self):
        mm = self.field(MM_OFFSET)
        map_count = mem.read_dword(mm+12*4)
        vma = vma_head = mem.read_dword(mm)
        res = []

        while vma:
            res.append(Vm_area(vma,self))
            vma = mem.read_dword(vma+12)

        assert (len(res) == map_count), "FAIL: vm_areas count != map_count"
        return res

```

```
#http://android.git.kernel.org/?p=kernel/common.git;a=blob;f=include/linux/fs.h
class File(Struct):
    def __init__(self, addr):
        Struct.__init__(self,addr)

        self.dentry = mem.read_dword(addr+12)

        qstr_len = mem.read_dword(self.dentry  + 4*7 + 4)
        qstr_str = mem.read_dword(self.dentry  + 4*7 + 8)
        assert qstr_len < 0xFFFF, "FAIL: qstr_len too big"

        self.filename = mem.read_string(qstr_str, qstr_len)

        f_mapping = mem.read_dword(addr+ 29*4)

        nr_pages = mem.read_dword(f_mapping + 10*4)

        inode = mem.read_dword(f_mapping)

        uid = mem.read_dword(inode + 44)
        gid = mem.read_dword(inode + 48)
        #inode->i_size
        self.size = mem.read_dword(inode + 16*4)

    def __repr__(self):
        return "%s size=%d" % (self.filename, self.size)

#http://android.git.kernel.org/?p=kernel/common.git;a=blob;f=include/linux/mm_types.h
class Vm_area(Struct):
    def __init__(self, addr, task):
        Struct.__init__(self,addr)
        self.task = task

        self.vm_start = mem.read_dword(addr + 4)
        self.vm_end = mem.read_dword(addr + 8)
        self.num_pages = (self.vm_end - self.vm_start) / 4096
        self.pa_start = get_phys_addr_v5(mem, self.vm_start, self.task.pgd)

        self.file = None

        file = mem.read_dword(addr + 72)
        if file != 0:
            self.file = File(file)

    def read_pages(self, missing_entries={}):
        s = ""
        va = self.vm_start

        for i in xrange(self.num_pages):

            if missing_entries.has_key(va):
                pa = missing_entries[va]
            else:
                pa = get_phys_addr_v5(mem, va, self.task.pgd)

            if pa == 0xdeadbeef:
                s +=  "\xde\xad\xbe\xef" * 1024
                print "VA 0x%x => Page fault" % va
            else:
                print "VA 0x%x => PA 0x%x" % (va,pa)
                s += mem.read_string(pa, 4096)
            last_pa = pa
            va += 4096

        if self.file:
            return s[:self.file.size]
        return s

    def __repr__(self):
        return "0x%x-0x%x num_pages=%d pa_start=%x file=%s" % (self.vm_start,
                self.vm_end, self.num_pages, self.pa_start, str(self.file))

def write_file(filename, data):
    with open(filename, "wb") as f:
        f.write(data)
```

```
mem = Mem("challv2")

swapper_offset = mem.search("swapper\x00")

assert swapper_offset != -1, "FAIL: can't find 'swapper'"

swapper = Task(swapper_offset-COMM_OFFSET)

tasks = swapper.get_children_recursive()

#on connait les pids des 2 applis ANSSI
secret_task = tasks[233]
textviewer_task = tasks[227]

SECRET_APK = "com.anssi.secret.apk"
TEXT_APK = "com.anssi.textviewer.apk"

missing_pages = {0x426f5000: 0xc27000, 0x426f6000: 0xe08000}

for vma in secret_task.vm_areas():
    if vma.file and vma.file.filename == SECRET_APK:
        print vma
        write_file(SECRET_APK + ".zip", vma.read_pages(missing_pages))
        break

for vma in textviewer_task.vm_areas():
    if vma.file and vma.file.filename == TEXT_APK:
        print vma
        write_file(TEXT_APK + ".zip", vma.read_pages())
        break
```