DEPARTAMENTO DE COMPUTACION

Facultad de Ciencias Exactas y Naturales - UBA

Trabajo Práctico: Malito

La Llama que no llama

Seguridad de la Información

Segundo Cuatriméstre de 2013

Grupo "La barra de Platense"

Integrante	LU	Correo electrónico
Julián Sackmann	540/09	jsackmann@gmail.com
Juan Pablo Darago	272/10	jpdarago@gmail.com
Vanesa Stricker	159/09	vanesastricker@gmail.com

Facultad de Ciencias Exactas y Naturales

Universidad de Buenos Aires

Ciudad Universitaria - (Pabellón I/Planta Baja)

Intendente Güiraldes 2160 - C1428EGA

Ciudad Autónoma de Buenos Aires - Rep. Argentina

Tel/Fax: (54 11) 4576-3359

http://www.fcen.uba.ar

Índice

1. Primer Ejercicio

La primer parte del TP requiere implementar un *malware* para `Android` que, al instalarse, envíe las fotos del teléfono a las que tenga acceso a un servidor controlado por nosotros. Para esto, es necesario tomar una serie de decisiones, que se detallan a continuación:

1.1. Façade

Si bien no tenemos datos certeros de esto, creemos que es extremadamente poco probable que el usario promedio de `Android` instale voluntariamente una aplicación que se publicite a si mismo como "TeRoboLasFotosApp". Es por esto que consideramos necesario **disfrazar** el app, de tal forma que la probabilidad de una instalación se incremente ligeramente.

Aprovechando esta necesidad de disfrazar el app, nos propusimos cumplir uno de nuestros sueños y realizar la façade de este app siguiendo el diseño general de un *soundboard*. Este tipo de aplicaciones tienen como objetivo el de presentar al usuario con una serie de botones, cada uno de los cuales emite un sonido al ser presionado. En general, estas aplicaciones ofrecen la funcionalidad de setear estos sonidos como ringtones de mensajes o llamadas.

En nuestro caso decidimos instanciar el modelo de soundboard en el viejo y querido **La Llama Que Llama** en su nueva versión: La Llama que **no** Llama.

1.2. Information Upload

Otra decisión que se tuvo que tomar a la hora de implementar el robo de imágenes era la concerniente a la forma de transmisión de esas imágenes al servidor controlado por nosotros.

Consideramos las siguientes opciones:

- Email
- `POST`
- `scp`

Principalmente por facilidad en su implementación, así como compatibilidad con la mayor parte de celular posibles[1], elegimos el método `POST`.

La implementación es batante sencilla, utilizando solamente una petición HTTP multipart para enviar los archivos. Esto se ve complicado por la falta de soporte nativo en Android, para lo cual utilizamos las librerías de "Apache". El upload por el momento solo incluye el DCIM en la tarjeta SD, que es donde se almacenan por *default* las fotos de la cámara del celular. Para evitar subir archivos innecesariametne, utilizamos un *digest* de SHA-1 del contenido de la imágen que enviamos al servidor antes de enviar la imágen, de manera de poder así saber si una foto fue enviada ya o no. Esto evita subidas innecesarias y acelera la aplicación. También, en el momento actual, la aplicación lanza un *thread* separado por cada foto. Estamos investigando alternativas a esto que permitan disminuir el tiempo esperado de respuesta de manera de que la aplicación realice su tarea de la manera menos sospechosa posible.

[1] `scp` sería mucho más fácil de implementar, pero sólo funcionaría con teléfonos `Android` que tengan `BusyBox` instalado, lo que limita mucho el alcance posible del ataque.

1.3. Servidor

Del lado del servidor se implementó un script de PHP que se ocupa de recibir los `POST`, validarlos y almacenar la imagen.

1.3.1. Validación

En primer lugar, dado que el servicio de PHP está ubicado en un servidor de acceso público es imprescindible realizar algún tipo de validación de que las solicitudes de `POST` que llegan sean efectivamente enviadas por la aplicación y no por un tercero. Es por esto que se incluyó en el código de la aplicación una clave secreta con la que se hashea el la imagen. Este hash es replicado oportunamente por el servidor y, en caso de no coincidir, el `POST` es descartado.

1.4. Envío Diferencial

Uno de los principales problemas que tuvimos al testear la aplicación era que, en un principio, se subían todas las fotos todo el tiempo. Si bien esto no es un problema del lado del servidor (simplemente se pisaban los archivos, no se generaban duplicados), genera mucho tráfico de red inútilmente. Es por esto que decidimos utilizar un sistema de envío diferencial de imágenes.

Consideramos dos métodos para realizar esto:

- Que la aplicación almacene en el mismo teléfono la lista de las imágenes ya subidas (por ejemplo, aprovechando los servicios de SQLite).
- Que la aplicación consulte con el servidor qué imágenes debe mandar.

Tanto por simplicidad de implementación como para evitar dejar rastros en el teléfono, decidimos implementar la segunda. Además, tiene la ventaja de que si el usuario desistanala la aplicación (o elimina sus datos locales desde el menú de Androdid) no vuelven a enviarse todas las fotos.

Como desventaja de esto, se puede mencionar que el teléfono debe realizar más transimisones por la red (aunque muy pequeñas), que podrían ser detectadas por un *sniffer*.

La implementación actual de esta característica está hecha de la siguiente manera: para cada foto a la que tiene acceso, la aplicación envía un `POST` al servidor con el hash **SHA1** de la imagen en cuestión. Cuando el servidor lo recibe, verifica si alguna de las imágenes que tiene guardadad tiene ese mismo **SHA1**. En el caso de no encontrarla, el servidor responde al `POST` con el mensaje "OK", tras lo cual la aplicación envía la imágen mediante otro `POST`. En caso contrario, la respuesta es "FOUND" (y, obviamente, la aplicación no envía la imagen).

1.5. Mejoras

- Hacer un solo request para las fotos en vez de muchos.
- Almacenamiento ordenado por IMEI.
- Encriptar las fotos que van por la red (no queremos darle 100 años de perdón a nadie).
- Poder setear sonido como ringtone.

1.5.1. Almacenamiento Ordenado

Otro de los problemas que tuvimos a la hora de utilizar la aplicación surgió a la hora de subir fotos de distintos teléfonos. Este problema se presentó en dos formas:

- Cuando se subía una imagen de un celular y ya había otra con otro nombre: por defecto, las fotos sacadas con un sólo teléfono no se almacenan con el mismo nombre. Sin embargo, si puede ocurrir (y nos pasó) que dos fotos sacads con distintos teléfonos se llamen igual. Cuando se subieron al servidor, la segunda pisó a la primera.
- Cuando se subía mucha cantidad de imágenes de muchos teléfonos, se volvió un desorden importante la carpeta del servidor donde se almacenaban las imágenes.

Si bien el primer problema se puede resolver simplemente renombrando un archivo si hay una colisión, la solución al segundo problema también solucionaba el primero: se pensó en añadir a cada mensaje enviado por la aplicación un identificador del teléfono (`IMEI`). Cuando el servidor recibe una imagen, la almacena en la carpeta correspondiente a ese teléfono. De esta forma queda asociada cada imagen al teléfono del que fue subido (suponiendo que el `IMEI` que reporta el sistema operativo a la aplicación es válido) y el servidor queda más ordenado.