Writeup Wargame NcN 2012 – Pau Oliva Fora

Website: http://pof.eslack.org

Email: <pof@eslack.org>

Twitter: @pof

1) Lanzamos el emulador, con la opcion `-tcpdump` para guardar un pcap y poder analizar el tráfico posteriormente. Por precaución lo hacemos con una máquina sin salida a internet.

```
$ emulator -tcpdump capture5.pcap -avd Wargame
```

2) Ejecutamos '`adb shell`' y obtenemos shell de root en el teléfono, por los timestamps de los archivos en /data/data/app vemos los paquetes que se han instalado: shazam, instagram, angry birds, skype y finalmente, nos llama la atención la aplicación 'pimp my app', asi que decidimos descargarnos el APK:

```
$ adb pull /data/app/com.pimpmyapp-1.apk
```

3) usamos `apktool`, para extraer el AndroidManifest.xml, vemos que los permisos que requiere la aplicación parecen muy sospechosos: (`RECEIVE_SMS, READ_SMS, WRITE_SMS, SEND_SMS, READ_CONTACTS`, etc...) además tiene una "activity" llamada `SmsTrackerActivity` que se activa cada vez que se recibe o se manda un mensaje... vamos a investigar más!

```
$ apktool d com.pimpmyapp-1.apk
```

4) Usamos `dex2jar` y `jad` para obtener el código java de la aplicación, analizando las clases podemos hacernos una idea de su comportamiento, a grandes rasgos la aplicación guarda el contenido de la tarjeta SD en un archivo ZIP cifrado, los datos del IMEI, numero de teléfono, operador, etc en otro archivo cifrado y el contenido de los SMS en un tercer archivo cifrado. Cuando se recibe el comando 'MASTER' a través de SMS estos archivos se mandan a un servidor FTP usando las siguientes credenciales:

```
    private static final String FTP = "176.58.123.34";
    private static final String PASS = "penacolada";
    private static final String USER = "cocomoco";
```

La clave para cifrar los archivos se obtiene haciendo una petición HTTP al recurso /data del mismo servidor que devuelve un array codificado en JSON.

5) Ejecutamos la aplicación y vemos que salta un stack trace... algo no funciona, damos salida a internet al emulador, y sigue sin funcionar, intentamos forzar la activity mandando un SMS (`telnet localhost 5554; sms send 31337 hola`) y tampoco funciona. Con '`adb logcat`' vemos que el stacktrace es debido a que no puede parsear el json, ya que el servidor HTTP no está funcionando :(

6) Hacemos un `nmap` del servidor y vemos que el puerto SSH está abierto, y para nuestra sorpresa el mismo usuario y password del ftp nos dan una shell en el sistema via SSH... una VM con Debian 6.0.3 en Linode }:) lamentablemente no hay ni rastro del servidor web, ni del json que debiera mandar, aunque vemos algún *problemilla* con los permisos ya que la carpeta `/srv/ftp` no debería ser world writeable :P

7) Como el servidor HTTP está caído, tendremos que modificar el APK para que apunte a un servidor nuestro, donde colocaremos en `/data` un archivo JSON igual al que espera la aplicación, con esto conseguimos que la aplicación funcione y nos muestre una "awesome face"!

8) En el servidor FTP hemos encontrado archivos que pertenecen a otros terminales, supuestamente infectados con el mismo malware, así que suponemos que ahí encontraremos la respuesta a las preguntas del wargame... pero de momento no sabemos nada de PAULA, así que habrá que investigar más! Con nuestro JSON los archivos generados por el malware se codifican en RC4 con la clave que nosotros mismos hemos indicado, vemos que el archivo más pequeño (el que contiene el Device ID, PhoneNumber...) podemos decodificarlo con nuestra clave así que suponemos que el archivo con el mismo nombre, y el mismo tamaño que hay en el servidor FTP contendrá los mismos datos, ya conocemos el plaintext y el cyphertext, pero no conocemos la clave.

9) Como RC4 es un cifrado de clave simetrica que funciona por stream (flujo de datos), haciendo una simple XOR del plaintext y el cyphertext obtendremos parte del keystream, lamentablemente el archivo que tenemos es muy pequeño (140 bytes) y no nos servirá para decodificar todos los archivos del servidor FTP. Hacemos la prueba, y vemos que efectivamente podemos decodificar correctamente los primeros 140 bytes de cualquier archivo. Con esto, obtenemos parte de los SMS y vemos que nuestra querida PAULA tiene el identificador 069!

Content: HOLA **PAULA**, NO RESPONDISTE MIS SMS. PASA ALGO?
Content: HOLA **PAULITA** QUE TAL ESTAS? ME GUSTO MUCHO VERTE.

10) Para obtener el keystream entero, vamos a centrarnos en el archivo zip más grande de todas las muestras (4d7673dd1f336fe6e0efcb4e64e52b31.000) que corresponde al /sdcard/tmp.zip que el malware elimina (si no está en modo debug). Así que volvemos a editar el APK, para que no elimine el archivo ZIP después de cifrarlo.

11) Usamos la utilidad TestDisk[1] para recuperar los archivos borrados del sdcard.img original:

```
drwxr-xr-x     0     0         0  9-Oct-2012 15:07 LOST.DIR
drwxr-xr-x     0     0         0  9-Oct-2012 15:07 .android_secure
-rwxr-xr-x     0     0   1322673  9-Oct-2012 18:28 tmp.zip
drwxr-xr-x     0     0         0  9-Oct-2012 18:01 DCIM
-rwxr-xr-x     0     0    444448  9-Oct-2012 17:53 when_im_waiting_for_the_connect_back.gif
-rwxr-xr-x     0     0    881427  9-Oct-2012 17:55
when_you_find_too_much_data_on_the_client_side_during_a_web_audit.gif
-rwxr-xr-x     0     0       140  9-Oct-2012 18:28 2ac4b9fd2e51e170994554297afb946.000
-rwxr-xr-x     0     0   1322673  9-Oct-2012 18:28 4d7673dd1f336fe6e0efcb4e64e52b31.000
-rwxr-xr-x     0     0      2545  9-Oct-2012 18:33 3a944862bf026179fb14d1517ff9caa.000
-rwxr-xr-x     0     0    881427  9-Oct-2012 17:55 when_you_find_too_much_dat
```

12) Lamentablemente el fichero tmp.zip está corrupto, y sólo podemos obtener los primeros ~1500 bytes del fichero original, el resto se ha sobre escrito con el contenido de otros archivos... pero podemos hacer también undelete de los archivos GIF que contiene el ZIP, así que vamos a intentar generar el trozo del archivo ZIP que nos falta!

1 http://www.cgsecurity.org/wiki/TestDisk

13) Copiamos los archivos GIF recuperados en /sdcard/ usando '`adb push`', y ejecutamos la aplicación 'pimp my app' modificada para que genere el zip y no lo borre. Obtenemos el archivo ZIP con '`adb pull`'.

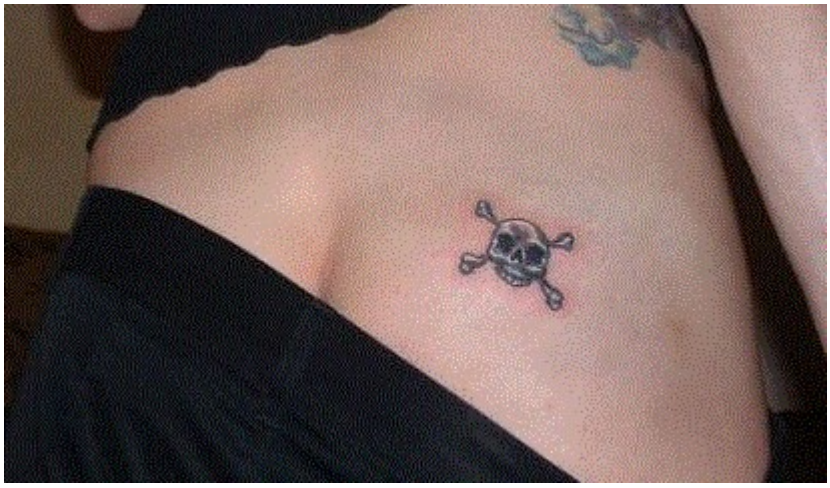
14) Aprovechamos los primeros 1500 bytes del fichero ZIP original que obtuvimos con undelete, y con la ayuda de la especificación del formato ZIP[2] modificamos con un editor hexadecimal el ZIP que hemos generado para que los archivos contenidos tengan los mismos timestamps que los originales, es importante hacer el `adb push` del paso anterior en el mismo orden que se habían copiado los GIF originalmente, ya que si no ordena los GIF del revés dentro del zip y queremos un ZIP que se parezca lo máximo posible al original. Finalmente, probamos que el ZIP generado con el editor muestra los mismos timestamps y el mismo orden que los ficheros originales, recalculamos el checksum del ZIP y *voilà*, hemos conseguido un ZIP igual que el original que se usó para generar el fichero 4d7673dd1f336fe6e0efcb4e64e52b31.000 del FTP!

15) Hacemos una XOR de nuestro fichero ZIP y el fichero .000 cifrado del FTP, con esto obtenemos 1,3Mb de keystream, que nos servirá para poder descifrar el resto de ficheros, ya que todos tienen un tamaño menor.

16) Finalmente, vamos a por PAULA... en los ficheros *.069 descifrados encontramos la respuesta a las preguntas:

    ¿Cómo se llama el perro de Paula?
    EL PERRO DE PAULA SE LLAMA “JOSE CARLOS”

    ¿Qué lleva Paula tatuada en el trasero?
    UNA CALAVERA!



2 http://www.pkware.com/documents/casestudies/APPNOTE.TXT