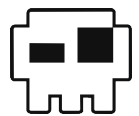
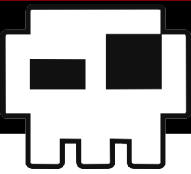





Reversing Android

Aïe, robot ...








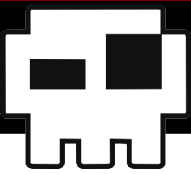


Système Android

-  Dalvik VM
-  Format de paquetage
-  Structure d'un paquetage

Ressources

-  Stockage des ressources
-  Strings
-  Ressources graphiques
-  Menus
-  Manifeste

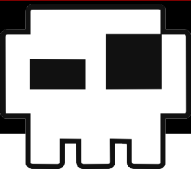


Première approche

- ☞ Dex2jar + jd-gui
 - Full java
- ☞ Avantages & inconvénients

Seconde approche

- ☞ Smali & baksmali
 - Full smali
- ☞ Avantages & inconvénients

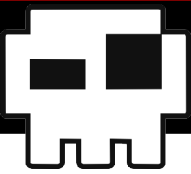


Langage smali

- ☞ Présentation
- ☞ Méthodes, types, et variables
- ☞ Instructions courantes

Patching

- ☞ Modification de code
- ☞ Reconstruction d'APK
- ☞ Signature



Plan (4/4)

Protections usuelles

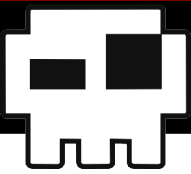
- ☞ Analyse de protections d'applications du market
- ☞ Evaluation du niveau des protections

Systèmes de protection

- ☞ Protectors android
- ☞ Google licensing
- ☞ Astuces anti-reverse




Pistes à suivre

- ☞ Manipulation de fichiers DEX
- ☞ Techniques de protection des applications








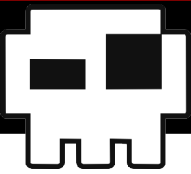
Plan (1/4)

Système Android

-  Dalvik VM
-  Format de paquetage
-  Structure d'un paquetage

Ressources

-  Stockage des ressources
-  Strings
-  Ressources graphiques
-  Menus
-  Manifeste



Android

 OS pour smartphones, PDAs, et tablet

☞ Développé par une startup

☞ Racheté par Google

 Avantages

☞ SDK fourni par Google

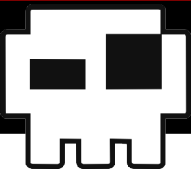
- Outils de développement dédiés

☞ Plateforme opensource et ouverte

☞ Emulateur Android fourni

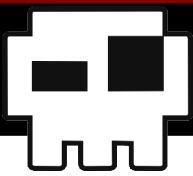
☞ Outils de dev basés sur Eclipse

 <http://android.google.com>



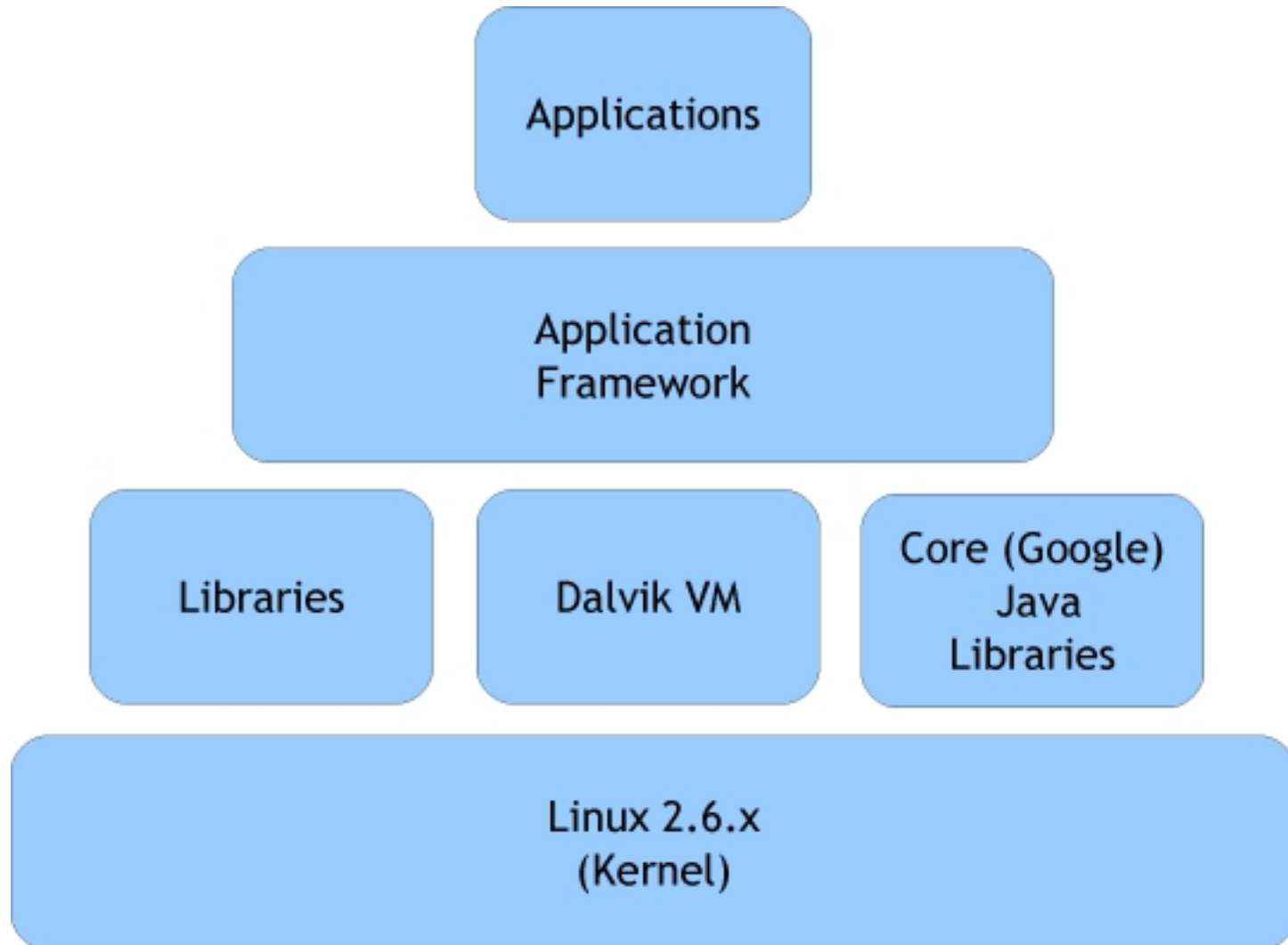
Dalvik VM

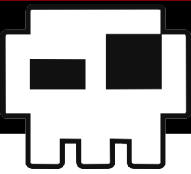
- 🤖 Ecrite originellement par Dan Bornstein
- 🤖 Porte le nom d'un village islandais
- 🤖 VM employant un bytecode dérivé de celui de la JVM
 - ☞ VM employant des registres (!= JVM)
 - ☞ A partir d'android 2.2, intègre un JIT-compiler
 - ☞ Intègre un cloisonnement des applications
 - ☞ Intègre un système de « permissions »
- 🤖 Pas de JAR, mais un fichier DEX
 - ☞ Dalvik EXecutable



Dalvik VM

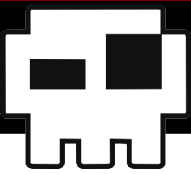
🐘 Structure d'android (VM + système)





Contraintes liées au reversing

- ☞ Bytecode spécifique à Dalvik
- ☞ Format de fichier DEX spécifique (mais connu)
- ☞ Debogage difficile
 - Impossible de déboguer (a priori) sans les sources
 - Utilisation d'un émulateur nécessaire
 - Utilisation d'un vrai téléphone possible



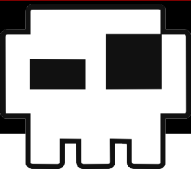
Format de paquetage

 Android utilise des « APKs »

- ☞ Android Package
- ☞ Fichier au format ZIP

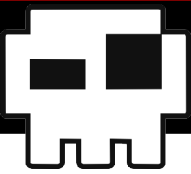
 Un APK contient

- ☞ Le fichier DEX (code exécutable de l'application)
- ☞ Un dossier contenant les ressources
 - Graphismes
 - Chaînes de caractères localisées
 - Layouts
- ☞ Un dossier contenant les signatures des fichiers
 - Applications signées



Structure d'un APK dézippé

- AndroidManifest.xml
- classes.dex
- res/
 - drawable/
 - layout/
 - raw/
 - rml/
- resources.arsc

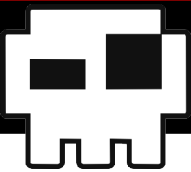


AndroidManifest.xml

- ☞ Définit les propriétés et activités de l'application
- ☞ Format XML encodé pour Android
 - Illisible tel quel

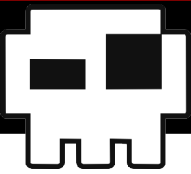
Le fichier de manifeste contient

- ☞ Les permissions nécessaires à l'application
- ☞ Les informations sur les activités
- ☞ Les informations sur les services



Fichier classes.dex

- ☞ Fichier au format DEX contenant le code de l'application
- ☞ Le format DEX est connu
- ☞ Caractéristiques de DEX
 - Chaînes de caractères mutualisées et stockées de manière unique (annuaire)
 - Stockage des propriétés des classes de manière compacte
 - Emploie certains types permettant de gagner de la place
 - ULEB128 et SLEB128
 - Dictionnaire de valeurs
- ☞ Optimisé pour économiser de la place sur disque



Format de paquetage

Android Asset Packaging Tool

☞ Outil nécessaire à l'analyse de paquetage

Permet de

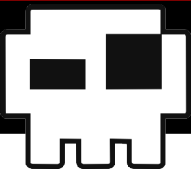
☞ Lister les ressources

☞ Lister les permissions

☞ Modifier un fichier APK

- Ajout/Suppression de fichier
- Renommage
- ...

☞ Packager un APK



Format de paquetage

Dexdump

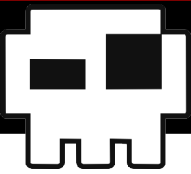
- 👉 Outil du SDK
- 👉 « must-have » fourni par google !

Permet de

- 👉 Désassembler des sections de code
- 👉 Afficher des informations sur les en-têtes
- 👉 Afficher des informations sur les méthodes
- 👉 Afficher des informations sur les types

 Maintenu par Google, pourquoi s'en priver ?

 Axé sur l'analyse de *classes.dex*

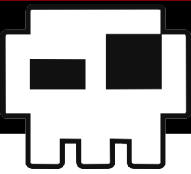


DEX

- ☞ Format d'exécutable de Dalvik
- ☞ Format de fichier connu et documenté




Particularités de DEX

- ☞ Emploi de types « spéciaux »
 - ULEB128 et SLEB128
- ☞ Structures spécifiques à DEX et documentées (enfin, presque ;)
- ☞ Données indexées
 - Ressemble à du .Net en vraiment plus lite
 - Index croisés








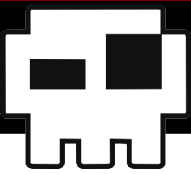
Plan (1/4)

Système Android

-  Dalvik VM
-  Format de paquetage
-  Structure d'un paquetage

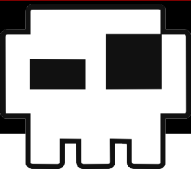
Ressources

-  Stockage des ressources
-  Strings
-  Ressources graphiques
-  Menus
-  Manifeste



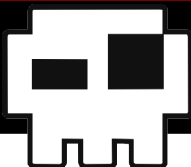
Ressources d'un APK

- ☞ Dictionnaires de textes (Strings)
 - Internationalisation des applications
 - Facilité de traduction
- ☞ Graphismes
 - Centralisation des images
- ☞ Layouts
 - Elements d'affichage structurés
 - Fichiers de définition d'interface
 - Peuvent contenir du texte (attributs)
- ☞ Menus
- ☞ Manifeste



Ressources XML

- ☞ XML encodé spécialement pour Android
- ☞ Illisible tel quel
- ☞ Des outils existent pour les décoder
 - Aapt
 - Apktool
- ☞ Éléments référencés par des ID sur 32 bits dans le code
 - On va devoir faire la correspondance entre ID de ressource et contenu de la ressource
 - Mapping nécessaire



Strings

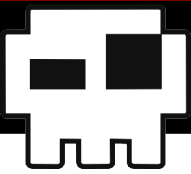
☞ Stockées dans */res/values*

- Fichier *strings.xml*

☞ Internationalisation

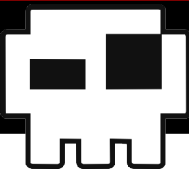
- Dossiers par locale
- *Values-fr*
- *Values-en*
- ...

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
  <string name="hello">Hello World, Crackoid!</string>
  <string name="app_name">Crackoid</string>
</resources>
```



Layouts

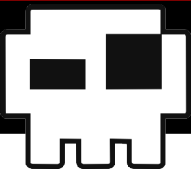
- ☞ Fichiers XML définissant les éléments graphiques
- ☞ Chargés par le code
 - Méthode *SetContentView* de l'activité
- ☞ Remplacent avantageusement la création de l'interface par programmation
 - Nombreux widgets disponibles
 - Gestion native du redimensionnement
 - Gestion des proportions
 - Atributs précisant la visibilité des éléments
 - Visibility = (GONE | VISIBLE | INVISIBLE)



Manifeste

- 👉 Fichier *Manifest.xml*
- 👉 Permet d'identifier l'activité principale
- 👉 Permet d'identifier les services

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.hzv.crackoid"
    android:versionCode="1"
    android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Crackoid"
            android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

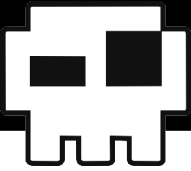


Première approche

- ☞ Dex2jar + jd-gui
 - Full java
- ☞ Avantages & inconvénients

Seconde approche

- ☞ Smali & baksmali
 - Full smali
- ☞ Avantages & inconvénients



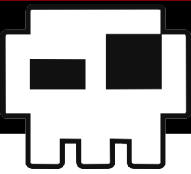
Principe

- ☞ L'outil *dx* convertit des fichiers *.class* en fichier DEX
- ☞ Le processus est réversible
- ☞ DEX → JAR → Java

Outils

- ☞ Dex2jar
 - Effectue le processus inverse (d'où le nom)
 - Régénère le bytecode Java
- ☞ Jd-gui
 - Désassembleur Java

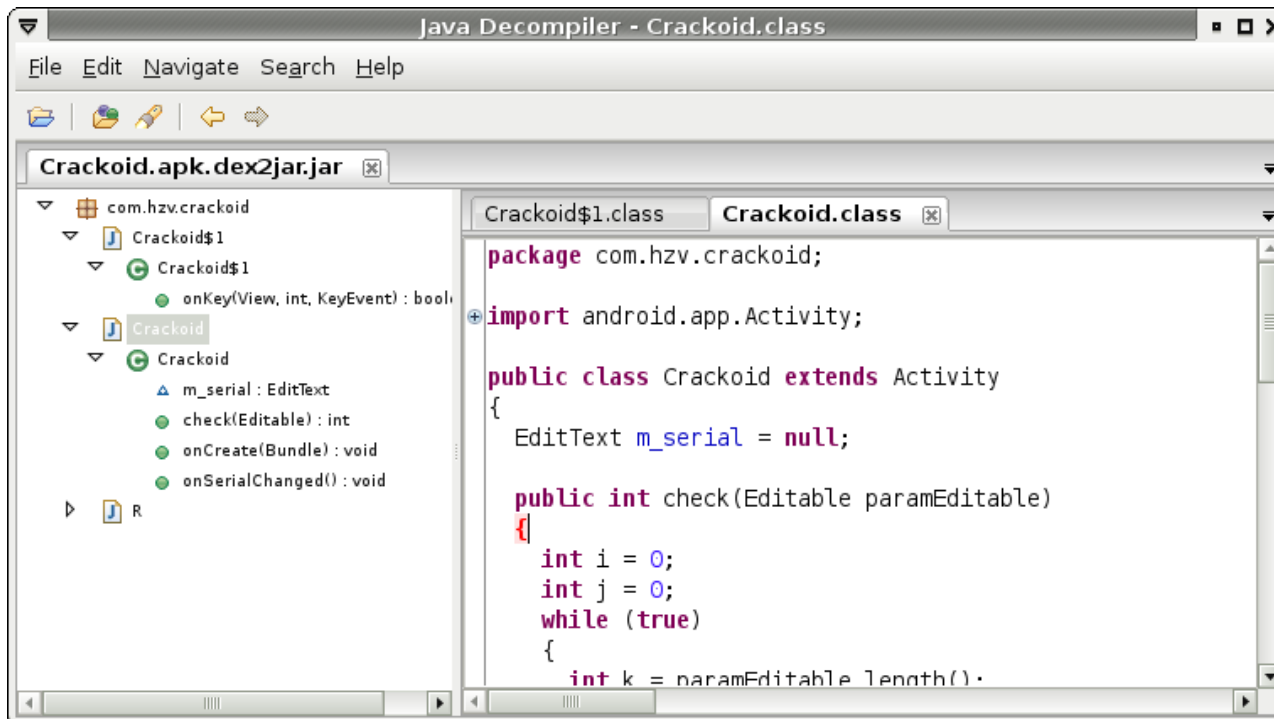
Cross-platform



Dex2jar + jd-gui

Mode opératoire

```
$ dex2jar Example.apk
version:0.0.7.8-SNAPSHOT
2 [main] INFO pxb.android.dex2jar.v3.Main - dex2jar Example.apk ->
Example.apk.dex2jar.jar
Done.
$ jd-gui Example.apk.dex2jar.jar
```



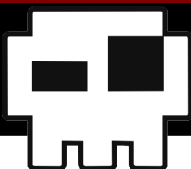
The screenshot shows the Java Decompiler application window titled "Java Decompiler - Crackoid.class". The interface includes a menu bar (File, Edit, Navigate, Search, Help), a toolbar with icons for file operations, and a project tree on the left. The project tree shows the package structure: com.hzv.crackoid, Crackoid\$1, Crackoid, and R. The main editor displays the decompiled Java code for Crackoid.class:

```
package com.hzv.crackoid;

import android.app.Activity;

public class Crackoid extends Activity
{
    EditText m_serial = null;

    public int check(Editable paramEditable)
    {
        int i = 0;
        int j = 0;
        while (true)
        {
            int k = paramEditable.length();
```



Dex2jar + jd-gui

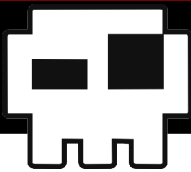
Exemple de code désassemblé

```
package com.hzv.crackoid;

import android.app.Activity;

public class Crackoid extends Activity
{
    EditText m_serial = null;

    public int check(Editable paramEditable)
    {
        int i = 0;
        int j = 0;
        while (true)
        {
            int k = paramEditable.length();
            if (j >= k)
                return i;
            int m = paramEditable.charAt(j);
            i = (9439507 * m + i) * 4919;
            j += 1;
        }
    }
}
```

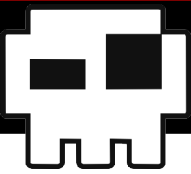


Avantages

- ☞ Obtention du code java quasi original
- ☞ Simple d'emploi
- ☞ Analyse facilitée par l'éditeur

Inconvénients

- ☞ Impossible de modifier quoique ce soit
- ☞ Constantes non résolues (valeurs entières)
- ☞ Ressources non décodées
- ☞ Identifiants de ressources présents sous forme numérique seule

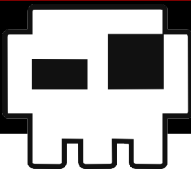


Première approche

- ☞ Dex2jar + jd-gui
 - Full java
- ☞ Avantages & inconvénients

Seconde approche

- ☞ Smali & baksmali
 - Full smali
- ☞ Avantages & inconvénients

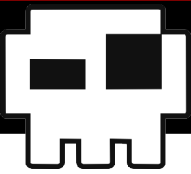


Principe

- ☞ Désassemblage du bytecode Dalvik
- ☞ Production d'un code source smali
- ☞ Réassemblage du code source smali
- ☞ Production d'un nouvel APK

Outil

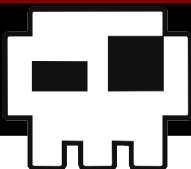
- ☞ Smali
 - Assembleur
- ☞ Baksmali
 - Désassembleur
- ☞ Intégrés dans l'outil apktool



Mode opératoire

Utilisation d'apktool

```
$ apktool d Example.apk  
I: Baksmaling...  
I: Loading resource table...  
I: Decoding resources...  
I: Loading resource table from file: /home/xxxx/apktool/framework/1.apk  
I: Copying assets and libs...
```



Exemple de code produit

```
# virtual methods
.method public check(Landroid/text/Editable;)I
    .locals 4
    .parameter "t"

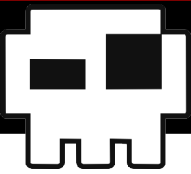
    .prologue
    .line 36
    const/4 v1, 0x0

    .line 37
    .local v1, sum:I
    const/4 v0, 0x0

    .local v0, i:I
    :goto_0
    invoke-interface {p1}, Landroid/text/Editable;->length()I

    move-result v2

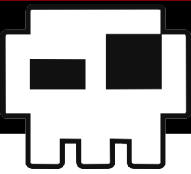
    if-lt v0, v2, :cond_0
```

Exemple de mapping de ressources

☞ Propre à apktool (fichier */res/values/public.xml*)

```
<?xml version="1.0" encoding="UTF-8"?>
<resources>
  <public type="drawable" name="android_logo" id="0x7f020000" />
  <public type="drawable" name="icon" id="0x7f020001" />
  <public type="layout" name="main" id="0x7f030000" />
  <public type="string" name="hello" id="0x7f040000" />
  <public type="string" name="app_name" id="0x7f040001" />
  <public type="id" name="serial_input" id="0x7f050000" />
  <public type="id" name="serial" id="0x7f050001" />
  <public type="id" name="serial_cracked" id="0x7f050002" />
  <public type="id" name="cracked" id="0x7f050003" />
</resources>
```

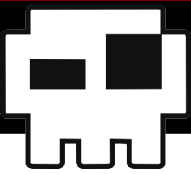


Avantages

- ☞ Désassemblage ET réassemblage
- ☞ Permet de patcher le code
- ☞ Permet de reconstruire un APK
- ☞ Décode les ressources

Inconvénients

- ☞ Langage Smali
- ☞ Beaucoup de fichiers générés
 - Pas simple de s'y retrouver
 - Pas simple de localiser le code intéressant

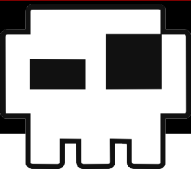


Langage smali

- 👉 Présentation
- 👉 Méthodes, types, et variables
- 👉 Instructions courantes

Patching

- 👉 Modification de code
- 👉 Reconstruction d'APK
- 👉 Signature



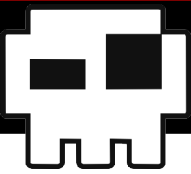
Langage Smali

Abus de langage

- ☞ Ce n'est pas un langage
- ☞ « smali » signifie « assembleur » en islandais
- ☞ Smali & baksmali utilisent la syntaxe Jasmin

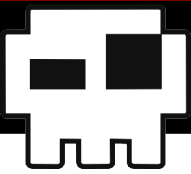
Plus précisément

- ☞ Transposition du bytecode DEX en pseudo-assembleur
- ☞ On va quand meme appeler ca « langage smali »
 - Plus simple
 - Pas forcément correct
 - Mais ça ponce les mamans ours



Types

- ☞ La notation de type est particulière
 - Une lettre précise le type
 - Un complément peut préciser une classe, une interface, etc...
- ☞ Types standards
 - V: Void (valide seulement pour les types de retour)
 - Z: booléen
 - B: octet (byte)
 - S: short
 - C: char
 - I: int





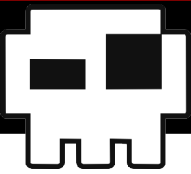
Types (suite)

Types standards (suite)

- J: long
- F: float
- D: double
- L<nom>: classe
- [<nom>: tableau d'éléments de type <nom>

Méthodes

-  Utilisent la description des types vue précédemment
-  Type de retour précisé



Langage Smali

Exemple de méthode smali

 *.method public onCreate(Landroid/os/Bundle;)V*

 Nom

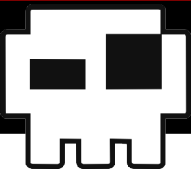
- onCreate

 Paramètre

- Instance de classe android.os.Bundle

 Retourne

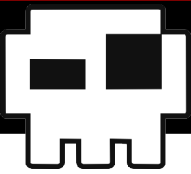
- Void



Exemple d'appel à une méthode

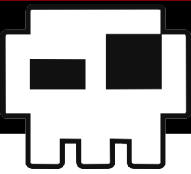
```
invoke-virtual {p0, v0}, Lcom/hzv/crackoid/Crackoid;->findViewById(I)Landroid/view/View;
```

- Appelle une méthode virtuelle nommée *findViewById*
- Cette méthode prend en paramètre un entier, ici le contenu de la variable *v0*
- Elle retourne une instance de la classe *android.view.View*
- Le paramètre *p0* est une instance de classe de type *com.hzv.crackoid.Crackoid*



Variables

- ☞ La VM Dalvik utilise des registres, et non la stack
- ☞ Il existe donc des registres définis
 - Vx: registres dédiés
 - Nombre de registres définis dans le code smali
 - .locals 2
- ☞ Ces registres peuvent contenir n'importe quel type de donnée



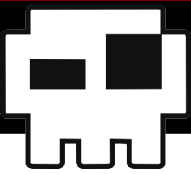
Langage Smali

Instructions courantes




- ☞ `invoke-virtual`: appelle une méthode virtuelle
- ☞ `iput-object`: copie la valeur d'une variable dans un champ d'une instance de classe
- ☞ `iget-object`: copie la valeur d'un champ d'une instance de classe dans une variable
- ☞ `new-instance`: instancie une classe, et stocke la référence de l'instance dans une variable
- ☞ `Move-result-object`: stocke l'objet résultant d'un précédent appel dans une variable

Approfondir




- ☞ http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html

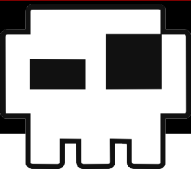


Langage smali

-  Présentation
-  Méthodes, types, et variables
-  Instructions courantes

Patching

-  Modification de code
-  Reconstruction d'APK
-  Signature



Patching

Modification du code

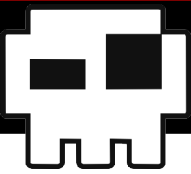
- 👉 Une fois le code intéressant localisé,
- 👉 On le modifie et on sauvegarde le fichier smali

```
invoke-virtual {p0, v3}, Lcom/hzv/crackoid/Crackoid;->check(Landroid/text/Editable;)I
move-result v1

.line 45
.local v1, s:I
const v3, 0x5bed342
if-ne v1, v3, :cond_0

.line 47
const/high16 v3, 0x7f05

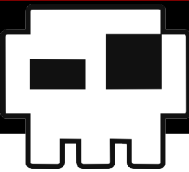
invoke-virtual {p0, v3}, Lcom/hzv/crackoid/Crackoid;->findViewById(I)Landroid/view/View;
```



Reconstruction de l'APK

☞ Apktool à la rescousse !

```
$ apktool b Crackoid  
I: Checking whether sources has changed...  
I: Smaling...  
I: Checking whether resources has changed...  
I: Building apk file...
```

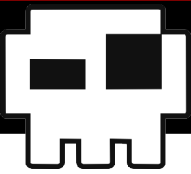


Patching

Signature de l'APK

- ☞ Un APK contient des signatures de fichiers
- ☞ Génération d'un keystore
 - Keytool -genkeypair -keystore <keystore>
- ☞ Signature du fichier APK
 - Jarsigner -verbose -keystore <keystore> <apk> <alias>

```
$ jarsigner -verbose -keystore ~/android.keystore Crackoid.apk virtualabs
Enter Passphrase for keystore:
  adding: META-INF/MANIFEST.MF
  adding: META-INF/VIRTUALA.SF
  adding: META-INF/VIRTUALA.RSA
signing: res/drawable/android_logo.png
signing: res/drawable/icon.png
signing: res/layout/main.xml
signing: AndroidManifest.xml
signing: classes.dex
signing: resources.arsc
```



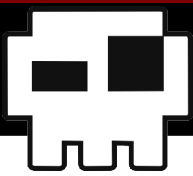
Patching

 Le fichier APK est prêt à être installé

- ☞ NB: la clef publique utilisée pour la signature n'a pas été approuvée par Google
- ☞ Il faut accepter les sources d'applications inconnues

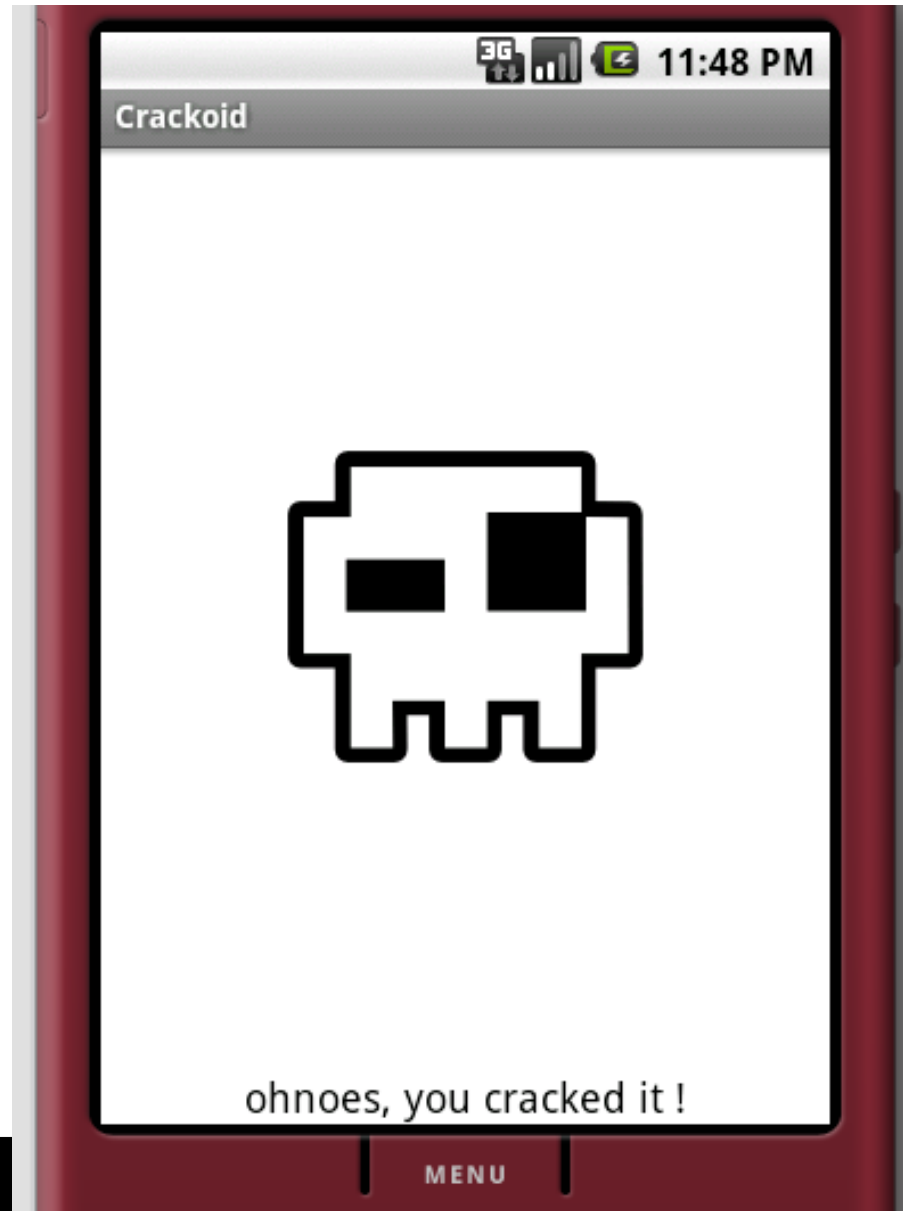
 Installation de l'apk avec adb

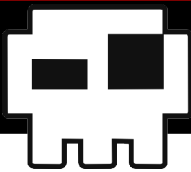
```
$ adb install Crackoid.apk
121 KB/s (8770 bytes in 0.070s)
  pkg: /data/local/tmp/Crackoid.apk
Success
$
```





Patching

🤖 Test de l'application patchée








Protections usuelles

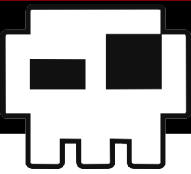
-  Analyse de protections d'applications du market
-  Evaluation du niveau des protections

Systèmes de protection

-  Protectors android
-  Google licensing
-  Astuces anti-reverse

Pistes à suivre

-  Manipulation de fichiers DEX
-  Techniques de protection des applications



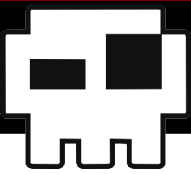
Protections usuelles

Mécanismes de protection

- ☞ Retour aux années 1990 !!
- ☞ Pas d'obfuscation de code
- ☞ Pas de schéma complexe de protection
- ☞ Routines de vérification aisément modifiables
- ☞ Checks franchement idiots

Analyse de deux applications du market

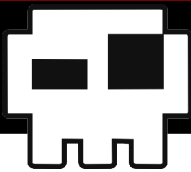
- ☞ DroidPod Shuffle Black (je ne sais même pas à quoi ca sert ...)
- ☞ OfficeSuite (coûte cher en plus ...)



DroidShuffle Black

- ☞ Désassemblage avec apktool
- ☞ Un fichier de gestion de licence !
 - LicMan.smali
- ☞ Plusieurs méthodes intéressantes
 - check4Key()
 - getExpireTime()
 - GetLicense()

Analysons !



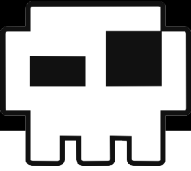
Méthode `getLicense()`

☞ Appel à `check4Key()`

☞ `check4Key()`

- Variable locale `v8` « `unlocked` » à `False` dès le départ
- On peut patcher cette méthode de manière à retourner toujours « `unlocked` »

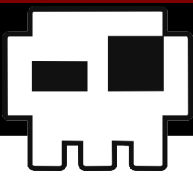
☞ Suffisant pour forcer l'enregistrement !



Modification de check4Key()

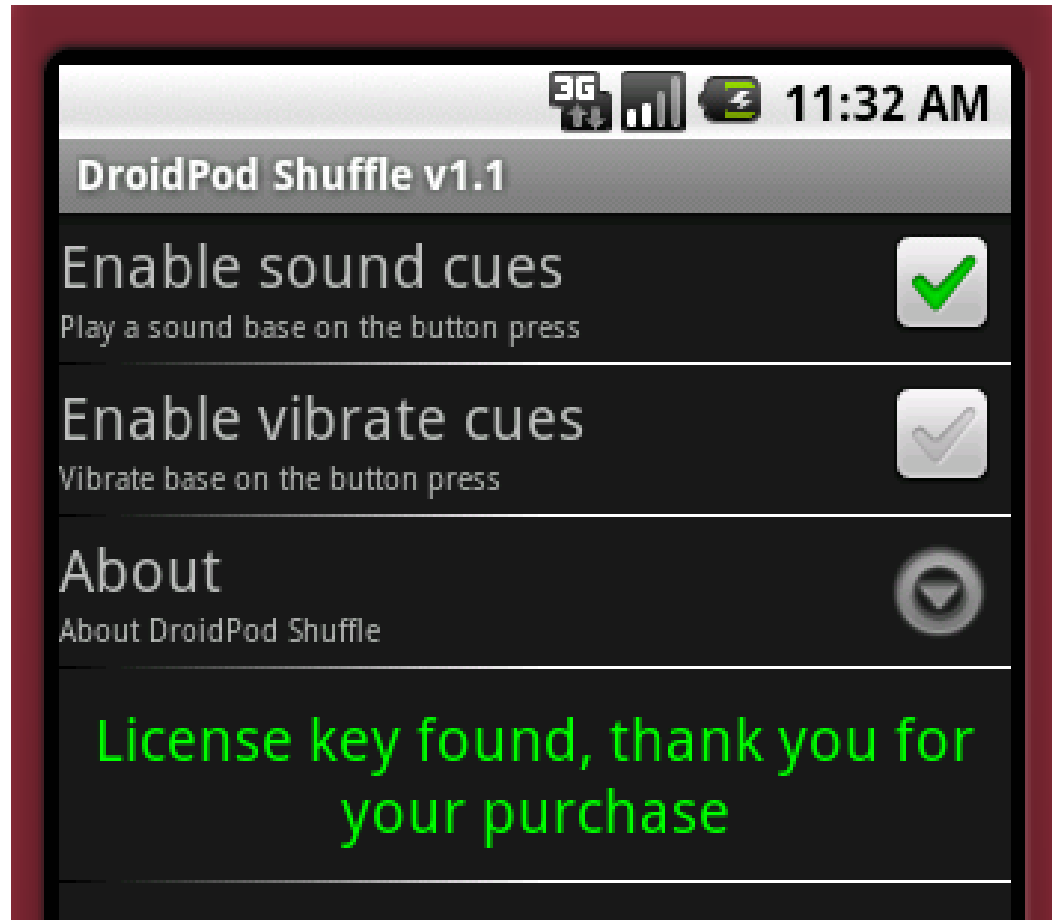
```
.method private static
check4key(Landroid/content/Context;)Z
    .locals 1
    const/4 v0, 0x1

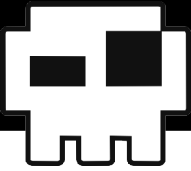
    return v0
.end method
```



Protections usuelles

🤖 Aperçu du résultat

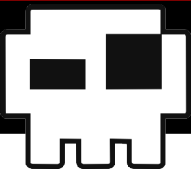




OfficeSuite

- ☞ Désassemblage avec apktool
- ☞ Dossier *registration*
 - Contient la classe *ActivationManager*
- ☞ Méthodes intéressantes
 - `setActivated()`
 - `isActivated()`
 - `CheckActivation()`

Analysons !



Activation Manager

isActivated()

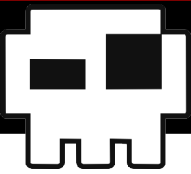
- Ouvre le fichier « .activated »
- Ferme ce fichier
- Si le fichier existe, alors l'application est « activée »

setActivated()

- Crée le fichier « .activated »

checkActivation()

- Vérifie une clef d'activation en ligne



Vérification en ligne

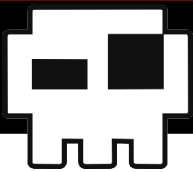
Contacte l'URL suivante

- [http://mobisystems.com/officesuitepro_keys2.php?imei=%1\\$s&key=%2\\$s](http://mobisystems.com/officesuitepro_keys2.php?imei=%1$s&key=%2$s)
- Récupération du numéro de série du téléphone
- Clef calculée en fonction du numéro de série

Accessoirement, donne l'IMEI à un service distant !

Méthode checkActivation()

- Seulement appelée sur saisie de la clef.



Contournement simple

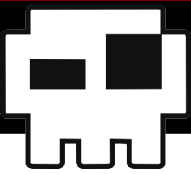
Réécriture de la méthode isActivated()

- Retourne toujours vrai (1)

```
.method public static
isActivated(Landroid/content/Context;) Z
    .locals 1
    .parameter "context"

    const/4 v0, 0x1

    return v0
.end method
```



Protections usuelles

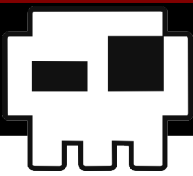
Contournement simple

- Modification de la méthode *IsRegistered* du fichier *SerialNumber.smali*
- Modification purement cosmétique :)

```
.method public IsRegistered() Z
    .registers 2

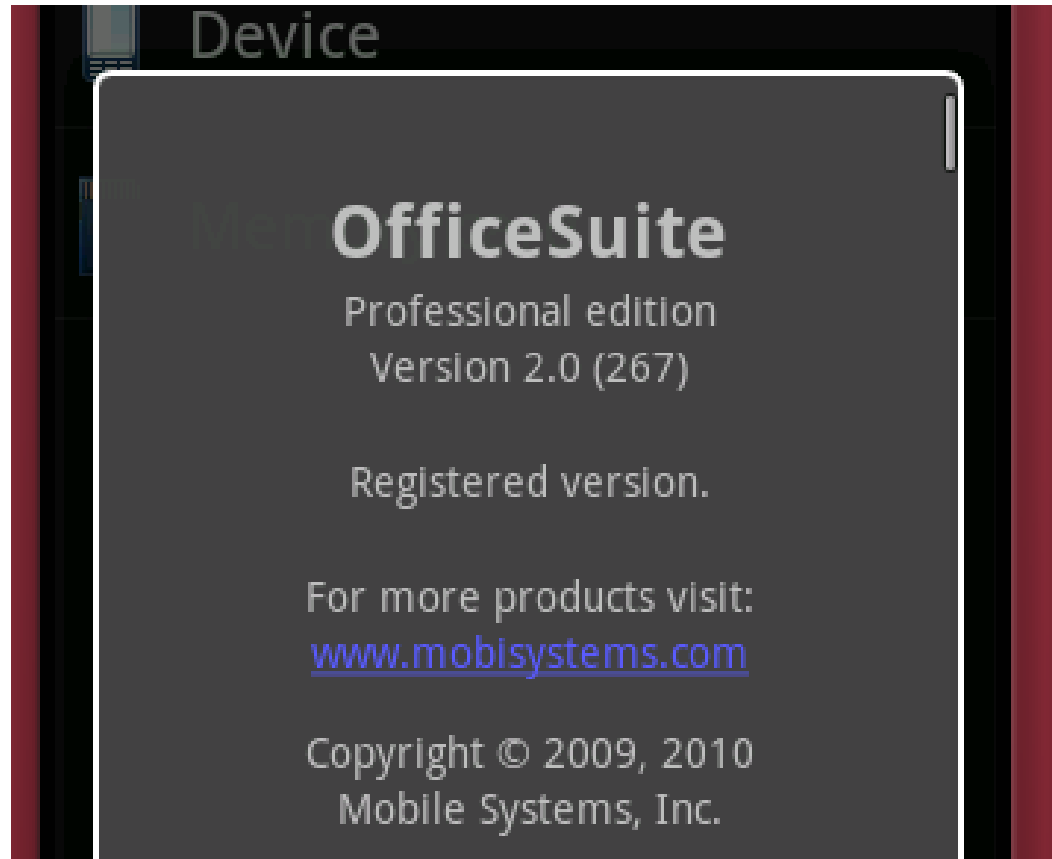
    .prologue
    .line 222
    const/4 v0, 0x1

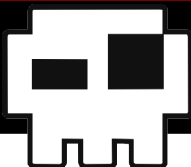
    return v0
.end method
```



Protections usuelles

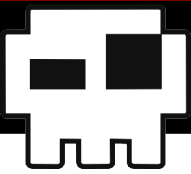
Office Suite







Constat

- ☞ Certaines protections exploitent un enregistrement en ligne
 - Génération de clef/licence
 - Vérification en ligne
 - Traçabilité des utilisateurs
- ☞ La grande majorité des vérifications effectuées par l'application sont aisément contournables
- ☞ Fuites d'information
 - IMEI, etc ...
 - En HTTP certaines fois ! (la plupart ?)





Protections usuelles

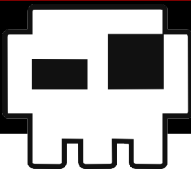
Remarques

-  Je n'ai pas désassemblé toutes les applications Trial du market
-  Ce ne sont que deux exemples parmi d'autres

Des études ont déjà été réalisées

-  Démontrent les fuites d'information
-  N'ont pas forcément évalué la solidité des protections

Il existe des solutions à ce problème



Plan (4/4)

Protections usuelles

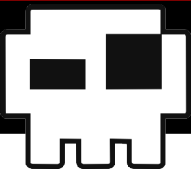
- 👉 Analyse de protections d'applications du market
- 👉 Evaluation du niveau des protections

Systèmes de protection

- 👉 Protectors android
- 👉 Google licensing
- 👉 Astuces anti-reverse

Pistes à suivre

- 👉 Manipulation de fichiers DEX
- 👉 Techniques de protection des applications



Nécessité

☞ Nous avons vu que

- Il est facile de désassembler des applications
- On peut retrouver quasiment le code original (Java)
- Analyser sans trop de problème le code

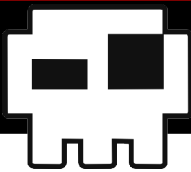
☞ Protection d'algorithme

☞ Protection de Propriété Intellectuelle

Techniques répandues

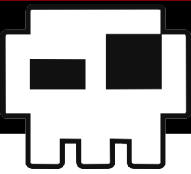
☞ Obfuscation des noms de méthodes

☞ Optimisation du bytecode



Obfuscation

- Modification des noms de champs, de méthodes et de classes (textuels)
- Le code ne fait référence qu'à des index de classes, de méthodes ou de champs
 - Aucune référence textuelle
 - Les textes sont simplement là à titre « informatif » (enfin, presque ...)
 - Les changer ne change foncièrement pas le fonctionnement du programme
- Visibilité Java
 - Deux packages différents peuvent avoir deux classes de même nom possédant des méthodes de même noms ...



Optimisation du bytecode

Suppression de code non-atteint

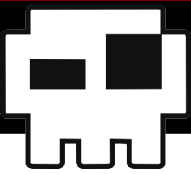
- Évaluation du code
- Détection des sections de code non-atteintes
- Suppression du bytecode

Effets de bord

- Peut supprimer du code atteint dynamiquement

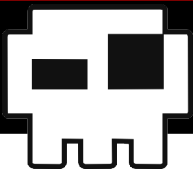
Objectifs

- Supprimer des informations inutilement présentes



Proguard

- ☞ Systeme fourni avec android
- ☞ Implémente l'obfuscation et l'optimisation de bytecode
- ☞ Configurable
 - Fichier proguard.cfg
- ☞ Intégré à l'environnement de développement Eclipse
- ☞ Mal documenté ...
- ☞ ... mais efficace !

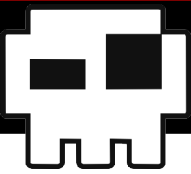


Androguard

- ☞ Projet OpenSource
- ☞ Vise à fournir un framework python de manipulation de DEX et d'APK
- ☞ Intègre un protector
 - Implémentation de l'obfuscation
 - Obfuscation des entiers
 - Formules mathématiques



<http://code.google.com/p/androguard/>

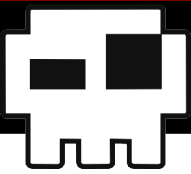


Google Licensing

- ☞ Framework permettant de gérer des licences
 - Développé par google
 - Bibliothèque appelée par l'application
- ☞ Permet
 - De vérifier/valider une licence
 - De dialoguer avec le Market (qui effectue les vérifications)
 - D'empêcher tout usage frauduleux

Inconvénients

- ☞ Dépend de l'implémentation faite par le développeur de l'application



Google Licensing (suite)

Contournement toujours possible

- Patch de l'application

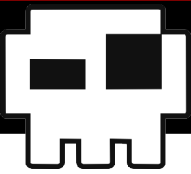
N'implémente pas

- Validation selon le checksum de l'application
- Blacklist de licence

Conclusion

Le système de licensing de Google reste faible

- Le système est aussi faible que son maillon le plus faible
- On peut toujours désactiver les vérifications dans l'application (patching)

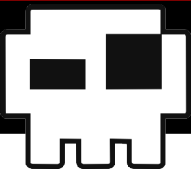


Astuces anti-reverse

- ☞ Effectuer de nombreux checks dans des endroits différents du code
- ☞ Encoder les chaînes de caractères
 - Pas forcément compatible avec la localisation
 - Plus difficile à mettre en œuvre

Ca va être dur de faire plus ...

- ☞ Android ne fournit qu'un accès limité au système
 - Impossible de bidouiller à partir de l'application en VM
 - Certains éléments restent inaccessibles bien qu'essentiels à l'anti-debugging

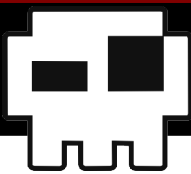


 ... mais une solution se profile

- ☞ Google a dévoilé son NDK
 - Native Development Kit
- ☞ Le NDK permet de créer des bibliothèques JNI
 - Fichier .so
 - Contient des méthodes implémentées en C/C++
- ☞ Possibilité d'intégrer un système de gestion de licence via le NDK

 Avantages

- ☞ Plus « dur » à reverser (mais ca reste possible)
- ☞ Moins exposé aux néophytes (archi. spéciale)



Plan (4/4)

Protections usuelles

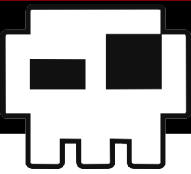
- 👉 Analyse de protections d'applications du market
- 👉 Evaluation du niveau des protections

Systèmes de protection

- 👉 Protectors android
- 👉 Google licensing
- 👉 Astuces anti-reverse

Pistes à suivre

- 👉 Manipulation de fichiers DEX
- 👉 Techniques de protection des applications

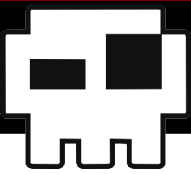


Manipulation de DEX

- ☞ Le format DEX est complexe et vaste
- ☞ Il y a certainement des manipulations possibles
 - Inspiration des manipulations du format PE
 - Possibilité de « cacher » des informations

Meilleure obfuscation de bytecode

- ☞ La manipulation du bytecode DEX en est à ses débuts
- ☞ Il y a certainement moyen d'atteindre des niveaux de protection identiques aux protectors d'exe



Bibliographie

 Opcodes Dalvik

 http://pallergabor.uw.hu/androidblog/dalvik_opcodes.html

 Format DEX

 <http://www.netmite.com/android/mydroid/dalvik/docs/dex-format.html>

 Apktool

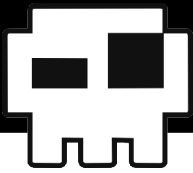
 <http://code.google.com/p/android-apktool/>

 Dex2jar

 <http://code.google.com/p/dex2jar/>

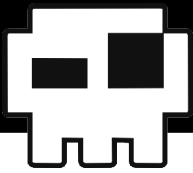
 Site developpeur d'Android

 <http://developer.android.com/index.html>



Questions

Questions ?



Remerciements

Heurs, Trance, et ceux ayant testé et analysé des applis Android :)