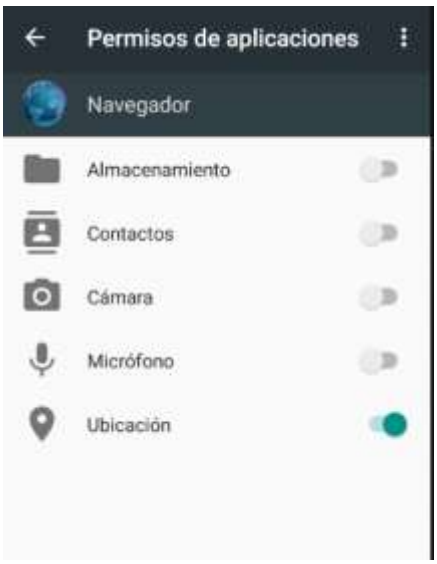
Permisos en Android 6 Marshmallow

En un dispositivo con versión de Android anterior a Marshmallow un usuario concede los permisos a una aplicación en el momento de la instalación. Si no está de acuerdo con algún permiso, la única alternativa para el usuario es no instalar la aplicación. Una vez instalada la aplicación, puede realizar las acciones asociadas a estos permisos tantas veces como y cuando desee. Esta forma de trabajar dejaba a los usuarios indefensos ante posibles abusos. Por ejemplo, si queremos utilizar WhatsApp o jugar a Apalabrados tenemos que aceptar la larga lista de permisos innecesarios que nos solicitan. El usuario se resigna y acaba aceptando prácticamente cualquier permiso.

En la versión 6 se introducen importantes novedades a la hora de conceder los permisos a las aplicaciones. En primer lugar los permisos son divididos en **normales** y **peligrosos**. A su vez los permisos **peligrosos se dividen en 9 grupos**: almacenamiento, localización, teléfono, SMS, contactos, calendario, cámara, micrófono y sensor de ritmo cardíaco. En el proceso de instalación el usuario da el visto bueno a los permisos normales, de la misma forma como se hacía en versiones anteriores. Por el contrario, los permisos peligrosos no son concedidos en la instalación. La aplicación consultará al usuario si quiere conceder un permiso peligroso en el momento de utilizarlo:

Además se recomienda que la aplicación indique para que lo necesita. De esta forma el usuario tendrá más elementos de juicio para decidir si da o no el permiso. Si el usuario no concede el permiso la aplicación ha de tratar de continuar el proceso sin este permiso. Otro aspecto interesante es que el usuario podrá configurar en cualquier momento que permisos concede y cuáles no. Por ejemplo, podemos ir al administrador de aplicaciones y seleccionar la aplicación Navegador. En el apartado permisos se nos mostrará los grupos de permisos que podemos conceder:



Observa como de los grupos de permisos solicitados, en este momento solo concedemos el permiso de Ubicación.

El usuario concede o rechaza los permisos por grupos. Si en el manifiesto se ha pedido leer y escribir en la SD, concedemos los dos permisos o ninguno. Es decir, no podemos conceder permiso de lectura, pero denegar el de escritura.

A partir de Android Marshmallow (Android 6) trabajar con acciones que necesiten de un permiso va a suponer un esfuerzo extra para el programador. Antes de realizar la acción tendremos que verificar si tenemos el permiso. En caso negativo hay que exponer al usuario para qué lo queremos y pedírselo. Si el usuario no nos diera el permiso, tendremos qué decidir qué hacer. ¿Podemos realizar la acción solicitada aunque no dispongamos de cierta información? ¿Dejamos de hacer la acción solicitada? ¿O salimos de la aplicación? A continuación veremos cómo realizar esta tarea.

Ejemplo solicitud de permisos en Android Marshmallow

1. El primer paso va a ser verificar que tenemos el permiso adecuado antes de realizar una acción que lo requiera. Resulta sencillo, simplemente has de añadir el **`if`** como se muestra a continuación.

```
void borrarLlamada() {

    if (ContextCompat.checkSelfPermission(this,
            Manifest.permission.WRITE_CALL_LOG)
            == PackageManager.PERMISSION_GRANTED) {

        //Acciones a realizar cuando se dispone de permisos
        getContentResolver().delete(CallLog.Calls.CONTENT_URI,
                "number ='555555555'", null);

        Snackbar.make(view, "LLamadas borradasd del registro.",
                Snackbar.LENGTH_SHORT).show();
    }
}
```

   Si ejecutamos la aplicación ahora en un dispositivo con android versión 6 o superior, ya no se producirá el error, pero esto no soluciona el problema.

2. Nuestra aplicación no puede limitarse a no realizar la acción cuando no disponga del permiso. Ha de avisar al usuario y solicitar el permiso. Para ello añadimos una sección **`else`** al **`if`** anterior.

```
void borrarLlamada() {

    if (ContextCompat.checkSelfPermission(this,
            Manifest.permission.WRITE_CALL_LOG)
            == PackageManager.PERMISSION_GRANTED) {

        //Acciones a realizar cuando se dispone de permisos
        getContentResolver().delete(CallLog.Calls.CONTENT_URI,
                "number ='555555555'", null);
```

```
            Snackbar.make(view, "LLamadas borradasd del registro.",
                    Snackbar.LENGTH_SHORT).show();
        }
        else {

            //Solicitamos permisos
            solicitarPermiso(Manifest.permission.WRITE_CALL_LOG, "Sin el" +
                            " permiso ADMINISTRAR LLAMADAS no puedo borrar" +
                            " llamadas del registro.",
                            SOLICITUD_PERMISO_WRITE_CALL_LOG, this);
        }
    }
```

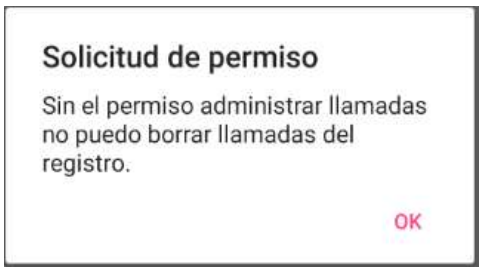
3. Añade el siguiente método, para solicitar permisos:

```
    public static void solicitarPermiso(final String permiso,
                                        String justificacion,
                                        final int requestCode,
                                        final Activity actividad) {

        if (ActivityCompat.shouldShowRequestPermissionRationale(actividad,
                                                                permiso)) {

          //Informamos al usuario para qué y por qué se necesitan los permisos
            new AlertDialog.Builder(actividad)
                    .setTitle("Solicitud de permiso")
                    .setMessage(justificacion)
                    .setPositiveButton("OK",
                                       new DialogInterface.OnClickListener() {

                        public void onClick(DialogInterface dialog,
                                            int whichButton) {

                            ActivityCompat.requestPermissions(actividad,
                                    new String[]{permiso}, requestCode);
                        }
                    })
                    .show();
        } else {
           //Muestra el cuadro de dialogo para la solicitud de los permisos
           // y registra el permiso según respuesta del usuario
            ActivityCompat.requestPermissions(actividad,
                    new String[]{permiso}, requestCode);
        }
    }
```

Es posible que haya que solicitar permisos desde diferentes puntos de la aplicación. Por esta razón declaramos este método público y estático. Además, se ha pasado como parámetro toda la información que necesita: el permiso a solicitar, la justificación de porque lo necesitamos, un código de solicitud y la actividad que recogerá la respuesta. Una vez el usuario decida si da el permiso, se llamará al método `onRequestPermissionsResult()`, que tendrás que declarar en la actividad que se pasa en el cuarto parámetro. El código es un valor numérico que permitirá identificar diferentes solicitudes.

Android recomienda que se indique al usuario para qué se le está solicitando el permiso. Si consideras que no es necesario, se puede eliminar la primera parte del método y dejar solo el código que aparece dentro del **`else`**. Antes de mostrar la explicación usando un `AlertDialog`, se verifica en el **`if`** si interesa mostrar esta información. Si el usuario ha indicado que no nos da el permiso y

además ha marcado la casilla de que no quiere que volvamos a preguntar, no es conveniente insistir. El sistema se encarga de recordar esta información, nosotros simplemente tenemos que usar el método `shouldShowRequestPermissionRationale()`. Este método, es de gran utilidad, muestra `true` si el usuario ha rechazado la solicitud anteriormente y muestra `false` si el usuario rechazó un permiso y selecciono la opción de **No volver a preguntar** en el dialogo de solicitud de permiso, o si una política de dispositivo lo prohíbe.



***NOTA**: Este código se ejecuta en el hilo principal, por lo tanto nunca utilices un método para preguntar al usuario ya que puede bloquear el hilo. Observa como en el ejemplo se utilizan llamadas asíncronas.*

El trabajo más importante lo hace el método `requestPermissions()` que muestra un cuadro de diálogo como el siguiente y registra el permiso según la respuesta del usuario:

4. Una vez que el usuario escoja, se realizará una llamada a `onRequestPermissionsResult()`. Con el siguiente código, se procesa la respuesta.

```
@Override
public void onRequestPermissionsResult(int requestCode,
                                       String[] permissions,
                                       int[] grantResults) {

    if (requestCode == SOLICITUD_PERMISO_WRITE_CALL_LOG){
        if (grantResults.length >= 1 &&
                grantResults[0] == PackageManager.PERMISSION_GRANTED) {
            borrarLlamada();
        }
        else {
            Toast.makeText(this, "Sin el permiso, no puedo realizar" +
                    " la acción",Toast.LENGTH_SHORT).show();
        }
    }
}
```

Este método ha de estar declarado en una actividad. En caso de que el usuario nos conceda el permiso, tenemos que volver a realizar la acción que no pudo realizarse. En caso de que hayas solicitado el permiso desde diferentes acciones o que hayas solicitado diferentes permisos, el valor de `requestCode` permitirá diferenciar cada caso.

5. Declara la siguiente variable al principio de la clase:

```
private static final int SOLICITUD_PERMISO_WRITE_CALL_LOG = 0;
```

Los tres pilares de la seguridad en Android

La seguridad es un aspecto clave de todo sistema. Si nos descargáramos una aplicación maliciosa de Internet o de Google Play Store, esta podría leer nuestra lista de contactos, averiguar nuestra posición GPS, mandar toda esta información por Internet y terminar enviando 50 mensajes SMS.

En algunas plataformas antiguas, como Windows Mobile, estábamos prácticamente desprotegidos ante aplicaciones maliciosas. Por lo tanto, los usuarios tenían que ser muy cautos antes de instalar una aplicación.

En otras plataformas, como en iOS, toda aplicación ha de ser validada por Apple antes de poder ser instala en un terminal. Además, solo está permitido instalar aplicaciones de la tienda oficial de Apple. Esto limita a los pequeños programadores y da un poder excesivo a Apple. Se trata de un planteamiento totalmente contrario al *software* libre.

Android propone un esquema de seguridad que protege a los usuarios, sin la necesidad de imponer un sistema centralizado y controlado por una única empresa. La seguridad en Android se fundamenta en los tres pilares siguientes:

- Como ya sabemos Android está basado en Linux, por lo tanto, vamos a poder aprovechar la seguridad que incorpora este sistema operativo. De esta forma Android puede impedir que las aplicaciones tengan acceso directo al *hardware* o interfieran con recursos de otras aplicaciones.
- Toda aplicación ha de ser firmada con un certificado digital que identifique a su autor. La firma digital también nos garantiza que el fichero de la aplicación no ha sido modificado. Si se desea modificar la aplicación está tendrá que ser firmada de nuevo, y esto solo podrá hacerlo el propietario de la clave privada. Es habitual que un certificado digital sea firmado a su vez por una autoridad de certificación, sin embargo en Android esto no es necesario
- Si queremos que una aplicación tenga acceso a partes del sistema que pueden comprometer la seguridad del sistema hemos de utilizar un modelo de permisos, de forma que el usuario conozca los riesgos antes de instalar la aplicación.

Usuario Linux y acceso a ficheros

Para proteger el acceso a recursos utilizados por otras aplicaciones, Android crea una cuenta de usuario Linux (user ID) nueva por cada paquete (.apk) instalado en el sistema. Este usuario es creado cuando se instala la aplicación y permanece constante durante toda su vida en el dispositivo.

Cualquier dato almacenado por la aplicación será asignado a su usuario Linux, por lo que normalmente no tendrán acceso otras aplicaciones. No obstante, cuando crees un fichero puedes usar los modos `MODE_WORLD_READABLE` y/o `MODE_WORLD_WRITEABLE` para permitir que otras aplicaciones puedan leer o escribir en el fichero. Aunque otras aplicaciones puedan escribir el fichero, el propietario siempre será el usuario asignado a la aplicación que lo creó.

Dado que las restricciones de seguridad se garantizan a nivel de proceso, el código de dos paquetes no puede, normalmente, ejecuten en el mismo proceso. Para que dos aplicaciones se ejecutaran en un mismo proceso, sería necesario usar el mismo usuario. Puedes utilizar el atributo `sharedUserId` en `AndroidManifest.xml` para asignar un mismo usuario Linux a dos aplicaciones. Con esto conseguimos que a efectos de seguridad ambas aplicaciones sean tratadas como una sola. Por razones de seguridad, ambas aplicaciones han de estar firmadas con el mismo certificado digital.

El esquema de permisos en Android

Para proteger ciertos recursos y características especiales, Android define un esquema de permisos. Toda aplicación que acceda a estos recursos está obligada a declarar su intención de usarlos. En caso de que una aplicación intente acceder a un recurso del que no ha solicitado permiso, se generará una excepción de permiso y la aplicación será interrumpida inmediatamente.

Cuando el usuario instala una aplicación este podrá examinar la lista de permisos que solicita la aplicación y decidir si considera oportuno instalar dicha aplicación. A continuación se muestra una lista con algunos de los permisos más utilizados en Android:

PERMISOS PELIGROSOS:

Almacenamiento Externo :

- **`WRITE_EXTERNAL_STORAGE`** – **Modificar/eliminar almacenamiento USB (API 4).** Permite el borrado y la modificación de archivos en la memoria externa. Lo ha de solicitar toda aplicación que necesite escribir un fichero en la memoria externa; por ejemplo, exportar datos en XML. Pero al permitirlo también podrán modificar/eliminar ficheros externos creados por otras aplicaciones.
- **`READ_EXTERNAL_STORAGE`** **– Leer almacenamiento USB (API 16).** Permite leer archivos en la memoria externa. Este permiso se ha introducido en la versión 4.1. En versiones anteriores todas las aplicaciones pueden leer en la memoria externa. Por lo tanto, has de tener cuidado con la información que dejas en ella.

Ubicación :

- **`ACCESS_COARSE_LOCATION`** **– Localización no detallada (basada en red).** Localización basada en telefonía móvil (*Cell-ID*) y Wi-Fi. Aunque en la actualidad esta tecnología suele ofrecernos menos precisión que el GPS, no siempre es así. Por ejemplo, se está aplicando en el interior de aeropuertos y museos con precisiones similares.

- **ACCESS_FINE_LOCATION – Localización GPS detallada.** Localización basada en satélites GPS. Al dar este permiso también estamos permitiendo la localización basada en telefonía móvil y Wi-Fi (ACCESS_COARSE_LOCATION).

Telefono  :

- **CALL_PHONE – Llamar a números de teléfono directamente. *Servicios por los que tienes que pagar.*** Permite realizar llamadas sin la intervención del usuario. Nunca solicites este permiso en tus aplicaciones, muchos usuarios no instalarán tu aplicación. Si has de realizar una llamada, es mejor realizarla por medio de una intención. A diferencia de la llamada directa, no necesitas ningún permiso, dado que el usuario ha de pulsar el botón de llamada para que comience.
- **READ_PHONE_STATE – Consultar identidad y estado del teléfono.** Muchas aplicaciones, como los juegos, piden este permiso para ponerse en pausa cuando recibes una llamada. Sin embargo, también permite el acceso al número de teléfono, IMEI (identificador de teléfono GSM), IMSI (identificador de tarjeta SIM) y al identificador único de 64 bits que Google asigna a cada terminal. Incluso si hay una llamada activa, podemos conocer el número al que se conecta la llamada.
- **READ_CALL_LOG y WRITE_CALL_LOG – Leer y modificar el registro de llamadas telefónicas.**
- **ADD_VOICEMAIL – Añadir mensajes de voz.** Permite crear nuevos mensajes de voz en el sistema.
- **USE_SIP – Usar Session Initial Protocol.** (API 9). Permite a tu aplicación usar el protocolo SIP.
- **PROCESS_OUTGOING_CALLS – Procesar llamadas salientes.** Permite a la aplicación controlar, modificar o abortar las llamadas salientes.

Mensajes de texto (SMS)  :

- **SEND_SMS – Enviar mensaje SMS. *Servicios por los que tienes que pagar*.** Permite la aplicación mandar de texto SMS sin la validación del usuario. Al igual que **CALL_PHONE**, a no ser que tu aplicación tenga que mandar SMS sin la intervención del usuario, resulta más conveniente enviarlos por medio de una intención.
- **RECEIVE_SMS – Recibir mensajes de texto.** Permite a la aplicación recibir y procesar SMS. Una aplicación puede modificar o borrar los mensajes recibidos.
- **READ_SMS – Leer mensajes de texto.** Permite a la aplicación leer los mensajes SMS entrantes.
- **RECEIVE_MMS – Recibir mensajes MMS.** Permite monitorizar los mensajes multimedia entrantes, pudiendo acceder a su contenido.
- **RECEIVE_WAP_PUSCH – Recibir mensajes WAP Push.** Permite monitorizar los mensajes WAP Push entrantes. Un mensaje WAP PUSH es un tipo de SMS que se usa para acceder de manera sencilla a una página WAP en lugar de teclear su direccion URL en el navegador.

Contactos :

- `READ_CONTACTS` **– Leer datos de contactos.** Permite leer información sobre los contactos almacenados (nombres, correos electrónicos, números de teléfono). Algunas aplicaciones podrían utilizar esta información de forma no lícita
- `WRITE_CONTACTS` **– Escribir datos de contactos.** Permite modificar los contactos.
- `GET_ACCOUNTS` **– Obtener Cuentas.** Permiten acceder a la lista de cuentas en el Servicio de Cuentas.

Calendario :

- `READ_CALENDAR` **– Leer datos de contactos.** Permite leer información del calendario del usuario.
- `WRITE_CONTACTS` **– Escribir datos de contactos.** Permite escribir en el calendario, pero no leerlo.

Camara :

- `CAMARA` **– Hacer fotos / grabar vídeos.** Permite el acceso al control de la cámara y a la toma de imágenes y vídeos. El usuario puede no ser consciente.

Microfono :

- `RECORD_AUDIO` **– Grabar audio.** Permite acceso grabar sonido desde el micrófono del teléfono.

Sensores corporales :

- `BODY SENSORS` **– Leer sensores corporales.** Da acceso a los datos de los sensores que están monitorizando el cuerpo del usuario. Por ejemplo, el lector de ritmo cardiaco.

PERMISOS NORMALES:

Comunicaciones :

- `INTERNET` **– Acceso a Internet sin límites.** Permite establecer conexiones a través de Internet. Este es un permiso muy importante, en el que hay que fijarse a quién se otorga. La mayoría de las aplicaciones lo piden, pero no todas lo necesitan. Cualquier *malware* necesita una conexión para poder enviar datos de nuestro dispositivo.
- `ACCESS_NETWORK_STATE` **– Ver estado de red.** Información sobre todas las redes. Por ejemplo para saber si tenemos conexión a internet.
- `CHANGE_NETWORK_STATE` **– Cambiar estado de red.** Permite cambiar el estado de conectividad de redes.
- `NFC` **– Near field communication.** (API 19) Algunos dispositivos disponen de un trasmisor infrarrojo para el control remoto de electrodomésticos.

- **TRASMIT_R – Trasmitir por infrarrojos.** (API 19) Algunos dispositivos disponen de un trasmisor infrarrojo para el control remoto de electrodomésticos.

Conexión WIFI :

- **ACCESS_WIFI_STATE – Ver estado de Wi-Fi.** Permite conocer las redes Wi-Fi disponibles.
- **CHANGE_WIFI_STATE – Cambiar estado de Wi-Fi.** Permite cambiar el estado de conectividad Wi-Fi.
- **CHANGE_WIFI_MULTICAST_STATE – Cambiar estado multicast Wi-Fi.**(API 4). Permite pasar al modo Wi-Fi Multicast.

Bluetooth :

- **BLUETOOTH – Crear conexión Bluetooth.** Permite a una aplicación conectarse con otro dispositivo Bluetooth. Antes ambos dispositivos han de emparejarse.
- **BLUETOOTH_ADMIN – Emparejar Bluetooth.** Permite descubrir y emparejarse con otros dispositivos Bluetooth.

Consumo de batería :

- **WAKE_LOCK – Impedir que el teléfono entre en modo de suspensión.** Para algunas aplicaciones, como un navegador GPS, puede ser importante que no sean suspendidas nunca. Realmente, a lo único que puede afectar es a nuestra batería.
- **FLASHLIGHT – Linterna.** Permite encender el flash de la cámara.
- **VIBRATE – Control de la vibración.** Permite hacer vibrar al teléfono. Los juegos suelen utilizarlo.

Aplicaciones :

- **RECEIVE_BOOT_COMPLETED – Ejecución automática al encender el teléfono.** Permite a una aplicación recibir el anuncio broadcast `ACTION_BOOT_COMPLETED` enviado cuando el sistema finaliza un inicio. Gracias a esto la aplicación pondrá ponerse en ejecución al arrancar el teléfono.
- **BROADCAST_STICKY – Enviar anuncios broadcast permanentes.** Un broadcast permanente llegará a los receptores de anuncios que actualmente estén escuchando, pero también a los que se instancien en un futuro. Por ejemplo, el sistema emite el anuncio broadcast `ACTION_BATTERY_CHANGED` de forma permanente. De esta forma, cuando se llama a `registerReceiver()` se obtiene la intención de la última emisión de este anuncio. Por lo tanto, puede usarse para encontrar el estado de la batería sin necesidad de esperar a un futuro cambio en su estado. Se ha incluido este permiso dado que las aplicaciones mal intencionadas pueden ralentizar el dispositivo o volverlo inestable al demandar demasiada memoria.
- **KILL_BACKGROUND_PROCESSES – Matar procesos en Background (API 9)**. Permite llamar a `killBackgroundProcesses(String)`. Al hacer esta llamada el sistema mata de inmediato a todos los procesos de fondo asociados con el paquete indicado. Es el mismo método que usa el sistema cuando necesita memoria. Estos procesos serán reiniciados en el futuro, cuando sea necesario.

- **REORDER_TASKS – Reordenar tareas.** Permite a una aplicación cambiar el orden de la lista de tareas.
- **INSTALL_SHORTCUT** y **UNINSTALL_SHORTCUT – Instalar y desinstalar acceso directo (API 19).** Permite a una aplicación añadir o eliminar un acceso directo a nuestra aplicación en el escritorio.
- **GET_PACKAGE_SIZE – Obtener tamaño de un paquete.** Permite a una aplicación conocer el tamaño de cualquier paquete.
- **EXPAND_STATUS_BAR – Expandir barra de estado**. Permite a una aplicación expandir o contraer la barra de estado

Configuraciones del sistema :

- **CHANGE_CONFIGURATION – Modificar la configuración global del sistema**. Permite cambiar la configuración del sistema (como la configuración local).
- **SET_WALLPAPER – Poner fondo de pantalla.** Permite establecer fondo de pantalla en el escritorio.
- **SET_WALLPAPER_HITS – Sugerencias de fondo de pantalla.** Permite a las aplicaciones establecer sugerencias de fondo de pantalla.
- **SET_ALARM – Establecer Alarma.** Permite a la aplicación enviar una intención para poner una alarma o temporizador en la aplicación Reloj.
- **SET_TIME_ZONE – Cambiar zona horario.** Permite cambiar la zona horaria del sistema.
- **ACCESS_NOTIFICATION_POLICY – Acceso a política de notificaciones (API 23).** Permite conocer la política de notificaciones del sistema.

Audio :

- **MODY_AUDIO_SETTINGS – Cambiar ajustes de audio.** Permite cambiar ajustes globales de audio, como el volumen.

Sincronización :

- **READ_SYNC_SETTINGS – Leer ajustes de sincronización**. Permite saber si tienes sincronización en segundo plano con alguna aplicación (como con un cliente de Twitter o Gmail).
- **WRITE_SYNC_SETTINGS – Escribir ajustes de sincronización.** Permite registrar tu aplicación como adaptador de sincronización (`SyncAdapter`).
- **READ_SYNC_STATS – Leer estadísticas de sincronización.**

Ubicación :

- **ACCESS_LOCATION_EXTRA_COMMANDS – Mandar comandos extras de localización.** Permite a una aplicación acceder a comandos adicionales de los proveedores de localización. Por ejemplo, tras pedir este permiso podríamos enviar el siguiente comando al GPS, con el método: `sendExtraCommand("gps", "delete_aiding_data", null);`.

Seguridad :

- **USE_FINGERPRINT – Usar huella digital (API 23).** Permite usar el hardware de reconocimiento de huella digital.
- **DISABLE_KEYGUARD – Deshabilitar bloqueo de teclado**. Permite a las aplicaciones desactivar el bloqueo del teclado si no es seguro.

La lista completa de todos los permisos en Android con cierta información sobre ellos, se puede consultar en la página oficial de Android Manifest.permission

NOTA: *Los permisos peligrosos pertenecen a uno de los 9 grupos anteriores. Estos grupos son importantes dado que el usuario concede o deniega el permiso a un grupo entero. Por el contrario, a partir de la versión 6.0 los permisos normales ya no se clasifican en grupos.*

NOTA: *Existen otros permisos que no han sido incluidos en esta lista dado que no podemos solicitarlos en nuestras aplicaciones al estar reservados para aplicaciones del sistema.*

Para solicitar un determinado permiso en tu aplicación, no tienes más que incluir una etiqueta `<uses-permission>` en el fichero *AndroidManifest.xml* de tu aplicación. En el siguiente ejemplo se solicitan dos permisos:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.amaia.xxx">

. . .
    <uses-permission android:name="android.permission.RECEIVE_SMS"/>
    <uses-permission android:name="android.permission.SEND_SMS" />

. . .

</manifest>
```

Permisos definidos por el usuario en Android

Además de los permisos definidos por el sistema, los desarrolladores van a poder crear nuevos permisos para restringir el acceso a elementos de nuestro *software*.

Esto se trata de un aspecto avanzado y no es necesario en la mayoría de las aplicaciones.