Documentation reverse-engineering avancé DALVIK

Dans ce document, nous allons voir comment rajouter des fonctions à un logiciel pour Android dont nous ne disposons pas du code source. Nous désassemblerons le programme puis assemblerons nos modifications avec BakSmali.

Pour une introduction à la rétro-ingénierie sur Android, vous pouvez lire ce petit article du wiki de SmartPhoneFrance (SPF).

La plate-forme de Google, Android, est une interface graphique simplifiée, et adaptée aux périphériques mobiles, reposant sur un noyau Linux.
L'avantage de cette plate-forme est qu'elle est optimisée pour les micro-processeurs de type ARM, et potentiellement pour tous les processeurs, car Linux est développé à la fois en open-source et par les constructeurs (exemple : le Cell de la PS3 a été développé avec Linux).
Enfin, Android démarre très rapidement, et se débarrasse de tous les trop vieux concepts UNIX (comme X, qui bien que progressant fortement avec Freedesktop et xorg, n'est pas encore apte à tourner confortablement sur mobile).

Il est à noter qu'Android est incubé chez Google, et donc n'est pas réellement développé par une communauté internaute, comme c'est plus ou moins le cas avec Open Office ou Linux par exemple. Toutefois, ça permet de modifier le système et donc de personnaliser son téléphone à sa guise. Google va aussi lancer Chrome OS, dont la cible est plutôt le netbook.

Le fait que la plate-forme de base soit « open-source » n'empêche pas les constructeurs d'utiliser des pilotes propriétaires, comme le fait HTC afin de garder certains secrets. Cela n'empêche pas les opérateurs de brider le téléphone (comme ce fût le cas du Dream chez Orange ou T-Mobile) et donc de vous obliger à passer par des étapes supplémentaires (rooting, appelé jailbreak dans le monde iphone) si vous souhaitez personnaliser votre mobile.

Heureusement, cela n'empêche pas non plus un particulier comme vous et moi, de se faire un peu d'argent de poche en vendant un petit logiciel de sa conception sur l'android market :)

Pour en revenir à l'objectif de cette documentation, comme les téléphones mobiles proposent depuis quelques années aux développeurs, une plate-forme unifiée Java pour les mobiles (J2ME), Google a eu la bonne idée (contrairement à l'iphone) de suivre, en restant donc sur du Java pour la conception d'applications tierces sous Android.
Par contre, le compilateur et le jeu d'instructions sont en fait propres à Google. Les programmeurs retrouvent donc leur langage favori (disons plutôt le langage le plus usité, et multi-plateforme puisqu'il repose sur une vm), mais sans réellement Java, puisque le byte-code ainsi que la vm sont différents.
C'est Dalvik. Dalvik permet d'exploiter le téléphone et ses fonctions via le système d'exploitation, ce qui ne serait pas le cas avec le classique J2ME. Après, pourquoi ce choix de snober Sun à ce point en sortant leur propre VM, on ne sait pas.

En ce qui nous concerne, nous allons donc devoir utiliser un outil propre à Dalvik pour désassembler une application Android, Smali. Vérifiez d'abord, si à tout hasard, l'application ne serait pas open-source ! Si c'est le cas, chargez le code dans Eclipse, ce sera beaucoup plus simple ;)
Le cas de Xgalaga était très particulier, car ce soft était basé sur la version Unix, mais le code en Java de dcsoft n'était pas disponible.

Index

Prérequis....3
Désassemblage....3
  Prérequis....3
  Machine virtuelle Android....3
  Téléchargement téléphone → ordinateur....4
  Décompilation....5
Etude....6
Notions....8
  Code....8
  Ressources....9
  Exemple code + ressource....10
Test et assemblage....11
  Release....12

Prérequis

Installer le Java run-time (vous l'avez sûrement déjà)
Installer le Java development kit (JDK)
Installer le SDK Android
Installer Smali

Désassemblage

Pour désassembler (mais ce mot existe-t-il ?) un programme Android closed-source, il faut donc que vous téléchargiez Smali et Baksmali (les 2 fichiers jar, et les wrapper script), ou bien que vous utilisiez Dedexer. Les deux se présentent sous forme de cmd-let Java, mais Smali présente l'avantage de pouvoir ré-assembler du code ! En revanche, on peut espérer que Dedexer permette à moyen terme la décompilation totale en Java.

Prérequis

Pour désassembler le programme vous allez d'abord devoir télécharger l'APK sur votre ordinateur.
Connectez votre téléphone à votre ordinateur avec la câble fourni.
Installez le SDK Android. C'est le kit de développement permettant aux programmeurs de développer les applications pour Android.

Vous devrez dans un premier temps (si c'est votre 1ère utilisation du kit) démarrer le programme « android » qui se trouve sous « android-sdk-linux/tools », en tapant dans un terminal « ./android » tout en étant, donc, dans le répertoire « tools ».

Si votre ordinateur fonctionne sous windows, vous ne devez pas taper « ./ », et cela vaut pour toutes les commandes qui suivent.

Commencez par créer, sous tools, un répertoire qui contiendra tout votre travail, la décompilation, les scripts que vous téléchargerez, etc...

Machine virtuelle Android

Ce programme vous permet de créer des machines virtuelles (au sens machine complète, cela n'a rien à voir avec la VM Dalvik) dans les trois générations actuelles d'Android (1.5, 1.6, et 2.0). Créez donc (c'est facile) une VM par exemple en version android-1.5 (la plus répandue... pour le moment), si vous pensez devoir tester le code, sinon si vous êtes un warrior, vous pourrez aussi le tester directement sur votre téléphone. Il est inutile pour l'instant de lancer l'émulateur, mais si vous voulez faites « ./emulator -avd nom_de_votre_image »

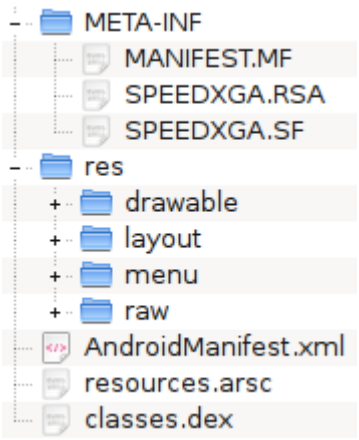
Téléchargement téléphone → ordinateur

Voici la commande à taper, pour télécharger le fichier APK de l'application que vous voulez... enrichir :

« ./adb pull /data/app/exemple.apk . » Vous obtiendrez le fichier exemple.apk dans votre répertoire courant.

Si le « device » (le téléphone) n'est jamais trouvé, il se peut que vous ne soyez pas administrateur de votre ordinateur. Si tel est le cas, lancez « sudo ./adb kill-server » puis « sudo ./adb start-server ». Ensuite recommencez. Si vous n'avez pas sudo, faites « su », tapez le mot de passe root, puis tapez directement les commandes adb kill-server et start-server.
Si vous êtes sous windows, il se peut qu'un anti-virus vous demande confirmation de charger un pilote. A savoir aussi qu'il existe une subtilité pour installer le bon pilote usb pour Android sous Vista.

Ensuite, vous devez décompresser ce fichier apk, qui n'est en fait qu'un zip. Vous obtenez alors un fichier « classes.dex » (c'est le code Dalvik), un répertoire « res » (les ressources, c'est à dire, images, définitions XML, fichiers audio...), et d'autres choses :



Ces fichiers sont signés dans le fichier MANIFEST.MF, donc il faudra re-signer avec votre signature personnelle pour la recompilation.
Vous verrez aussi que les fichiers XML disponibles ici sont en fait compilés, mais il existe un script pour les obtenir en clair.

Décompilation

Exécutez le wrapper script : « ./baksmali classes.dex », pour lancer la décompilation du programme. Vous trouverez le résultat sous un répertoire nommé « out ». Vous devriez mettre tout son contenu dans votre répertoire de travail.

Etude

Evidemment, tout bon citoyen commence par bricoler le magnétoscope sans lire la notice :)

Voyons donc à quoi ressemble du code Dalvik en syntaxe Jasmin. Notez que cette syntaxe a été retenue par Smali et Dedexer parce qu'ils pouvaient s'appuyer sur des librairies déjà existantes, me semble-t-il, des librairies nécessaires à la décompilation.

Vous remarquerez que la coloration syntaxique n'est pas appropriée, mais je n'ai pas trouvé mieux pour Jasmin. J'utilise Kate (disponible sous KDE).

Ici le début du fichier FunGameThread.smali :

```
.class Lcom/syl/speedxgalagus/FunGameThread;
.super Ljava/lang/Thread;
.source "FunGameThread.java"


# static fields
.field private static final BONUSSHIP:I = 0xc350

.field private static final GAMEOVR_STR:Ljava/lang/String; = "Game Over"

.field private static final LVL_STR:Ljava/lang/String; = "     Level: "

.field private static final PAUSED_STR:Ljava/lang/String; = "Paused"

.field private static final RESET_STR:Ljava/lang/String; = "Press any key to start a new FUN GAME"

.field private static final SCR_STR:Ljava/lang/String; = "Score: "

.field private static final UNPAUSE_STR:Ljava/lang/String; = "Press any key to continue"

.field private static final version:I = 0x1


# instance fields
.field private final GAME_OVER:I
```

Ici, une méthode privée de FunGameThread :

```
.method protected doKeyDown(ILandroid/view/KeyEvent;)Z
    .registers 8
    .parameter "keyCode"
    .parameter "msg"

    .prologue
#syl: test anti bouge pdt lvl up
    iget-object v0, v5, Lcom/syl/speedxgalagus/FunGameThread;->player:Lcom/syl/speedxgalagus/Player;
    iget-boolean v0, v0, Lcom/syl/speedxgalagus/Player;->isWarping:Z
    if-eqz v0, :goto_zza
    return v0
    :goto_zza
#syl
    const/4 v4, 0x0

    const/4 v3, 0x1

    .line 454
    iget-object v0, v5, Lcom/syl/speedxgalagus/FunGameThread;->surfaceHolder:Landroid/view/SurfaceHolder;

    monitor-enter v0

    .line 456
    :try_start_5
    invoke-virtual {v5}, Lcom/syl/speedxgalagus/FunGameThread;->gamePaused()Z

    move-result v1

    if-eqz v1, :cond_11

    .line 457
    invoke-virtual {v5}, Lcom/syl/speedxgalagus/FunGameThread;->resumeGame()V
```

Cette méthode « DoKeyDown » est en fait appelée quand une touche d'un clavier physique est relâchée.
Le code que j'ai rajouté sert à faire en sorte que si l'attribut « isWarping » (correspondant au changement de level) est positionné, alors on fait comme si la touche n'avait pas été pressée.
Une fois ceci fait dans les autres fonctions telles que « doTrackballUpAndDown », ça permet de bloquer le vaisseau du joueur durant la phase animée du changement de niveau.

Les commentaires comme « #syl: test anti bouge pdt lvl up » disparaissent malheureusement après compilation, donc il faut bien évidemment conserver votre code, afin de garder ces commentaires de vos modifications.

Ce fichier FunGameThread, je l'ai créé simplement en dupliquant GameThread, et en remplaçant dedans toutes les occurrences de « GameThread » en « FunGameThread ». Cela m'a permis de réaliser le FunMode (il y a aussi un FunGameView.smali) au mieux en séparant le mode normal du mode Fun. C'est une des méthodes que vous devez connaître.

Notions

Code

Après avoir fait le tour de ce que nous voulons, attardons nous tout de même sur les instructions Dalvik, dont vous aurez forcément besoin.

Par exemple pour augmenter une valeur constante, il faudra parfois changer sa déclaration, et donc lire un peu de doc.

Ainsi, si l'on veut mettre une valeur qui était à 7, à 8, il faut changer sa déclaration :
const/4 v0, 0x7
const/16 v0, 0x8
car ce n'est plus le même type de constante. Ca sera détecté par le compilateur sans problème.

En revanche il y a d'autres choses qui ne seront détectées que par un bon crash.
Comme on peut le lire dans l'exemple « DoKeyDown », il faut respecter certaines règles quand on rajoute du code.

- On peut voir que l'attribut « isWarping » peut être lu seulement après que l'objet « Player » ait été chargé dans « FunGameThread ».
- De plus, les registres v0 et v5 ne sont pas choisis par hasard.

Ces 2 constatations que je vous fait partager, ce sont des heures de recherche pour moi. Car parfois (imaginons que plus tôt dans le code, « Player » soit déjà utilisé) il n'est pas besoin d'initialiser quoi que ce soit. Cela dépend du contexte car Java est un langage objet.

90% des crash que j'ai occasionnés sont dûs au non respect de la première règle, les 9% restants au non respect de la seconde :)
Observez le code, trouvez l'endroit idéal pour y insérer votre fonction, vérifiez comment accéder aux valeurs ou aux méthodes dont vous avez besoin, et essayez de reproduire la logique des registres utilisés à cet endroit là.

Il reste 1%: c'est d'appeler une fonction convenablement, soit avec invoke-direct ou avec invoke-virtual.

En fait il faut garder à l'esprit que l'on manipule du code objet, mais à très bas niveau, et donc le contexte compte énormément. Il est bien évident que le programmeur, en Java, aurait eu la vie plus simple.

Rares sont les lignes inutiles, donc essayez de les analyser (même à d'autres endroits du programme) afin de comprendre si vous faites quelque chose de mauvais. Un outil de recherche de texte à l'intérieur des fichiers de tout un répertoire est indispensable.

Désolé de ne pas être plus bavard, mais tout est là et c'est l'expérience qui pourra le mieux vous aiguiller. Faut y aller à tâton au début car à ma connaissance, on ne peut débugger que du code Java avec Eclipse... Faites des sauvegardes, une chose à la fois, tout ça...

Ressources

Les ressources, ce sont les fichiers son, images, propriétés XML, et autres fichiers de données propres à l'application.
Des fichiers XML décrivent les fenêtres et les menus des applications Android.

Ces fichiers tels que vous les trouvez dans un apk sont en fait compilés, mais vous pouvez les décompiler avec AXMLPrinter2.jar.
Tapez la commande « java -jar AXMLPrinter2.jar AndroidManifest.xml » pour décompresser la description générale de l'application.

Vous noterez que les descriptions contiennent parfois des chiffres qui ressemblent à des adresses 32-bit hexadécimales, telles que
« «
android:label="@7F050000"
» »

cela signifie que vous devrez (mais uniquement si vous désirez insérer de nouvelles ressources) remplacer ces valeurs hexadécimales par des valeurs qui ont du sens pour le compilateur... ou bien ne rien toucher et laisser les fichiers xml compilés tels quels.

Vous devrez consulter les fichiers R$....smali pour comprendre comment décrypter totalement les fichiers XML.

Par déduction, vous changerez alors la déclaration par :

« «
android:label="@string/app_name"
» »

car en lisant le fichier R$string.smali, on peut lire

« «
.field public static final app_name:I = 0x7f050000
» »

Vous y êtes ?
Ca va être long, bon courage !

Mais encore une fois, vous devriez laisser les fichiers xml compilés en place si vous ne comptez pas insérer de nouvelles ressources. Et si vous voulez juste changer une image, vous n'avez qu'à la remplacer (avec le même nom, ou que ça reste à la même position dans l'ordre alphabétique).

Quelques notes prises au sujet de la décompilation des ressources:

#android:screenOrientation="1" android:screenOrientation="portrait"
globalement on observe qu'une valeur commençant par @7f07 se trouve sous R$id.smali, donc s'écrira @+id/valeur (le + car id est un peu spécial, c'est juste un repère pour le programme)
une valeur commençant par @7F02 se trouve sous R$drawable.smali, donc s'écrira @drawable/valeur

Exemple code + ressource

Pour finir, un exemple de ressource utilisée dans le code original de XGalaga, voici du code à déchiffrer, venant de SpeedXGalagus.smali :

```
.method public onStart()V
    .registers 5

    .prologue
    .line 55
    invoke-super {v4}, Landroid/app/Activity;->onStart()V

    .line 57
    const/4 v2, 0x1

    .line 59
    .local v2, saveGameExists:Z
    :try_start_4

## syl : ici le programme cree un menu si state existe (le menu resume)
    const-string v3, "SpeedXGalagus_State"
    invoke-virtual {v4, v3}, Lcom/syl/speedxgalagus/SpeedXGalagus;->openFileInput(Ljava/lang/String;)Ljava/io/FileInputStream;

    :try_end_9
    .catch Ljava/lang/Exception; {:try_start_4 .. :try_end_9} :handler_1c

    .line 64
    :goto_9
##syl: ici c'est le bouton resume
    const v3, 0x7f070003

    invoke-virtual {v4, v3}, Lcom/syl/speedxgalagus/SpeedXGalagus;->findViewById(I)Landroid/view/View;

    move-result-object v1

    check-cast v1, Landroid/widget/Button;

    .line 65
    .local v1, resume:Landroid/widget/Button;
    if-eqz v2, :cond_20

    const/4 v3, 0x0

    :goto_15
    invoke-virtual {v1, v3}, Landroid/widget/Button;->setVisibility(I)V

    .line 67
    const/4 v3, 0x1

    iput-boolean v3, v4, Lcom/syl/speedxgalagus/SpeedXGalagus;->newGame:Z

    .line 68
    return-void
```

Il s'agit de la fonction de démarrage de l'activité principale « SpeedXGalagus ».

Le logiciel créé le l'entrée de menu « Resume » (7f070003) et se sert de « SetVisibility » pour montrer le bouton, si le fichier « SpeedXGalagus_State » est découvert.

Test et assemblage

Maintenant que vous avez éventuellement fait vos modifications, vous allez pouvoir réassembler avec ce script:

```
# on supprime les fichiers temporaires (pour zipalign happy)
rm -rf bin/*

echo "" && echo "compilation dalvik dex"
# j'utilise le smali wrapper script (n'oubliez pas le "chmod +x smali")
./smali . -o bin/classes.dex

echo "" && echo "ressources"
../../platforms/android-1.5/tools/aapt p -f -M AndroidManifest.xml -S res -I ../../platforms/android-1.5/android.jar -F bin/speedxgalagus-res.ap_

echo "" && echo "assemblage apk"
../apkbuilder bin/speedxgalagus.apk_ -z bin/speedxgalagus-res.ap_ -f bin/classes.dex

# ancienne methode manuelle (modification de ressources impossible)
#echo "" && echo "compression"
#cd xgalagus-bin/
#zip -q -0 -r xgalagus.apk *
#mv xgalagus.apk ../
#cd ../
#echo "" && echo "signature"
#./my-release-key.keystore.sh xgalagus.apk

echo "" && echo "alignement"
../zipalign 4 bin/speedxgalagus.apk_ bin/speedxgalagus.apk

# vous devriez déinstaller d'abord si des changements ont lieu par rapport aux fichiers de données de votre programme
echo "" && echo "desinstallation"
../adb -e uninstall com.syl.speedxgalagus

# le "-r" pour réinstallation en laissant les fichiers de données de l'application intacts
echo "" && echo "reinstallation"
../adb -e install -r bin/speedxgalagus.apk
```

Vous notez que les commandes adb ont toutes le switch -e pour lancer sur l'émulateur. Si vous voulez lancer sur le device, précisez -d à la place.

Pour les nuls en Unix, « ../ » signifie répertoire parent, parce que ce script est à lancer dans un sous-répertoire de tools, et les différents scripts comme smali ont été amenés dedans.

Release

Maintenant, si vous projetez de releaser, générez votre jeu de clés :

keytool -genkey -v -keystore my-release-key.keystore -alias alias_name -keyalg RSA -validity 10000

mv my-release-key.keystore ~/.android/speedxgalagus.keystore

et rajoutez ceci à votre script juste avant le zipalign :

echo "" && echo "signature avec cle de release speedxgalagus"

jarsigner -verbose -keystore ~/.android/speedxgalagus.keystore bin/speedxgalagus.apk_ speedxgalagus

Sylvain------- :)