Índice

[Introducción 3](#__RefHeading__443_1049142930)

[Primer ataque: Extrayendo todas las imágenes de la SD. 3](#__RefHeading__374_1097190633)

[Objetivo 3](#__RefHeading__376_1097190633)

[Dos aplicaciones trabajan en conjunto: motivación 4](#__RefHeading__378_1097190633)

[Primera parte: Extrayendo imágenes de la SD 4](#__RefHeading__380_1097190633)

[Interfaz de la aplicación 4](#__RefHeading__382_1097190633)

[Cómo obtenemos y enviamos las imágenes a la otra aplicación 5](#__RefHeading__451_1049142930)

[Segunda parte: Recibo y envío de las fotos al servidor 6](#__RefHeading__474_1049142930)

[Interfaz de la aplicación 6](#__RefHeading__476_1049142930)

[Cómo recibimos y envíamos las imágenes al servidor. 6](#__RefHeading__478_1049142930)

[Resumen 6](#__RefHeading__480_1049142930)

[Segundo ataque: Remote Access Tool (RAT) 8](#__RefHeading__482_1049142930)

[Objetivo 8](#__RefHeading__651_1524953640)

[Interfaz de la aplicación 8](#__RefHeading__655_1524953640)

[Funcionalidades y Permisos 8](#__RefHeading__657_1524953640)

[Geolocalización 8](#__RefHeading__659_1524953640)

# Introducción

La utilización de la tecnología móvil ha incrementado notoriamente en estos últimos años. La comodidad que brindan los dispositivos móviles para realizar cualquier tipo de acción, hace que las personas dependan más y más de sus móviles, llegando a una dependencia jamás antes pensada.

Esto no sólo ocurre con la gente que entiende la tecnología, sino que, ante la facilidad de uso que brindan estos aparatos, las personas que no nacieron en la “época de la tecnología” también puedan rápidamente adaptarse a ellos y utilizarlos como cualquier otro.

El problema surge cuando las mismas personas que crean dicha depencia casi única entre la persona y su dispositivo (guardando información muy sensiblel), desconocen las potenciales implicancias de instalar una aplicación en él.

La gran mayoría de las personas cuando instala aplicaciones en sus dispositivos, no revisa los permisos que requiere la aplicación, aceptando cualquier conjunto de permisos mientras que la funcionalidad de dicha aplicación (o lo que la aplicación dice hacer) satisfaga sus necesidades.

En el siguiente trabajo, presentaremos dos ataques distintos basados en la ejecución de aplicaciones maliciosas. Para el primero, mostraremos cómo dos aplicaciones pueden trabajar en conjunto para extraer información del dispositivo requiriendo los persmisos mínimos. Para el segundo, cómo una aplicación nos engañará diciendo que hace cierta tarea, cuando, en realidad, una vez instalada, comienza a hacer otro tipo de cosas.

# Primer ataque: Extrayendo todas las imágenes de la SD.

## Objetivo

El objetivo de este primer ataque era el de poder extraer toda la imágenes que estuvieran prensentes en la SD del dispositivo y enviarlas a un servidor controlado por nosotros. Esta tarea, cómo mínimo, requiere dos tipos de permisos distintos:

* Acceso de lectura a la SD.
* Acceso a internet

La primer idea que surgió en nuestro equipo fue la de crear una aplicación que simplemente pidiera esos dos permisos e hiciera el trabajo por sí sola. Luego de discusiones, llegamos una conclusión por unanimidad: Si queremos que hasta los **MUY** cuidadosos caigan en nuestra trampa, teníamos que elegir una aplicación que tuviese muy buenos fundamentos para requerir ambos permisos y, aún así, la persona atenta podría rechazarla por el sólo hecho de los tipos de permisos que requería (sin importar qué hiciera).

Pero nosotros queríamos que esas personas también pudieran caer en nuestra trampa.

## **Dos aplicaciones trabajan en conjunto: motivación**

Luego de una etapa de investigación, nos encontramos con una forma de esconder estos permisos. No podíamos dejar de pedirlos ya que eran necesarios para llevar a cabo el objetivo, pero sí podíamos esconderlos.

La solución que encontramos fue, dividirlas aplicaciones pueden crear servicios públicos accessibles por cualquier aplicación instalada en el mismo dispositivo. ir las tareas en dos aplicaciones distintas. Encontramos que en Android, Por lo tanto nuestro enfoque cambió radicalmente. Ya no teníamos una sóla aplicación que requeriese ambos permisos, sino que serían dos aplicaciones, una que pidiese el permiso de lecutra a la SD y otra el permiso de acceso a internet. De esta forma, los fundamentos que deberíamos exponer para cada aplicación se verían simplificados, y además la relacion entre las aplicaciones sería **CASI**  transparente para el usuario.

Por lo tanto, la “Extracción de imágenes de la SD” se dividió en dos aplicaciones que llevararán cabo tareas distintas pero con un fin en común:

* Recolección de las imágenes SD (primera aplicación)
* Envío al servidor (segunda aplicación)

## Primera parte: Extrayendo imágenes de la SD

Debíamos elegir una aplicación que tuviera los fundamentos necesarios para solicitar el permiso de acceso a la SD. Lo que nos pareció más lógico fue crear una aplicación que se llamase 'Cámara', que actúe como tal pero agregando algo extra (aplicando filtros a las imágenes tomadas con esta aplicación). Sin embargo, un dato no menor, es que nuestra aplicación no pediría el acceso a la cámara del dispositivo. ¿Cómo logramos eso? Hacemos uso de la aplicación de la cámara nativa del dispositivo. Es decir, el usuario abre nuestra aplicación y elige la opción 'Tomar Foto'. Esto inmediatamente abre la aplicación 'Camara' nativa del dispositivo. Una vez el usuario saca la foto, internamente esa imagen es enviada a nuestra aplicación, la cual aplica los filtros que nosotros definimos.

### Interfaz de la aplicación

### Cómo obtenemos y enviamos las imágenes a la otra aplicación

**¿Cómo obtenemos las imágenes?** Esta quizás es la parte más fácil. Dado que ya tenemos en nuestras manos el poder de acceder a la SD, no nos queda mucho trabajo por hacer. Desde el primer momento que nuestra aplicación se instala en el dispositivo, comienza a recorrer la tarjeta SD en busca de archivos de imágenes. Para realizar esta tarea se utilizaron estructuras nativas de Android llamadas **Content Providers**, las cuales tienen por objetivo suministrar una forma de acceso a datos que están estructurados de cierta forma. Existen subdivisiones dentro de los **Content Providers** que refinan sobre el tipo de archivos que uno quiere acceder. Para lo que nosotros queríamos, utilizamos el **MediaStore.Images.Media** como **Content Provider**, ya que era lo más ajustado a nuestra necesidad.

La siguiente pregunta entonces es: **¿Cómo y a dónde envíamos cada imagen?**

Por cada imagen que obtenemos de la SD, la convertimos en un bitmap a través del uso de la librería **BitmapFactory**. Como ya adelantamos en la explicación general, deberá existir un servicio en la aplicación que se ocupará de la segunda parte, el cual será el encargado de recibir todos estos bitmaps. Para el envío del bitmap utilizamos **intent:** una objeto que tiene muchos tipos de funcionalidades, y entre ellas se encuentra la de poder enviar objetos encapsulados dentro de él. Por lo tanto, el bitmap es puesto dentro de este **intent,** que luego es enviado al servicio público mencionado anteriormente.

Sin embargo, el envío de estos bipmaps no se hacen constantemente. Creímos que, enviar una imágen detrás de la otra podría ocasionar problemas de lentitud (por ejemplo) en el dispositivo que serían rápidamente detectados por el usuario y harían que éste sospeche del accionar de la aplicación.

Dos cosas se tuvieron en cuenta en este proceso:

* **La presencia del servicio: ¿Qué ocurre si la segunda aplicación aún no está instalada y, por ende, el servicio no existe?**

Creímos que lo mejor era que el thread ocupado de hacer este paso, se duerma hasta por un tiempo estipulado, y una vez que se despierte vuelva a hacer la misma pregunta. De esta forma, nos aseguramos de mandar las imágenes solamente cuando el servicio está activo.

* **El tamaño de las imágenes**

Sabemos que algunas imágenes podrían ser muy grandes en cuanto a su tamaño y por lo tanto muy pesadas. Teniendo en cuenta esta problemática decidimos escalar las imágenes justo antes de ser enviadas con el fin de hacer el trabajo menos trabajoso y menos cargoso para el procesador del dispositivo.

Es importante destacar que esta tarea será llevada a cabo por más que el usuario esté usando o no nuestra aplicación. **NO** hará falta que el usuario saque, ni siquiera, una foto para comenzar con el robo de las imágenes presentes en la SD. Sin embargo, tomamos una decisión con respecto a las imágenes que son tomadas desde nuestra aplicación: les dimos más prioridad, y por lo tanto, éstas serán enviadas inmediatamente luego de que el usario haya sacado la foto. Por más que el usuario, desde nuestra aplicación, vea una imagen con un filtro aplicado, en realidad la aplicación envió la versión original de dicha imagen al servicio antes mencionado.

## Segunda parte: Recibo y envío de las fotos al servidor

La segunda parte consiste en poder enviar las imágenes recibidas a un servidor externo en algún lugar de Internet. Para esta parte debíamos hacer una aplicación que tuviera los suficientes fundamentos como para requerir el acceso a Internet. Siempre es fácil conseguir una excusa para esto, ya que hoy en día es muy común tener este privilegio. Nuestra idea fue hacer una aplicación que simulase una 'Pokédex'[[1]](#footnote-2). Como mencionamos anteriormente, la aplicación sólo haría de Pokédex, con el fin de engañar al usuario ya que por dentro haría la comunicación/envío de las imágenes al servidor. Cómo servidor, decidimos la utilización de una servidor FTP ya que era lo más ajustado a nuestras necesidades. Esta aplicación, será la encargada de instalar el servicio (tan mencionado) que estará a la escucha de nuevas imágenes.

### Interfaz de la aplicación

### Cómo recibimos y envíamos las imágenes al servidor.

Esta aplicación será la encargada de crear un servicio que estará siempre a la escucha y será el encargado de recibir las imágenes enviadas por la primer aplicación.

Cada vez que el servicio reciba un **intent**, conteniendo una imagen para enviar, éste creará una conexión con el servidor FTP configurado y procederá al envío de la misma. Tanto el inicio de la conexión con el servidor FTP, como la autenticación, el proceso de envío y el proceso de cierre de conexión fueron implementados con la librería de Android llamada **FTPClient.**

Para cada envío de imágen al servidor se realizan las siguientes acciones:

* Inicio de conexión contra el servidor en cuestión.
* Autenticación.
* Creación un directorio en el servidor (si todavía no existe) con el modelo del teléfono en cuestión. Si bien esto podría generar problemas a la hora de dos dispositvos del mismo modelo, creímos que con esa división ya era suficiente.
* Elección de un nombre random para cada una de las imágenes guardadas en la carpeta antes creada.
* Proceso de envío de la imagen.
* Cierre de conexión.

## Resumen

Ahora que sabemos bien cómo funciona cada una de las aplicaciones, podemos hacer un resúmen general de cómo es el procedimiento para el ataque, entrando un poco en detalle:

Una vez que el usuario instaló ambas aplicaciones, el servicio de las aplicación de 'Pokédex' estará levantando y esperando por nuevos intents que serán envíados por la aplicación 'Cámara'. Esta última aplicación enviará un intent cada un tiempo razonablemente fijado (cinco segundos en nuestro caso). Cada vez que el servicio reciba ese intent, creara una conexión FTP y alojará el bitmap correspondiente en el servidor antes configurado.

A su vez, las aplicaciones podrán ser utilizadas para lo que el usuario cree que es su tarea principal (sacar fotos y aplicar filtros, obtener información sobre cada pokemon a través de HTTP requests a la API).

El gráfico que presentamos a continuación describe de una forma más cómoda el flujo que siguen las aplicaciones para realizar el ataque esperado:

A description...

# Segundo ataque: Remote Access Tool (RAT)

## Objetivo

La intención de este segundo ataque consiste en la creación de un RAT (Remote Access Tool). ¿Qué es eso? Un RAT es un fragmento de código que permite el acceso o control remoto del dispositivo donde esté instalado. Nuestra finalidad, en particular, es tomar el control de los dispositivos de los usuarios que instalen la aplicación, conectándose en forma remota y enviándole instrucciones para ejecutar código malicioso.

Es notorio que conforme aumentan las capacidades de la aplicación maliciosa mayores deberán ser los permisos que requiera ya que necesitaríamos acceso a subsistemas críticos del dispositivo como podrían ser los de geolocalización, memoria, micrófono, etc. Por lo cual, si dichos permisos no estuvieran debidamente justificados entonces los usuarios podrían sospechar de las funcionalidades que proveemos. ¿Qué tipo de aplicación requiere dicha magnitud de permisos?

No fue fácil encontrar la justificación, debido a, como ya mencionamos anteriormente, la cantidad y calidad de los permisos requeridos. Si bien se nos ocurrieron algunas opciones que podrían haber funcionado, optamos por una aplicación de control parental.

¿Por qué creímos que era una buena justificación?

Hoy en día la seguridad es algo realmente preocupante sobre todo cuando de niños y adolescentes se trata. No sólo hablamos en términos informáticos, sino que también en la vida diaria. Cada día es más frecuente ver un adolescente con un dispositivo propio, producto de la exploción tecnológica que vivimos hoy en día. Ante esta situación, muchos padres optan por controlar dichos dispositivos, no sólo para evitar que accedan a contenidor peligrosos en internet, sino también para controlar su ubicación o qué llamadas realizó, por ejemplo.

La característica de tener acceso total al dispositivo es lo que buscábamos. Si bien esta aplicación es simplemente un mock-up, no realiza ninguna funcionalidad concreta, nada impide su posterior desarrollo y cumple la función de justificar los permisos necesarios.

En resumen, nuestra aplicación se conectará a un servidor en busca de órdenes. Una vez recibida dicha órden, se procederá a llevarla a cabo para más tarde enviar el resultado a un otro servidor.

## **Interfaz de la aplicación**

## ¿Cómo funciona?

Como ya mencionamos anteriormente, un RAT es un fragmento de código con el fin de controlar remotamente un dispositivo. ¿Cómo hacemos para hacer ese control?

Utilizamos dos servidor distintos:

* FTP: Utilizado para alojar el resultado una vez ejecutada la funcionalidad.
* HTTP: Utilizado para alojar la funcionalidad que se llevará a cabo.

Una vez instalada la aplicación o en su defecto iniciado el dispositivo, se crea un servicio que tiene como objetivo conectarse al servidor HTTP para recibir instrucciones (si bien esto podría implementarse de diversas formas nosotros optamos por guardar un archivo (“**command.txt”)** que expresará qué instrucción llevar a cabo. Dicho archivo tendrá un formato especial para cada intrucción (explicado en detalle más adelante). Con un parámetro númerico (fijado en cinco segundos) controlamos cada cuanto tiempo la aplicación preguntará por el contenido de dicho archivo. Cabe aclarar que para hacer una consulta HTTP utilizamos *Retrofit*[1], una API Rest que nos facilitó esta tarea.

Luego de descargada, la instrucción es parseada y ejecutada. Una vez obtenido el resultado, que dependiendo de qué funcionalidad fue ejectutada puede variar el formato radicalmente, es enviado al servidor FTP. Por cada dispositivo, se creará un directorio dentro del servidor FTP para guardar todos los archivos (de formatos variados), resultado de la ejecución de cada instrucción.

El manejo del archivo **“command.txt”** es de vital importancia. Ya que además de cambiar entre funcionalidad y funcionalidad, también determinará cuando el dispositivo deberá hacer **nada.** Esto se logrará poniendo un nombre de funcionalidad inválido (o simplemente dejándolo en blanco).

## **Funcionalidades y Permisos**

Antes de comenzar con la explicación de cada una de las funcionalidades, el respectivo formato que tendrá que respetar el **“command.txt”** para ejecutarla y los permisos necesarios, debemos mencionar algo transversal a cualquier instrucción.

Independiemente de qué funcionalidad se vaya a ejecutar, la aplicación necesitará conectarse al servidor HTTP. Por lo tanto, el primer permiso, transversal a todas las funcionalidades, que será requerido será: **Acceso a internet**.

### Geolocalización

### **Grabación de voz**

### **Vibración**

### **Listado de llamadas**

### **Descarga de binario**

1. http://es.pokemon.wikia.com/wiki/Pokédex [↑](#footnote-ref-2)