DEPARTAMENTO DE INFORMÁTICA

INGENIERÍA TÉCNICA EN INFORMÁTICA DE GESTIÓN

PROYECTO FIN DE CARRERA

CONTROLES Y SEGURIDAD BAJO ENTORNO ANDROID

Autor: Christian Madero García.

Tutor: Miguel Ángel Ramos.

Leganés, Octubre de 2013

Título: CONTROLES Y SEGURIDAD BAJO ENTORNO ANDROID
Autor: Christian Madero García
Director: Miguel Ángel Ramos

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 24 de Octubre de 2013 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Agradecimientos

En primer lugar me gustaría agradecer a mis padres, Justo y Otilia, por darme la oportunidad de adquirir esta educación y estar ahí en los momentos difíciles que he pasado en la universidad, sin su apoyo y dedicación no habría sido posible llegar hasta el final de este viaje. También a mi hermana Miriam por estar junto a nosotros e incordiarme a lo largo de los años con el tema de la carrera. Gracias.

También me gustaría agradecer su apoyo a mis compañeros de universidad, con los que he pasado buenos, malos momentos y momentos inolvidables durante estos años de carrera; Gracias a Nando, Miguel, Marcos, Abel, Elisa, Edu, Alberto…

Como mención especial y por supuesto sin olvidarme de Maila, gracias a ella por estar a mi lado en cada paso antes y durante este proyecto, gracias a ella me ha sido posible confeccionar este proyecto final de carrera, por su apoyo incondicional y por alentarme pasito a pasito ha sido posible llegar hasta la meta. Gracias.

Para finalizar, me gustaría agradecer a mi tutor Miguel Ángel, por su dedicación, esfuerzo, paciencia y experiencia, por guiarme a través de esta aventura. Gracias.

Resumen

Los dispositivos móviles, smartphones y tablets, constituyen un avance importante en lo que al mundo de las comunicaciones y tratamiento de la información se refiere, el avance tecnológico en esto últimos años ha sido exponencial respecto a estas tecnologías, permitiendo a las empresas procesar y gestionar información sensible de manera más versátil y dinámica, de una forma que hace tan sólo unos años nos parecería impensable, alcanzando una gran cuota de mercado el sistema operativo móvil Android, motivo de estudio en el presente proyecto.

Con dicho proyecto se busca conocer y comprender las características, el funcionamiento, seguridad y controles de los que dispone este sistema operativo, descubriendo sus posibilidades y ventajas. Además, abarca el desarrollo completo de una aplicación de nombre *AgendaPersonal*, que nos ha permitido ahondar en lo que ha desarrollo, seguridad, controles y herramientas para esta plataforma se refiere y con la que se busca ilustrar de forma práctica la construcción y naturaleza de las aplicaciones Android, ya que a menudo, este tipo de dispositivos alberga información importante que debe ser protegida de agentes externos.

Para finalizar propondremos una serie de añadidos que a nuestro juicio mejorarían algunos puntos de la plataforma.

Abstract

Mobile devices, smartphones and tablets represent important progress in the world of communications and data processing. Over the last few years,technological advances have led to an exponential growth in new technology being implemented, which in turn has allowed companies to manage sensitive information in a more versatile and dynamic manner. This was not possible only a few years ago and has earned the Android operating system a high market share, which will be studied in this document.

With this thesis we want to study and understand the properties, controls, security and behaviour of the operating system, focusing on its odds and benefits. We also explain the development of an application called AgendaPersonal, which has allowed us to work directly with different operating system tools and controls. With AgendaPersonal we want to show how to create an Android application in a practical way, demostrating that this kind of devices has important information that must be protected against external agents.

Finally we propose some changes that would probably improve some features of the platform.

Índice general

Índice de figuras

Índice de tablas

Índice de códigos fuente

Capítulo 1

Introducción y objetivos

1 Introducción

Al echar la vista atrás podemos observar el desarrollo tecnológico que ha experimentado la humanidad desde mediados del siglo XX hasta hoy, más que un avance se ha producido una verdadera revolución. El descubrimiento de la informática, su aplicación paulatina en todo tipo de áreas de conocimiento y de producción, así como su introducción en las tareas cotidianas de la gente a través de todo tipo de dispositivos, han cambiado nuestra sociedad y nuestra economía más rápido que cualquier otro invento o descubrimiento anterior.

La computadora personal u ordenador es uno de los inventos que mejor resume la nueva situación tecnológica. Aparecieron primero las computadoras de primera generación como enormes y costosas máquinas que solamente estaban disponibles en importantes universidades o centros de investigación por la década de 1950.

Con la aparición de nuevas técnicas de fabricación, como los circuitos integrados, su tamaño, sus capacidades, y sobre todo precio, variaron de tal forma que se convirtieron en un producto de masas, como lo podían ser la televisión o la radio. La aparición de Internet, y sobre todo su apertura al público general, determinaron de forma inequívoca la importancia de los ordenadores en la vida social, laboral o académica de cualquier persona hasta el día de hoy.

Simultáneamente a la aparición de Internet como servicio abierto, a principios de a década de 1990, surgió otro medio de comunicación que, si bien era más antiguo, se reinventaba a sí mismo gracias a los cambios en su tradicional soporte: la telefonía móvil.

El *boom* en la implantación de Internet, junto al furor de la telefonía móvil, confirmaba que esta revolución tecnológica no sólo afectaba a la investigación o la actividad económica, sino que implicaba un fenómeno sociológico donde la comunicación y el acceso a la información en cualquier lugar y momento eran sus pilares básicos.

Como no podía ser de otra manera, la reducción del tamaño de los componentes y el aumento de sus prestaciones permitió acercar cada vez más ambos mundos, de forma que a través de un teléfono móvil no sólo se podían hacer llamadas o enviar SMS, sino que además se podía tener un acceso a Internet, o incluso funciones añadidas como realizar fotografías o vídeos. Otros aparatos de similar tamaño, no directamente relacionados con la telefonía, surgieron y se hicieron tan populares como los primeros. Desde ese momento puede empezar a usarse el término genérico dispositivo móvil.

Así pues, un dispositivo móvil es un término general que describe una amplísima familia de aparatos electrónicos surgidos en los últimos años, de reducido tamaño, que ofrecen alguna capacidad de procesamiento y almacenamiento de datos y que están orientados a una función concreta o varias de ellas: desde los teléfonos móviles más revolucionarios y con mayor número de utilidades (los llamados *smartphones*), a ordenadores portátiles, tablets pc, cámaras digitales, reproductores de música o consolas de videojuegos.

La mayoría de estos aparatos cuentan con un sistema operativo de mayor o menor complejidad que permite realizar las tareas de gestión de memoria y control de hardware que precisan. En el caso de los ordenadores portátiles, con tanta o incluso mayor capacidad que los de sobremesa, los sistemas operativos habituales son perfectamente compatibles y funcionan sin diferencias. Sin embargo, en los dispositivos móviles es preciso diseñar nuevos sistemas operativos adaptados específicamente a sus características, dadas sus restricciones de memoria y procesamiento, un consumo mínimo de energía o gran estabilidad en su funcionamiento, entre otros.

Google es una joven compañía surgida a finales de los 90 que pronto se hizo muy popular gracias al potente buscador del mismo nombre. Durante años, Google no ha dejado de crecer y de ofrecer toda clase de servicios basados siempre en Internet y en la combinación de la última tecnología disponible con la máxima experiencia del usuario.

Android, más que un sistema operativo, representa toda una pila de software para dispositivos móviles que incluye gran cantidad de drivers, gestor de bases de datos, un completo *framework* de aplicaciones, y numerosas aplicaciones de usuario. Android está basado en el núcleo de Linux y todas sus aplicaciones se escriben en lenguaje Java, disponiendo además de una máquina virtual específica llamada Dalvik.

Con la aparición de este sistema, Google pretende aprovechar al máximo la cada vez mayor capacidad de los dispositivos móviles, que llegan a incluir componentes como GPS, pantallas táctiles, conexiones rápidas a Internet, y por supuesto, todos los servicios

asociados hasta ahora a los teléfonos móviles, además de aplicaciones de usuario hasta ahora limitadas a los ordenadores, como clientes de correo, aplicaciones ofimáticas o videojuegos. En Android, cada aplicación corre en su propio proceso, donde el sistema decide en cada momento qué aplicación debe ser eliminada para liberar recursos en caso de carencia, y responsabilizándose igualmente de restaurarla de forma totalmente transparente al usuario. Navegar entre varias aplicaciones abiertas deja de ser una característica propia de ordenadores.

Android se lanza bajo la licencia Apache, lo que implica que, como software libre, cualquier desarrollador tiene acceso completo al SDK del sistema, incluidas todas sus API, documentación y emulador para pruebas, pudiendo distribuirlo y modificarlo.

Además, esta licencia permite a los desarrolladores tanto publicar a su vez sus creaciones, como distribuirlas únicamente bajo pago ocultando el código fuente. Este nuevo sistema introduce también interesante conceptos, como es la composición de sus aplicaciones a través de combinación de módulos o bloques básicos, según la naturaleza de la aplicación, o la delegación en el sistema de determinadas acciones para que sean otras aplicaciones instaladas las que se hagan cargo de ellas.

Numerosos fabricantes, distribuidores y operadores se han unido a la plataforma de patrocinadores de Android, y fabricantes de gran renombre han anunciado la producción inminente de nuevos modelos con Android como sistema nativo [1]. Dada la creciente importancia de estos dispositivos y su más que probable implantación masiva, cada vez con mayores prestaciones y capacidades, unida al éxito que suele acompañar a Google en sus proyectos, todo parece indicar que Android podría posicionarse en un futuro más o menos cercano como uno de los sistemas operativos más utilizados en el mundo.

1.1 Objetivos

El presente proyecto fin de carrera se centra en el estudio de la plataforma Android para dispositivos móviles y el desarrollo de un prototipo inicial de aplicación que ofrezca la posibilidad de comprobar los controles de seguridad implementados para el desarrollo de aplicaciones en dispositivos Android. Para la consecución satisfactoria de dicha tarea se han impuesto algunos objetivos que necesariamente deben cumplirse.

Todo esto implica conocer las principales características de dicho sistema operativo como puede ser su arquitectura, componentes, y metodología que utiliza. Al ser la plataforma de código libre el SDK está disponible al público, este kit de desarrollo incluye desde una extensa documentación, hasta un complejo emulador de Android. Estos elementos serán estudiados y explicados para una mejor comprensión del lector.

El nuevo sistema operativo para dispositivos móviles de Google, Android, centra el desarrollo de este proyecto fin de carrera. Para poder dirigir con mayor éxito los esfuerzos por conocer y comprender las características de este nuevo sistema, es necesario fijar unos objetivos que abarquen las actividades que se pretenden realizar y, además, permitan al final de las mismas conocer el grado de desarrollo y cumplimiento alcanzado.

Por ello, los objetivos perseguidos en el desarrollo de este proyecto fin de carrera son los enumerados a continuación:

Conocer las principales características de Android. El primer paso para conocer este nuevo sistema debe consistir en indagar toda la información posible sobre él, a fin de conocer cuál es su arquitectura, sus componentes básicos, y cuál es su comportamiento al ejecutar las aplicaciones, documentando todos estos aspectos. Además, ha de averiguarse cuáles son las ventajas y las posibilidades reales que Android ofrece frente a otros sistemas competidores de similar naturaleza.

Estudiar y analizar las principales características de seguridad de Android. En este caso para conocer y analizar las principales características de seguridad de este nuevo sistema, se debe conseguir encontrar y analizar toda la información posible sobre la arquitectura y las posibilidades que esta nos ofrece, tales como: el núcleo del sistema (Kernel Linux), aplicaciones base, marco de trabajo de aplicaciones, bibliotecas y “runtime” de Android, para así poder recopilar las ventajas que nos ofrece la plataforma Android, dichas ventajas o características serán profundizadas en capítulos posteriores.

Estudiar el entorno de desarrollo de Android y su seguridad a la hora de desarrollar aplicaciones. Al lanzarse bajo una licencia de software libre, el SDK completo está disponible para cualquier desarrollador que desee descargarlo. Este incluye numerosas ayudas para comenzar a crear aplicaciones en Android, desde las API completas con todas las clases y paquetes, hasta herramientas de programación y un completo emulador para poder realizar pruebas. Todos estos elementos han de ser estudiados y explicados.

Desarrollar una aplicación completa para Android. Una vez conocidas las características de este sistema, así como el entorno de desarrollo que ofrece y sus posibilidades, debe crearse una aplicación que aproveche algunas de sus características más fundamentales. A través de este desarrollo y una detallada documentación del mismo, el lector debe poder comprender mejor el funcionamiento de aplicaciones para Android, así como conocer los pasos para crear sus propias aplicaciones.

Proponer mejoras en la seguridad del sistema operativo Android. Para finalizar la descripción de los objetivos principales de este proyecto fin de carrera, debemos comentar la proposición de mejoras en la seguridad de la plataforma Android a modo de reforzar la misma, dichas propuestas las describiremos con más detalle en el capítulo de conclusiones de este mismo proyecto final de carrera.

1.2 Fases del desarrollo

A continuación vamos a enumerar y describir al lector las fases del desarrollo para este proyecto final de carrera:

Investigación. El primer paso para la realización de este proyecto es conocer este nuevo sistema operativo para dispositivos móviles, indagar y recopilar toda la información posible sobre el mismo, a fin de conocer su arquitectura, sus componentes básicos, y cuál es su comportamiento al ejecutar y desarrollar aplicaciones para el mismo.

Instalación del entorno de desarrollo. Al lanzarse bajo una licencia de software libre, el SDK completo está disponible para cualquier desarrollador que desee descargarlo e instalarlo. Este último incluye numerosas ayudas para comenzar a desarrollar aplicaciones para la plataforma, desde las API completas con todas las clases y paquetes, hasta herramientas de programación y un completo emulador para poder realizar las pruebas pertinentes. Todos estos elementos han de ser estudiados y explicados.

Desarrollo de la aplicación de prueba. Una vez conocidas las características de este sistema, así como el entorno de desarrollo que ofrece y sus posibilidades y/o ventajas, debe crearse una aplicación que aproveche algunas de sus características más fundamentales como pueden ser el sistema de controles y permisos. A través de este desarrollo y una detallada documentación del mismo, podremos comprender mejor el funcionamiento de las aplicaciones para la plataforma, así como conocer los controles proporcionados para nuestras propias aplicaciones por el sistema Android.

Documentación. Como hemos comentado en los puntos anteriores todo lo investigado sobre la plataforma Android, así como la información sobre el entorno de desarrollo y herramientas instaladas, y por último las pruebas e investigación realizada sobre nuestra aplicación de prueba, todo ello debemos de documentarlo explícitamente para la mejor comprensión por parte del lector.

1.3 Medios empleados

Para la realización de este proyecto fin de carrera los medios empleados han sido los siguientes:

- Equipo portátil:

  Procesador: Intel Core 2 Duo CPU T5470 1.60GHz.

  Tarjeta gráfica: Mobile Intel 965.

  Memoria RAM: 2 GB.

  Disco duro: 160 GB.

  Sistema operativo: Windows 7 Ultimate 32 bits Service Pack 1.

- Equipo sobremesa:

  Procesador: Intel Core 2 Quad CPU Q9650 3.00GHz.

  Tarjeta gráfica: Nvidia 9500 GT 1GB.

  Memoria RAM: 4 GB.

  Disco duro: 500 GB.

  Sistema operativo: Windows 7 Professional 32 bits Service Pack 1.

- Máquina virtual:

  Versión: VMware Player 3.1.4 build - 385536.

  Memoria RAM: 1 GB.

  Disco duro: 40 GB.

  Sistema operativo: Windows XP Professional 32 bits Service Pack 3.

- Dispositivo de almacenamiento externo: ScanDisk 16 GB.

1.4 Estructura de la memoria

A continuación se explica brevemente el contenido de cada capítulo y anexos incluidos en esta memoria.

En el capítulo 1, *Introducción*, se expone la motivación del presente proyecto, los objetivos que se persiguen, así como una visión general de los contenidos de la memoria.

En el capítulo 2, *La Seguridad Informática*, realizamos una introducción al concepto de la misma, así como una visión general de lo que conlleva dentro de un sistema operativo y la clasificación de las distintas amenazas y/o ataques para el mismo.

En el capítulo 3, *Auditoría Informática*, realizamos una introducción al concepto de la misma, así como una visión general de los distintos tipos de auditorías, tanto internas como externas, haciendo hincapié en los distintos tipos de controles existentes para la realización de las mismas.

En el capítulo 4, *La Plataforma Android*, se explican al lector las características básicas del sistema operativo Android, su diseño, arquitectura y funcionamiento, explicando al lector los puntos fuertes en la seguridad de la plataforma y de las herramientas de las que está provista, ahondando en este último tema mencionado.

En el capítulo 5, *La Firma de Aplicaciones Android*, se explica al lector como se lleva a cabo el firmado de aplicaciones por parte del desarrollador para su posterior publicación en el mercado de aplicaciones de Google, Google Play.

En el capítulo 6, *Versionar aplicaciones Android*, se explica detalladamente al lector el proceso de versionado de las aplicaciones Android por parte del desarrollador de las mismas, para que el usuario final siempre tenga a su disposición la última versión de sus aplicaciones.

En el capítulo 7, *Publicar aplicaciones Android*, en este capítulo se explica detalladamente al lector como los desarrolladores de aplicaciones para la plataforma, ponen a disposición del usuario final las aplicaciones anteriormente mencionadas desarrolladas por los mismos mediante el servicio de Google, Google Play.

En el capítulo 8, *Ataques a la plataforma Android*, se explica al lector la historia en el tiempo de los diversos ataques que ha sufrido la plataforma hasta el momento por parte de aplicaciones malintencionadas.

En el capítulo 9, *La aplicación Agenda Personal*, en este capítulo se explica al lector detalladamente la aplicación que hemos desarrollado a modo de investigación, para el buen entendimiento y comprensión de las herramientas y seguridad, que ofrece la plataforma a los desarrolladores de la misma en lo que a seguridad y controles del sistema implica.

En el capítulo 10, *Conclusiones*, este capítulo es el resultado de la investigación tanto documental como del desarrollo de nuestra aplicación en este proyecto fin de

carrera, en él se detallan al lector las conclusiones a las que se han llegado en lo que al tema de la seguridad y los controles de este sistema operativo ofrecen.

En el capítulo 11, *Presupuesto*, detallamos al lector el presupuesto necesario para la realización de este proyecto fin de carrera.

En el anexo A, *Normalización y estándares*, se detalla al lector la normalización y estandarización de la ISO 27001, la ISO 27002 y el Reglamento de desarrollo de la Ley Orgánica 15/1999, aplicando todas ellas a la plataforma Android.

En el anexo B, *Instalación entorno de desarrollo*, se detalla al lector la instalación del entorno de trabajo para la realización del desarrollo de nuestra aplicación *Agenda Personal*.

Capítulo 2

La Seguridad Informática

2 Seguridad Informática

Para la realización de este capítulo de nuestro proyecto fin de carrera, hemos recopilado y estudiado información de las siguientes referencias [1], [2], [3] y [4].

La seguridad informática es el área de la informática que se enfoca en la protección de la infraestructura computacional y todo lo relacionado con esta (incluyendo la información contenida). Para ello existen una serie de estándares, protocolos, métodos, reglas, herramientas y leyes concebidas para minimizar los posibles riesgos relacionados con la infraestructura o la información. La seguridad informática comprende software, bases de datos, metadatos, archivos y todo lo que la organización valore (activo) y signifique un riesgo si la información llega a manos de otras personas. Este tipo de información se conoce como información privilegiada o confidencial.

La seguridad informática abarca los conceptos de seguridad física y lógica. La seguridad física se refiere a la protección mediante hardware y soportes de datos, así como la seguridad de los edificios e instalaciones en los que se encuentran. También se deben contemplar situaciones de incendios, inundaciones, sabotajes, robos, catástrofes naturales, etc.

Por otra parte, la seguridad lógica se refiere a la seguridad en el uso de software, la protección de los datos, procesos y programas, así como la de acceso ordenado y autorizado de los usuarios a la información.

La seguridad de los sistemas también implica que se debe tener cuidado de que no existan copias piratas, o bien que, al estar conectados en red con otros equipos, no exista la posibilidad de transmisión de virus a los mismos.

Para que un sistema informático, ya sea un sistema operativo, una aplicación, o servicios, sea seguro en la medida de lo posible ha de cumplir las siguientes características:

Confidencialidad. Requiere que la información sea accesible únicamente por las entidades autorizadas.

Integridad. Requiere que la información sólo pueda ser modificada por las entidades autorizadas a tal efecto. La modificación no solo incluye el cambio de los mensajes transmitidos sino también incluye escritura, borrado, creación, y actualización.

No repudio. Ofrece protección a un usuario frente a otro usuario que niegue posteriormente que se realizó cierta comunicación. El no repudio de origen protege al receptor de que el emisor niegue haber enviado el mensaje mientras que el no repudio de recepción protege al emisor de que el receptor niegue haber recibido el mensaje.

Disponibilidad. Requiere que los recursos del sistema informático estén disponibles a las entidades autorizadas cuando lo necesiten.

2.1 Objetivos de la Seguridad Informática

La seguridad informática está concebida para proteger los activos informáticos, entre los que se encuentran:

- **La información contenida:**

Es uno de los elementos más importantes dentro de una organización. La seguridad informática debe ser administrada según los criterios establecidos por los administradores y supervisores, evitando que usuarios externos y no autorizados puedan acceder a ella sin autorización. De lo contrario la organización corre el riesgo de que la información sea utilizada maliciosamente para obtener ventajas de ella o que sea manipulada, ocasionando lecturas erradas o incompletas de la misma. Otra función de la seguridad informática en esta área es la de asegurar el acceso a la información en el momento oportuno, incluyendo respaldos de la misma en caso de que esta sufra daños o pérdida, producto de accidentes, atentados o desastres.

- **Los usuarios:**

Son las personas que utilizan la estructura tecnológica y que gestionan la información. La seguridad informática establece normas que minimicen los riesgos a la información o infraestructura informática. Estas normas pueden incluir horarios,

restricciones de acceso a ciertos lugares, autorizaciones, perfiles y roles de usuario, planes de emergencia, protocolos y todo lo necesario que permita un buen nivel de seguridad informática. Además los usuarios son los principales contribuyentes al uso de programas realizados por programadores.

- **La infraestructura computacional:**

La infraestructura computacional es una parte fundamental para el almacenamiento y gestión de la información, así como para el buen funcionamiento de la organización. La funcionalidad de la seguridad informática en esta área es velar que los equipos funcionen correctamente y prever en caso de fallos, planes de robos, incendios, boicot, desastres naturales, fallos en el suministro eléctrico y cualquier otro factor que atente contra la infraestructura.

2.2 Las amenazas y sus tipos

Una vez que el desarrollo y el funcionamiento de un dispositivo de almacenamiento o transmisión de la información se consideran seguros, todavía se deben de tener en cuenta las circunstancias "no informáticas" que puedan afectar a la información, dichas circunstancias son a menudo imprevisibles o inevitables, de modo que la única protección posible es la redundancia (en el caso de la información) y la descentralización.

Estas circunstancias pueden ser causadas por:

- **Programas maliciosos:** Programas con el fin de perjudicar o a hacer un uso ilícito de los recursos del sistema. Se instala en el equipo abriendo una puerta a intrusos o bien modificando datos o información. Estos programas pueden ser del tipo virus informático, gusano informático, troyano, bomba lógica o programa espía (Spyware).

- **Un intruso:** Persona que consigue el acceso a la información o programas de los cuales no tiene el acceso permitido (ejemplos: cracker, defacer, script kiddie, Script boy, viruxer, etc.).

- **El usuario:** Son las personas que utilizan la estructura tecnológica y que gestionan la información. A menudo son la causa del mayor problema ligado a la seguridad de un sistema informático por desconocimiento del mismo.

- **Un siniestro o catástrofe:** Una mala manipulación o una mala intención derivan en la pérdida del material, archivos o información (ejemplos: robo, incendio, inundación).

- **El personal interno de Sistemas:** Las pujas de poder que llevan a disociaciones entre los sectores organizativos y la propuesta de soluciones incompatibles para la seguridad informática.

Tipos de amenazas:

El hecho de conectar una red a un entorno externo a la organización da la posibilidad de que algún atacante pueda entrar en ella, y con ello, se puede ocasionar un robo de información o alterar el funcionamiento de la red. Por el contrario el hecho de que la red no sea conectada a un entorno externo no nos garantiza la seguridad de la misma. De acuerdo con el Computer Security Institute (CSI) de San Francisco aproximadamente entre el 60 y 80 por ciento de los incidentes de red son causados desde dentro de la misma.

Basándonos en esto podemos decir que existen 2 tipos de amenazas:

- **Amenazas internas:** Generalmente estas amenazas pueden ser más críticas que las externas por varias razones que expondremos a continuación:

  - Los usuarios tienen conocimiento de la red y saben cómo es su funcionamiento.

  - Tienen un nivel de acceso respecto a la red por necesidades de su trabajo.

  - Los IPS y Firewalls son mecanismos de seguridad no efectivos en amenazas internas.

  Esta situación se da gracias a las ineficiencias en los esquemas de seguridad con los que cuentan la mayoría de las compañías a nivel mundial, y porque no existe conocimiento relacionado con el planteamiento de un esquema de seguridad eficiente que proteja los recursos informáticos de las actuales amenazas combinadas.

  El resultado de todo esto es la violación de los sistemas, provocando la pérdida o modificación de los datos sensibles o información de la organización, lo que puede representar un daño valorado en miles o millones de euros.

- **Amenazas externas:** Son aquellas amenazas que se originan fuera de la red. Al no tener conocimiento exacto de la red, un atacante tiene que realizar ciertos pasos para poder conocer qué es lo que hay en ella y buscar la manera de atacarla. La ventaja que se tiene en este tipo de caso es que el administrador de la red puede prevenir buena parte de los ataques externos.

2.3 Problemas que repercuten en la seguridad

En lo que a sistemas operativos se refiere existen diversas clases de Problemas que repercuten en la seguridad. Un alto porcentaje de los ataques que puede sufrir un sistema operativo podría englobarse en uno de los siguientes tres tipos:

- **Ataques de ingeniería social.** El usuario del sistema es el último responsable de mantener una conducta de seguridad adecuada a nivel de equipo y de red, no sirve de nada tener un equipo dotado de la mayor seguridad posible si posteriormente el usuario facilita sus contraseñas.

- **Conectividad.** No es lo mismo mantener un equipo libre de virus de forma aislada que un equipo conectado a la red de redes. El hecho de que un equipo esté conectado a redes como Internet complica la labor de mantener el sistema seguro.

- **Complejidad.** Los sistemas operativos son programas enormemente complejos. Un sistema complejo es mucho más difícil de mantener seguro que un sistema más simple. En la actualidad existen sistemas operativos que se olvidan de las funcionalidades que ha de tener dicho sistema, e incorporan elementos no necesarios acarreando una mayor complejidad, y por lo tanto un mayor número de líneas de código, aumentando así la posibilidad de que existan defectos en la programación y por tanto la seguridad sea más vulnerable.

2.4 Los reglas básicas de la seguridad informática

A pesar de que un sistema no pueda ser al 100% seguro, y de que la tecnología cambie constantemente, existen unas reglas básicas invariables en el tiempo para garantizar de la mejor forma posible la seguridad de un sistema, las mencionaremos a continuación:

1. **Si alguien puede convencerle de que ejecute su programa en su equipo, dejará de ser su equipo.** Los atacantes intentan convencer al usuario a instalar software en nombre de este, o incluso llegan a engañar a los usuarios diciéndoles que dicho software es un programa dado cuando en realidad es una aplicación bien distinta la cual lleva insertada código malicioso.

2. **Si alguien puede modificar el sistema operativo, dejará de ser su equipo.** Si un atacante puede remplazar o modificar cualquiera de los ficheros del sistema operativo, ya no podrá confiar en su equipo. Una vez que el sistema operativo se ha comprometido, ya no se debe confiar en él pues los cambios que se hayan podido realizar en ocasiones no son visibles.

3. **Si alguien tiene acceso físico sin restricciones a su equipo, dejará de ser su equipo.** Una persona con acceso pleno a un sistema operativo puede ser capaz de instalar software o hardware sin que el usuario habitual sea consciente de ello. Al mismo tiempo, un atacante con acceso físico a un equipo obtendrá con toda seguridad privilegios administrativos sobre el sistema operativo con lo que ello supone.

4. **Si permite que alguien cargue programas en su sitio Web, dejará de ser su sitio Web.** Si un atacante puede ejecutar aplicaciones o modificar código en el sitio Web, podrá apropiarse de él.

5. **Las contraseñas débiles anulan una seguridad fuerte.** Si los usuarios o administradores usan contraseñas predeterminadas, en blanco, o basadas en palabras del diccionario, la seguridad existente en el sistema será inútil si un atacante consigue descifrar la contraseña o consigue realizar un ataque de fuerza bruta.

6. **Una máquina es tan segura como digno de confianza sea el administrador.** Cuantos más privilegios administrativos tenga un administrador mayor confianza se ha de tener en este.

7. **Los datos de cifrado son tan seguros como las claves de descifrado.** Al igual que pasaba en puntos anteriores para las contraseñas de usuarios y administrador, por muy bueno que sea un algoritmo de cifrado, no servirá de nada si la contraseña o clave de descifrado cae en manos de un atacante.

8. **Un programa antivirus no actualizado es poco más seguro que no disponer del mismo.** Debido a la aparición continua de virus informáticos, un software antivirus puede quedarse desfasado muy rápidamente, por tanto, un usuario con un antivirus obsoleto corre el riesgo de creer que está protegido y con ello descuidar la seguridad.

9. **El anonimato absoluto no es práctico en la vida real ni en la Web.** Un error común es confundir anonimato con privacidad. Si bien el anonimato significa que la identidad y los detalles sobre un usuario son completamente desconocidos e imposibles de rastrear, la privacidad significa que la identidad y los detalles sobre la misma no se revelan. La privacidad es esencial, y la tecnología y las leyes permiten conseguirlo.

10. **La tecnología no es una panacea.** A pesar de que la tecnología es capaz de asegurar los equipos informáticos, no es una solución por sí misma.

2.5 Principales fallos genéricos en los sistemas operativos

Existen unos fallos genéricos funcionales de los sistemas operativos que aún no se han citado y que se citan a continuación:

- **Autenticación:** Los usuarios no pueden determinar si el hardware y el software con el que funcionan son los que deben ser. Un intruso podría reemplazar un programa sin conocimiento del usuario. Un usuario puede inadvertidamente teclear una contraseña en un programa de entrada falso.

- **Cifrado:** No se almacena cifrada la lista maestra de contraseñas.

- **Implementación:** Implementación improcedente de un buen diseño de seguridad.

- **Confianza implícita:** Una rutina supone que otra está funcionando correctamente cuando, de hecho, debería examinar los parámetros suministrados por la otra rutina.

- **Compartimiento implícito:** El sistema operativo deposita inadvertidamente información importante del sistema en un espacio de direcciones del usuario.

- **Comunicación entre procesos:** Usos inadecuados de los mecanismos de send/receive que pueden ser aprovechados por los intrusos.

- **Verificación de la legalidad:** Validación insuficiente de los parámetros del usuario.

- **Desconexión de línea:** Ante una desconexión de línea el S. O. debería:

  - Dar de baja al usuario (o los usuarios) de la línea.

  - Colocarlos en un estado tal que requieran la re - autorización para obtener nuevamente el control.

- **Descuido del operador:** Un intruso podría engañar a un operador y hacer que le habilite determinados recursos.

- **Paso de parámetros por referencia en función de su valor:** Es más seguro pasar los parámetros directamente en registros que tener los registros apuntando a las áreas que contienen los parámetros. El paso por referencia puede permitir que los parámetros, estando aún en el área del usuario, puedan ser modificados antes de ser usados por el sistema.

- **Contraseñas:** No deben ser fácilmente deducibles u obtenibles mediante ensayos repetidos.

- **Entrampamiento al intruso:** Los sistemas operativos deben tener mecanismos de entrampamiento para atraer al intruso inexperto.

- **Privilegio:** Cuando hay demasiados programas con demasiados privilegios se viola el principio del menor privilegio.

- **Confinamiento del programa:** Un programa *"prestado"* de otro usuario puede actuar como un *"Caballo de Troya".*

- **Prohibiciones:** Se advierte a los usuarios que no utilicen ciertas opciones porque los resultados podrían ser "indeterminados", pero no se bloquea su uso, con lo que puede robar o alterar datos.

- **Residuos:** Un intruso podría encontrar una lista de contraseñas con solo buscar en lugares tales como una "papelera" del sistema o física. La información delicada debe ser sobrescrita o destruida antes de liberar o descartar el medio que ocupa.

- **Blindaje:** Los intrusos pueden conectarse a una línea de transmisión sin hacer contacto físico:

  - Utilizan el campo inducido por la circulación de corriente en un cable.

  - Se previene con un adecuado blindaje eléctrico.

- **Valores de umbral:** Si no se dispone de valores umbral, no habrá:

- Límites al número de intentos fallidos de ingreso.
- Bloqueos a nuevos intentos.
- Comunicaciones al supervisor o administrador del sistema.

2.6 Vulnerabilidades de un sistema operativo

Aunque no sea objeto específico de nuestro proyecto, a continuación vamos a mostrar la forma generalizada que realiza un atacante para introducirse en un sistema ajeno. Se muestra con detalle pues conociendo el método con el cual se introducen en la máquina, es más fácil poder evitar dichas intrusiones.

- **Exploración:**

El primer paso que realiza un atacante es la exploración. Mediante la exploración, el atacante obtiene una lista de "*direcciones IP*" y de red utilizando descargas de transferencias de zonas y consultas "*whois*".

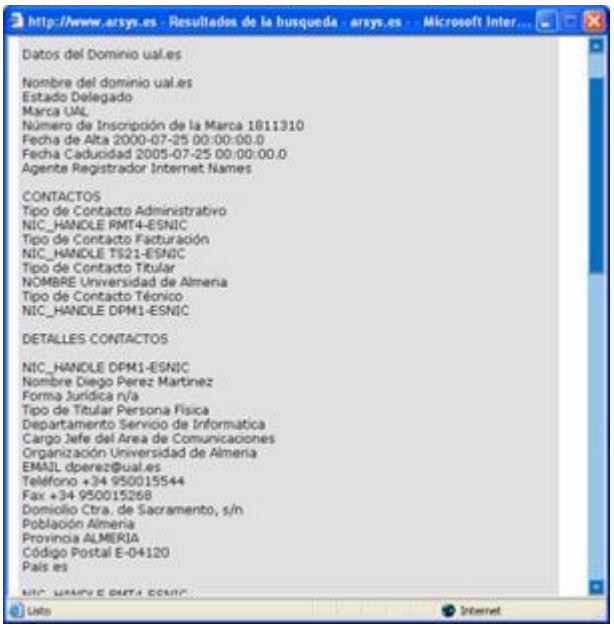


Figura 1. Consulta Whois.

Estas técnicas proporcionan una información valiosa a los atacantes, como puede ser: nombres de empleados, números de teléfono, rangos de direcciones IP, servidores *DNS*, etc.

Una vez que se tiene seleccionada una red, se ha de realizar un barrido de "*ping*" automatizado en un rango de direcciones IP y de bloques de red. "*Ping*" se utiliza tradicionalmente para enviar paquetes *ICMP ECHO_REQUEST* a un sistema destino, en un intento de obtener un *ICMP ECHOREPLY* el cual indica que el sistema destino está activo para una posterior exploración de puertos. Este sistema no sería el recomendable en redes de gran tamaño debido a la alta cantidad de tiempo que puede llevar el barrido.

- **Identificar servicios *TCP/IP* que se ejecutan en el sistema:**

En la exploración de puertos se realiza el proceso de conexión a puertos *UDP y TCP* del sistema destino que nos permite determinar los servicios que se están ejecutando en el sistema. Por ejemplo, si un equipo tiene abierto el puerto 80 entonces sabemos que el equipo tiene un servidor web.

Existen varias herramientas de exploración de puertos, la elección de una de ellas es crítica. En Linux y Windows por ejemplo existe la herramienta "*nmap*" de difícil detección pues fue creada para evadir los sistemas de detección de intrusos.

```
Starting nmap 3.81 ( http://www.insecure.org/nmap/ ) at 2005-05-01 22:12 CEST
Interesting ports on 192.168.0.1:
(The 1646 ports scanned but not shown below are in state: closed)
PORT     STATE SERVICE
21/tcp   open  ftp
53/tcp   open  domain
80/tcp   open  http
135/tcp  open  msrpc
139/tcp  open  netbios-ssn
445/tcp  open  microsoft-ds
1025/tcp open  NFS-or-IIS
1080/tcp open  socks
3001/tcp open  nessusd
3005/tcp open  deslogin
3006/tcp open  deslogind
3372/tcp open  msdtc
3389/tcp open  ms-term-serv
4500/tcp open  sae-urn
4557/tcp open  fax
4559/tcp open  hylafax
8080/tcp open  http-proxy
MAC Address: 00:40:F4:98:E0:62 (Cameo Communications)

Nmap finished: 1 IP address (1 host up) scanned in 0.619 seconds
[root@redhatserver nmap]# _
```

Figura 2. nmap –Sp.

- **Identificar el sistema operativo:**

Para poder identificar el sistema operativo instalado, es necesario utilizar las herramientas de rastreo de pilas. Este método es una técnica extremadamente potente que nos permite averiguar de forma rápida, y con gran porcentaje de éxito, cuál es el sistema operativo instalado en el "*host*". Dicho método se basa en que cada fabricante de sistemas operativos usa una política u otra a la hora del desarrollo de pilas "*IP*", y analizando las diferencias se pueden hacer suposiciones de cuál es el sistema operativo que se está usando. Para obtener la máxima fiabilidad, la técnica de rastreo de pilas necesitará que al menos un puerto esté a la escucha.

Existen diversos tipos de sondeos que podemos enviar al sistema y que nos permiten distinguir un sistema operativo de otro, por ejemplo:

- **Sondeo FIN.** Se envía un "*paquete FIN*" a un puerto abierto. La normativa "*RFC*" establece que el comportamiento correcto es no responder. Sin

embargo, muchos desarrolladores de pilas, responden con un "*paquete FIN/ACK*".

- **Sondeo de Bogus Flag.** Se introduce un flag "*TCP*" indefinida en la cabecera "*TCP*" de un paquete "*SYN*". Algunos sistemas operativos, como Linux, responderán con esta flag en su paquete de respuesta.

- **Muestreo de número de secuencia inicial (ISN).** La premisa básica es encontrar un patrón en la secuencia inicial elegida por la implementación "*TCP*" cuando responde a una petición de conexión.

- **Supervisión de "bit de No Fragmentación".** Algunos sistemas operativos activarán la opción "bit de No Fragmentación" para mejorar su rendimiento. Se puede analizar este bit para determinar qué tipos de sistema operativo se comportan de esta manera.

- **Tamaño de ventana inicial "*TCP*".** Se analiza el tamaño inicial recibido en los paquetes de respuesta. En algunos desarrollos de pilas, este tamaño es único y puede ayudar bastante en la precisión del mecanismo de seguimiento.

- **Valor "*ACK*"**. Las pilas "*IP*" difieren en el valor de la secuencia que usan en el campo "*ACK*", por lo que algunos desarrolladores responderán con el mismo número de secuencia que han recibido, y otros responderán con ese número de secuencia incrementado en una unidad.

- **Apagado del mensaje de error ICMP.** Los sistemas operativos pueden limitar la velocidad con la que se envían los mensajes de error. Si se envían paquetes "*UDP*" a algún puerto aleatorio que tenga asignado un número elevado, es posible contar el número de mensajes recibidos no contestados en un intervalo de tiempo determinado.

- **Cita de mensajes "*ICMP*".** Los sistemas operativos difieren en la cantidad de información que se cita cuando aparecen mensajes de error "*ICMP*". Si se examina el mensaje citado, se podrán realizar ciertas hipótesis sobre cuál es el sistema operativo. En Linux existen varias herramientas que nos permiten detectar cuál es el sistema operativo de un equipo remoto: *xprobe 2* y *Nmap* por ejemplo.

- **Integridad del eco de los mensajes de error ICMP**. Algunos desarrollos de pilas pueden alterar las cabeceras IP cuando devuelven mensajes de error ICMP. Si examinamos las modificaciones producidas en las cabeceras, se podrán realizar ciertas hipótesis sobre cuál es el sistema operativo destino.

- **Tipo de servicio (TOS).** En los mensajes del tipo "ICMP port unreachable", se pueden examinar el "*TOS*". La mayoría de pilas utilizan un 0 pero existen otras posibilidades.

- **Control de fragmentación.** Cada pila maneja de forma diferente los fragmentos de un mensaje. Algunas pilas cuando recomponen los fragmentos,

escriben datos nuevos encima de los viejos, y viceversa. Analizando cómo se recomponen los paquetes de sondeo, se podrán realizar ciertas hipótesis sobre el sistema operativo objetivo.

- **Opciones TCP.** Las opciones "*TCP*" se definen más recientemente en "*RFC 1323*". Las opciones más avanzadas proporcionadas por RFC 1323 suelen encontrarse en los desarrollos de pila más modernos. Enviando un paquete para el que se han definido una serie de opciones es posible realizar alguna hipótesis sobre cuál es el sistema operativo objeto.

- **Identificar las aplicaciones de un servicio:**

Una vez que el atacante ha identificado el sistema operativo, puede identificar la versión de un servicio o aplicación, y utilizar dicha información para aprovechar las vulnerabilidades de dicho servicio.

La aplicación "*nmap*" de Linux permite averiguar qué aplicaciones están ejecutándose en un puerto determinado tanto en nuestra máquina como en una máquina remota.

```
  TARGET PORT   The target address and port(s) to scan (additional to -i)
amap is a tool to identify application protocols on target ports.
Usage hint: Options "-bqv" are recommended, add "-1" for fast/rush checks.
[root@redhatserver amap-5.0]# amap -bqv 192.168.0.1 80
Using trigger file ./appdefs.trig ... loaded 23 triggers
Using response file ./appdefs.resp ... loaded 309 responses
Using trigger file ./appdefs.rpc ... loaded 450 triggers

amap v5.0 (www.thc.org/thc-amap) started at 2005-05-01 22:19:29 - MAPPING mode

Total amount of tasks to perform in plain connect mode: 17
Protocol on 192.168.0.1:80/tcp (by trigger http) matches http - banner: HTTP/1.1
 404 Objeto no encontrado\r\nServer Microsoft-IIS/5.0\r\nDate Sun, 01 May 2005 2
03046 GMT\r\nContent-Type text/html\r\nContent-Length 116\r\n\r\n<html><head><ti
tle>Sitio no encontrado</title></head>\n<body>No hay ningn sitio Web en esta dir
eccin.
Protocol on 192.168.0.1:80/tcp (by trigger http) matches http-iis - banner: HTTP
/1.1 404 Objeto no encontrado\r\nServer Microsoft-IIS/5.0\r\nDate Sun, 01 May 20
05 203046 GMT\r\nContent-Type text/html\r\nContent-Length 116\r\n\r\n<html><head
><title>Sitio no encontrado</title></head>\n<body>No hay ningn sitio Web en esta
 direccin.
Waiting for timeout on 16 connections ...

amap v5.0 finished at 2005-05-01 22:19:35
[root@redhatserver amap-5.0]#
```

Figura 3. amap.

- **Identificar las vulnerabilidades del sistema:**

Una vez llegamos a este punto en el que un atacante ha detectado los equipos que hay activos en una red, el sistema operativo de cada equipo y hasta el tipo de servidor que tiene instalado, lo único que queda es saber las vulnerabilidades del sistema, para ello basta con hacer algo tan simple como buscarlo en Internet en diversas páginas web como por ejemplo "www.thc.org" [3].

Figura 4. www.thc.org. [3]

Otra forma de detectar las vulnerabilidades, por ejemplo, podríamos utilizar una herramienta que nos muestre automáticamente las vulnerabilidades de un servidor. Existen varias herramientas para tal fin. Por ejemplo, “LANguard” es una herramienta que nos proporcionará todo tipo de información del sistema, como por ejemplo: actualizaciones instaladas, usuarios, grupos de usuarios, etc.

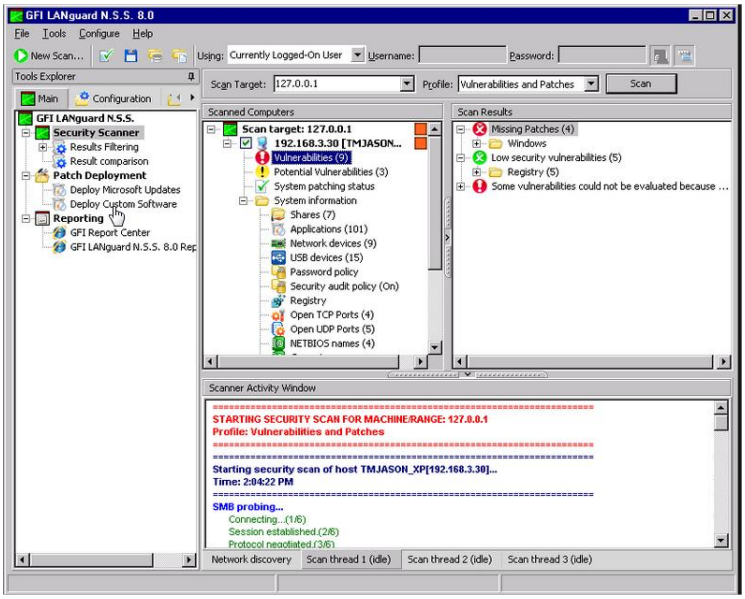


Figura 5. LANguard.

Una de las herramientas preferidas por los atacantes es “*Nessus*”, se basa en el modelo “cliente / servidor”. El servidor tiene que estar activo en una máquina Linux mientras que el cliente puede estar en Windows o en Linux.

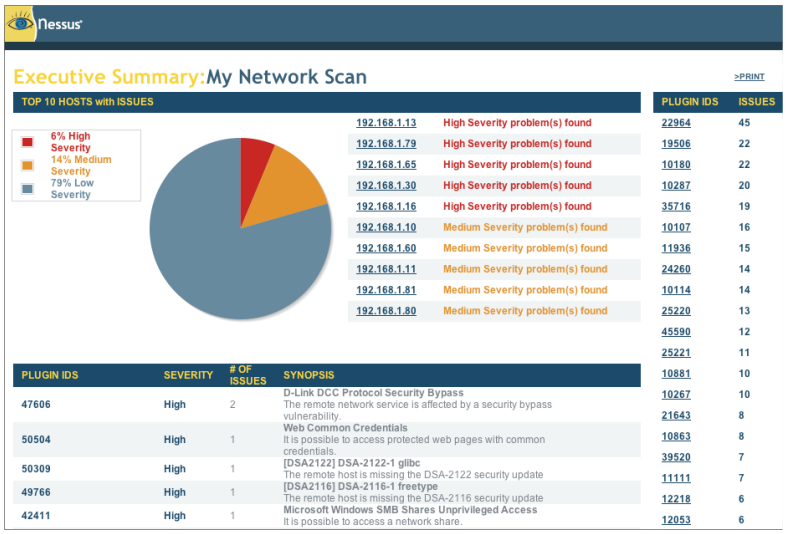


Figura 6. Nessus.

2.7 Tipos de ataques a sistemas operativos

Entendemos por amenaza una condición del entorno del sistema de información (por ejemplo: persona, máquina, idea, etc.) que dada una oportunidad, se podría dar lugar a que se produjese una violación de seguridad (confidencialidad, integridad, disponibilidad o uso legítimo).

Las políticas de seguridad y el análisis de riesgos habrán identificado las amenazas que han de ser contrarrestadas, dependiendo del diseño del sistema de seguridad, se especifican los servicios y mecanismos de seguridad necesarios.

Las amenazas de seguridad se pueden caracterizar modelando el sistema como un flujo de información, desde una fuente, como puede ser un fichero o una región de la memoria principal por ejemplo, a un destino, como puede ser otro fichero o un usuario.

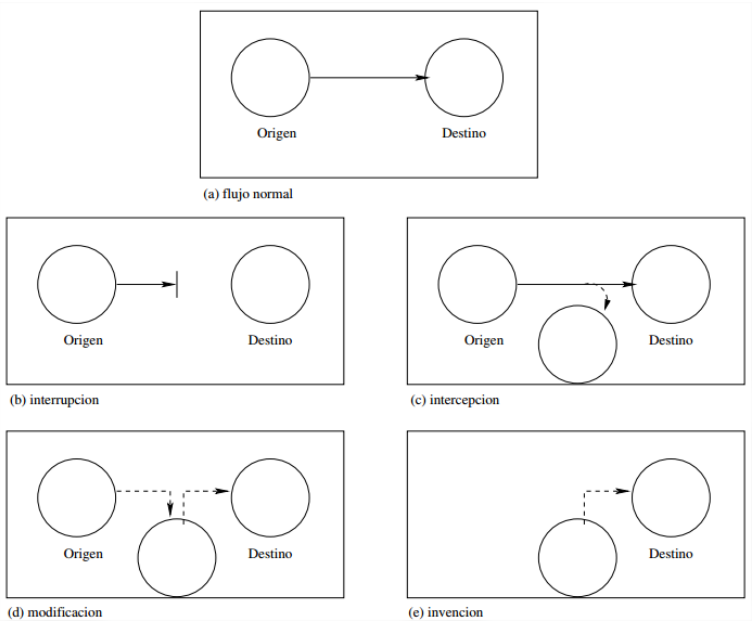


Figura 7. Tipos de amenazas.

Tal y como puede verse en la figura, las cuatro categorías generales de amenazas o ataques son las siguientes:

- **Interrupción:** Un recurso del sistema es destruido o se vuelve no disponible. Éste es un ataque contra la disponibilidad. Ejemplos de este tipo de ataque pueden ser la destrucción de un elemento hardware, por ejemplo un disco duro, o cortar una línea de comunicación o deshabilitar el sistema de gestión de ficheros.

- **Intercepción:** Una entidad no autorizada consigue acceso a un recurso. Éste es un ataque contra la confidencialidad. La entidad no autorizada puede ser una persona, un programa o un ordenador. Ejemplos de este tipo de ataque son por ejemplo, interceptar una línea para hacerse con datos que circulen por la red y la copia ilícita de ficheros o programas (intercepción de datos), o por ejemplo la lectura de las cabeceras de paquetes para desvelar la identidad de uno o más de los usuarios implicados en la comunicación observada ilegalmente (intercepción de identidad).

- **Modificación:** Una entidad no autorizada no sólo consigue acceder a un recurso, sino que puede ser capaz de manipularlo. Este es un ataque contra la integridad. Ejemplos de este ataque pueden ser alterar un programa para que funcione de forma diferente, o por ejemplo modificar el contenido de un fichero o de un mensaje transferido por la red.

- **Invención:** Un ataque contra la autenticidad es cuando una entidad no autorizada inserta objetos falsificados en el sistema. Ejemplos de este tipo de ataque pueden ser la inserción de mensajes basura en una red o añadir registros a un fichero por ejemplo.

Estos ataques pueden asimismo clasificarse de forma útil en términos de ataques pasivos y ataques activos.

- **Ataques Pasivos:** En este tipo de ataques, el atacante no altera la comunicación sino que únicamente la escucha o monitoriza para obtener información de lo que está siendo transmitido. Sus objetivos son la intercepción de datos y el análisis de tráfico, una técnica más sutil para obtener información de la comunicación por ejemplo puede consistir en:

    - La obtención de origen y destinatario de la comunicación, leyendo las cabeceras de los paquetes monitorizados.

    - El control de volumen de tráfico intercambiado entre las entidades monitorizadas, obteniendo así información acerca de actividad o inactividad inusuales.

    - El control de las horas habituales de intercambio de datos entre las entidades de comunicación, para así extraer información acerca de los períodos de actividad.

Estos tipos de ataques son muy difíciles de detectar, ya que no provocan ninguna alteración en los datos. Sin embargo, es posible evitar su éxito mediante el cifrado de información y otros mecanismos que se mencionarán más adelante.

- **Ataques Activos:** Este tipo de ataques implican algún tipo de modificación del flujo de datos transmitido o la creación de un falso flujo de datos, pudiendo subdividirse en cuatro categorías:

  - **Suplantación de identidad**: En este tipo de ataques, el atacante se hace pasar por una entidad diferente. Normalmente incluye alguna de las otras formas de ataque activo. Un ejemplo de este tipo de ataque sería, las secuencias de autentificación pueden ser capturadas y repetidas, permitiendo al atacante acceder a una serie de recursos privilegiados suplantando a la entidad que posee esos privilegios, como ocurre al robar la contraseña de acceso a una cuenta por ejemplo.

  - **Reactuación**: En este tipo de ataques, uno o varios mensajes legítimos son capturados y repetidos para producir un efecto no deseado, como por ejemplo ingresar dinero repetidas veces en una cuenta.

  - **Modificación de mensajes**: En este tipo de ataques, una porción del mensaje legítimo es alterada, o los mensajes son retardados o reordenados, para producir un efecto no autorizado. Un ejemplo de este tipo de ataques sería, el mensaje "Ingresa un millón de euros en la cuenta A" se modificaría para decir "Ingresa un millón de euros en la cuenta B".

  - **Degradación fraudulenta del servicio**: En este tipo de ataques se impide o inhibe el uso normal o la gestión de recursos informáticos y de comunicaciones. Un ejemplo de este tipo de ataques sería, el intruso suprimiría todos los mensajes dirigidos a una determinada entidad o se podría interrumpir el servicio de una red inundándola con mensajes basura. Entre estos tipos de ataques se encuentran los famosos ataques de denegación de servicio que consisten en paralizar temporalmente el servicio de un servidor de correo, Web, FTP, etc.

A continuación mostraremos una tabla resumen de los ataques más usuales:

Nombre	Descripción	Solución
Sniffing.	Consiste en la escucha de datos que atraviesan una red. Se usan para obtener passwords.	El cifrado de datos o usar passwords no reusables (S/key).
Spoofing.	El atacante envía paquetes con una dirección fuente incorrecta. Las respuestas se envían a la dirección falsa. Pueden usarse para: ▪ Cifrado del protocolo. ▪ Acceder a recursos confiados sin privilegios para DoS (Denial of Service) directo	El cifrado del protocolo.

	como indirecto o recursivo.	
Hijacking.	Permiten a un usuario robar una conexión de un usuario que ha sido autentificado en el sistema.	El filtrado de paquetes. Criptografía a nivel de IP (IPsec) o a nivel de aplicación (SSL, TSL, SSH).
Explotar bugs del software.	Consiste en aprovechar los fallos del software para poder hacerse con el control de la máquina.	El actualizar el software o las baterías de tests.
Negación de servicio.	Consiste en bloquear un determinado número de servicios para que los usuarios legítimos no los puedan usar.	Eliminar paquetes con direcciones falsas. Los servidores deben limitar el número de recursos reservados por una única entidad.
Ingeniería social.	Consiste en convencer a un usuario legítimo a que le facilite información (contraseñas, configuraciones, etc.)	La autentificación e información.
Phising.	Es una variante de la ingeniería social. A través de mensajes de texto “falsificados” y sitios Web fraudulentos engañan a los usuarios con el fin de que revelen datos financieros, datos personales, contraseñas, etc.	Autentificación e información.
Confianza transitiva.	Consiste en aprovechar la confianza UNIX entre usuarios o hosts para tomar sus privilegios.	La autentificación y filtrado de paquetes.
Ataques dirigidos por datos.	Virus, gusanos, páginas de javaScript.	Los antivirus, firma digital e información.
Troyanos.	Software que se instala en el ordenador atacado que permite al atacante hacerse con el control de la máquina.	La firma digital, verificación de software y filtrado de paquetes.
Mensajes de control de red o enrutamiento fuente.	Consiste en enviar paquetes ICMP para hacer pasar los paquetes por un router comprometido.	Filtrado de paquetes y cifrado.
Reenvío de paquetes.	Consiste en la retransmisión de paquetes para engañar o duplicar un mensaje.	Rechazar paquetes duplicados, usando marcas de tiempo o número de secuencia.
Adivinación de password.	Fuerza bruta o ataques basados en diccionarios.	Información y firma digital.
Rubber-hosse.	Consiste en utilizar soborno o tortura para obtener una determinada información.	Defensa personal.

Tabla 1. Tipos de ataques. [4]

2.8 Posibles o futuros problemas y amenazas

Nuestro sistema no solo está expuesto a amenazas por parte de intrusos o atacantes, existen diversos problemas los cuales se muestran a continuación:

- **Personas.** El principal problema para un sistema operativo proviene de las personas que de forma intencionada o no (por desconocimiento de lo que hacen) pueden causar daño al equipo.

- **Amenazas lógicas.** Son programas que mediante diversas formas dañan nuestro equipo, normalmente de forma intencionada. Es lo que comúnmente se conocen de forma generalizada como virus.

- **Catástrofes.** Este grupo se puede subdividir a su vez en catástrofes naturales como puedan ser incendios, terremotos, inundaciones... o catástrofes artificiales como puedan ser exceso de tensión eléctrica, incendios provocados, campos magnéticos que borran la información del disco duro, etc.

Actualmente las tendencias cibercriminales indican que la nueva modalidad es manipular los significados de la información. Lo indicado anteriormente se convirtió en el núcleo de los ataques debido a la evolución de la Web 2.0 y las redes sociales, factores que llevaron al nacimiento de la generación 3.0.

En este sentido, las amenazas informáticas que vienen en el futuro ya no son con la inclusión de troyanos en los sistemas o software espías, sino con el hecho de que los ataques se han profesionalizado y manipulan el significado del contenido virtual.

Para no ser presa de esta nueva ola de ataques más sutiles, Se recomienda:

- El mantenimiento de las soluciones activadas y actualizadas.

- Evitar realizar operaciones comerciales en equipos de uso público.

- La verificación de los archivos adjuntos de mensajes sospechosos y evitar su descarga en caso de duda.

Capítulo 3

Auditoría Informática

3 Introducción a la Auditoría Informática

3.1 Objetivos de la Auditoría Informática

Para la realización de este capítulo de nuestro proyecto fin de carrera, hemos recopilado y estudiado información de las siguientes referencias [5] y [6].

Entre las múltiples definiciones de Auditoría Informática existentes, se recogen a continuación varias:

- “Proceso sistemático, independiente y documentado para obtener evidencias de auditoría y evaluarlas de manera objetiva para determinar el grado en que los criterios de auditoría se han cumplido”.

  (ISO 19011)

- “Este proceso recoge y evalúa las evidencias para determinar si los sistemas de información y recursos relacionados, protegen adecuadamente los activos, mantienen los datos y la integridad, la disponibilidad del sistema, proporcionan información relevante y confiable, alcanzan los objetivos de la organización con eficacia, consumen recursos de manera eficiente, y en efecto, los controles internos que proporcionan una seguridad razonable de

que los objetivos de negocio, operativos y de control, se cumplan, y que los eventos no deseados se preverán, o detectarán y corregirán, de manera oportuna".

(ISACA [IS Audit])

- "Examen de la información por terceras partes, distintas de quien la genera y la utiliza, con la intención de establecer su suficiencia y adecuación, e informar de los resultados del examen con objeto de mejorar su utilidad".

(Thomas Porter, 1971. Citado en [RAM092, Introducción])

3.2 Las líneas de defensa

Existe un concepto que va unido al de auditoría informática, éste es el de las líneas de defensa. Las tres líneas de defensa se complementan, ejerciendo de barreras para prevenir y detectar el error / fraude en los entornos informáticos:

3.2.1 El control Interno

El control interno Informático puede definirse como el sistema integrado al proceso administrativo, en la planeación, organización, dirección y control de las operaciones con el objeto de asegurar la protección de todos los recursos informáticos y mejorar los índices de economía, eficiencia y efectividad de los procesos operativos automatizados.

(Auditoría Informática - Aplicaciones en Producción - José Dagoberto Pinilla)

El Informe COSO define el Control Interno como "Las normas, los procedimientos, las prácticas y las estructuras organizativas diseñadas para proporcionar seguridad razonable de que los objetivos de la empresa se alcanzarán y que los eventos no deseados se preverán, se detectarán y se corregirán.

También se puede definir el Control Interno como cualquier actividad o acción realizada manual y/o automáticamente para prevenir, corregir errores o irregularidades que puedan afectar al funcionamiento de un sistema para conseguir sus objetivos.

(Auditoría Informática - Un Enfoque Práctico - Mario G. Plattini)

Objetivos principales:

- Controlar que todas las actividades se realizan cumpliendo los procedimientos y normas fijados, y asegurarse del cumplimiento de las normas legales.
- Asesorar sobre el conocimiento de las normas.
- Colaborar y apoyar el trabajo de Auditoría Informática interna/externa.
- Definir, implantar y ejecutar mecanismos y controles para comprobar el grado de cumplimiento de los servicios informáticos.

3.2.2 Tipos de control Interno

En informática, el control interno se materializa fundamentalmente en controles de dos tipos:

- **Controles manuales**: aquellos ejecutados por el personal usuario o de informática sin la utilización de herramientas computacionales.

- **Controles Automáticos**: generalmente incorporados en el software, llámense estos de operación, de comunicación, de gestión de base de datos, programas de aplicación, etc.

Los controles según su finalidad se clasifican en:

- **Controles Preventivos**: tratan de evitar la producción de errores o hechos fraudulentos, como por ejemplo el software de seguridad que evita el acceso a personal no autorizado.

- **Controles Detectivos**: tratan de descubrir a posteriori errores o fraudes que no haya sido posible evitarlos con controles preventivos.

- **Controles Correctivos**: tratan de asegurar que se subsanen todos los errores identificados mediante los controles detectivos.

- **Controles Directivos**: están constituidos por acciones, políticas, procedimientos, directivas o líneas guía que causan o estimulan la ocurrencia de eventos deseados. Por su naturaleza, estos controles afectan a todo el proyecto. Los controles directivos implantan los cinco atributos UMACS del software.

- **Controles Preventivos**: estos controles incluyen estándares, líneas guía, métodos, prácticas o herramientas y técnicas que darán como resultado aplicaciones fiables. Minimizan la ocurrencia de eventos indeseables tales como fraudes, robos o desfalcos, así como posibles errores, irregularidades y omisiones. Los password o palabras de paso son un ejemplo de controles preventivos que implantan los aspectos de mantenimiento, seguridad y control en el sistema.

- **Controles Detectivos**: detectan errores, irregularidades, y omisiones, e identifican los aspectos del sistema en cuanto a calidad, control y seguridad que necesitan de la atención de los directivos. Determinan si los controles preventivos y/o directivos del sistema han alcanzado sus objetivos y si los estándares o líneas guía han sido respetados. Estos controles incluyen técnicas manuales y automáticas. Proveen información sobre si las pistas de auditoría son adecuadas y completas, haciendo la aplicación auditable, segura y controlable.

- **Controles Correctivos**: indican procedimientos, información e instrucciones para corregir los errores, irregularidades, y omisiones que han sido detectadas. Pueden identificar las áreas donde es necesaria una acción correctiva o pueden facilitar la acción correctiva. También incluyen herramientas, técnicas manuales y automáticas. Implantan la usabilidad y auditabilidad en la aplicación.

- **Controles de Recuperación**: facilitan la copia de seguridad, restauración, recuperación y reanudación de una aplicación después de cualquier interrupción en el proceso. Fomenta la creación de un entorno ordenado en el cual todos los recursos requeridos están realmente disponibles para asegurar una recuperación razonablemente fácil de un desastre, lo que permite la continuación de una actividad específica o de todas las operaciones de la organización. Estos controles incluyen copias de seguridad cada cierto tiempo y rotación de datos y ficheros de programas, procedimientos de reanudación/reinicio, y registro y conservación de ficheros. Hacen aplicaciones usables, auditables, controlables y seguras.

3.2.3 Funciones Control interno

El objetivo del control interno es el de controlar que todas las actividades relacionadas con los sistemas de información automatizados se realicen en cumplimiento de las normas, estándares, procedimientos y disposiciones legales establecidas interna y externamente. Entre sus funciones específicas se encuentran:

- Difundir y controlar el cumplimiento de las normas, estándares y procedimientos de los programadores, técnicos y operadores.

- Diseñar la estructura del sistema de controles internos de la Dirección de Informática en los siguientes aspectos:

  - Desarrollo y mantenimiento del software de aplicación.

  - Explotación de servidores principales.

  - Software de Base.

  - Redes de Computación.

  - Seguridad Informática.

  - Licencias de software.

  - Relaciones contractuales con terceros.

  - Conocimiento de riesgo informático en la organización.

3.2.4 Áreas de aplicación del Control interno informático

Son la base para la planificar, controlar y evaluar por parte de la Dirección General, las actividades del Departamento de Informática, y debe contener la siguiente planificación:

- Plan Estratégico de Información realizado por el Comité de Informática.
- Plan Informático, realizado por el Departamento de Informática.
- Plan General de Seguridad (física y lógica).
- Plan de Contingencia ante desastres.

Los controles dependiendo del área de aplicación son:

Controles de desarrollo y mantenimiento de sistemas de información. Permiten alcanzar la eficacia del sistema, economía, eficiencia, integridad de datos, protección de recursos y cumplimiento con las leyes y regulaciones a través de metodologías como la del Ciclo de Vida de Desarrollo de aplicaciones por ejemplo.

Controles de explotación de sistemas de información. Tienen que ver con la gestión de los recursos tanto a nivel de planificación, adquisición y uso del hardware así como los procedimientos de, instalación y ejecución del software.

Controles en aplicaciones. Toda aplicación debe llevar controles incorporados para garantizar la entrada, actualización, salida, validez y mantenimiento completos y exactos de los datos.

Controles en sistemas de gestión de base de datos. Afectan a la administración de los datos para asegurar su integridad, disponibilidad y seguridad.

Controles informáticos sobre redes. Repercuten sobre el diseño, instalación, mantenimiento, seguridad y funcionamiento de las redes instaladas en una organización sean estas centrales y/o distribuidos.

Controles sobre computadores y redes de área local. Se relacionan a las políticas de adquisición, instalación y soporte técnico, tanto del hardware como del software de usuario, así como la seguridad de los datos que en ellos se procesan.

3.2.5 La Auditoría Informática Interna

Podríamos definirla como el control de los controles (Control Interno incluido). El objetivo de la auditoría informática interna es ayudar a la Dirección de la entidad de forma objetiva en el cumplimiento de sus responsabilidades y contribuir a que se logren resultados óptimos en el área informática y todo lo relacionado con esta última, proporcionando a la Dirección de la entidad el análisis de objetivos, evaluaciones y recomendaciones. En definitiva, su finalidad es prestar un servicio de asistencia y crítica constructiva.

En relación con el control interno, se deberá determinar por medio de revisiones periódicas a los sistemas de procedimientos operativos, si dicho control proporciona la protección necesaria, así como la máxima eficacia operativa. Además deberá comprobar mediante verificaciones efectuadas sobre las transacciones, si los procedimientos operativos y métodos se utilizan tal y como está establecido por las normas de la entidad. La unidad de auditoría interna no puede en ningún caso tomar parte en funciones de tipo operativo.

Cada vez es más habitual que la auditoría interna llegue a estar orientada al servicio y se afane por aumentar su valor dentro de la organización. Por tanto, la auditoría interna se ha desarrollado para ayudar a la Dirección en un empeño efectivo, siendo sus responsabilidades las siguientes:

- Evaluar la suficiencia del sistema de control interno.
- Evaluar la formalidad y la honradez de los datos financieros y operativos generados por la Dirección.
- Evaluar el cumplimiento de las metas y objetivos del negocio.
- Evaluar el uso de los recursos de la organización de forma eficiente.
- Prevenir y detectar fraudes.
- Coordinación con los auditores externos.

3.2.6 La Auditoría Informática Externa

Es realizada por una persona, física o jurídica, ajena a la entidad.

Las relaciones entre este tipo de auditoría y la mencionada en el punto anterior deben ser de máxima colaboración al ejercer cometidos complementarios.

Si la labor de los auditores internos es eficaz, el trabajo de los auditores externos será menos complicado y más efectivo.

Los auditores internos deben ser los interlocutores de los externos, estos últimos no deben perder su objetividad, informando de situaciones de deficiencia achacables a los internos si fuera necesario.

Pocas son las ocasiones en las que estos dos tipos de auditoría trabajan de forma conjunta, pero en el momento en que se produce este hecho, el objetivo de la coordinación entre auditoría interna y externa es para lo siguiente:

- Reducir al máximo los esfuerzos dobles.
- Fomentar una auditoría esencial que cubra todos los espacios.

Para que se produzca una coordinación buena de los esfuerzos entre los dos tipos de auditoría, debe producirse lo siguiente:

- Reuniones periódicas para planificar una auditoría completa y asegurar una cobertura adecuada. Todo esto se alcanza con la menor cantidad de esfuerzo.
- Reuniones periódicas para discutir materias de interés mutuo.
- Acceso mutuo para revisar programas y papeles de trabajo.
- Intercambios de información sobre informes.
- Común entendimiento y frecuente desarrollo de técnicas de revisión, métodos y terminología.

La Dirección de la entidad, está autorizada a esperar que los auditores externos reconozcan el trabajo realizado por los auditores internos basados en normas profesionales.

El trabajo realizado por los auditores internos puede ser un factor decisivo, oportuno y al alcance de los procedimientos de los auditores externos.

Si los auditores externos determinan que el trabajo de los auditores internos puede tener una acción en sus procedimientos de auditoría, ellos considerarían la competencia y objetividad de los auditores internos.

Además los auditores externos deberían examinar, sobre una base de pruebas, la evidencia documental del trabajo desempeñado por auditores internos y deberían considerar otros factores, tales como los siguientes:

- La adecuación del alcance del trabajo.
- Adecuado programa de trabajo.
- La suficiencia de la documentación suplementaría de auditoría y papeles de trabajo.
- Alcance de las conclusiones.
- Consistencia de los informes con los resultados del trabajo realizado.

Los auditores internos tienen una responsabilidad similar para evaluar cuidadosamente el trabajo de los auditores externos y deberían seguir los mismos pasos de revisión comentados arriba para efectuar su responsabilidad.

En la mayoría de los casos los auditores internos tienen más conocimiento de la entidad a auditar que los externos, ya que ellos mismos forman parte de la plantilla de esa entidad.

3.3 Tipos de Auditoría Informática

Dentro de la auditoría informática destacan los siguientes tipos:

- **Auditoría de la gestión.** Dirección y gestión del sistema de información, gestión de recursos humanos, la contratación de bienes y servicios, documentación de los programas, gestión de calidad (tanto en desarrollo como en producción), etc.

- **Auditoría legal del Reglamento de Protección de Datos.** Cumplimiento legal de las medidas de seguridad exigidas por el Reglamento de desarrollo de la Ley Orgánica de Protección de Datos (LOPD).

- **Auditoría de la organización y planificación.** La seguridad, el desarrollo, explotación, normativa, gestión de recursos, recursos humanos, contratación, calidad, productividad, datos y bases de datos, comunicaciones y redes, comercio electrónico, datos personales.

- **Auditoría de los datos.** Proceso de datos, verificar la clasificación de los datos, estudio de las aplicaciones y análisis de los flujogramas, verificación del cumplimiento de procedimientos, evaluación de la utilidad de controles (preparación, entrada, proceso y salida).

- **Auditoría de las bases de datos.** Controles de acceso, de actualización, de integridad y calidad de los datos.

- **Auditoría de la seguridad.** Referidos a datos e información, verificando disponibilidad, integridad, confidencialidad, autenticación y no repudio.

- **Auditoría de la seguridad física.** Referido a la ubicación de la organización, evitando ubicaciones de riesgo, y en algunos casos no revelando la situación física de esta. También está referida a las protecciones externas (arcos de seguridad, CCTV, vigilantes, etc.) y protecciones del entorno.

- **Auditoría de la seguridad lógica.** Comprende los métodos de autenticación de los sistemas de información.

- **Auditoría de las comunicaciones.** Se refiere a la auditoria de los procesos de autenticación en los sistemas de comunicación, vulnerabilidades de redes, confidencialidad e integridad.

- **Auditoría de la seguridad en producción.** Frente a errores, accidentes y fraudes.

- **Auditoría de la seguridad en aplicaciones.** Frente a errores y fraudes, seguridad en desarrollo, aplicaciones y paquetes de aplicaciones.

Capítulo 4

La Plataforma Android

4 Android

Para la realización de este capítulo de nuestro proyecto fin de carrera, hemos recopilado y estudiado información de las siguientes referencias [7], [8], [9], [10], [11], [12], [25] y [26].

Android constituye una pila de software pensada especialmente para dispositivos móviles y que incluye tanto un sistema operativo, como *middleware* y diversas aplicaciones de usuario. Representa la primera incursión seria de Google en el mercado móvil y nace con la pretensión de extender su filosofía a dicho sector.

Todas las aplicaciones para Android se programan en lenguaje Java y son ejecutadas en una máquina virtual especialmente diseñada para esta plataforma, que ha sido bautizada con el nombre de Dalvik. El núcleo de Android está basado en Linux 2.6.

La licencia de distribución elegida para Android ha sido Apache 2.0, lo que lo convierte en software de libre distribución. A los desarrolladores se les proporciona de forma gratuita un SDK y la opción de unos *plug-in* para el entorno de desarrollo Eclipse, varias que incluyen todas las API necesarias para la creación de aplicaciones, así como un emulador integrado para su ejecución. Existe además disponible una amplia documentación de respaldo para este SDK.

El proyecto Android está capitaneado por Google y un conglomerado de otras empresas tecnológicas agrupadas bajo el nombre de *Open Handset Alliance* (OHA). El objetivo principal de esta alianza empresarial (que incluye a fabricantes de dispositivos y operadores, con firmas tan relevantes como Samsung, LG, Telefónica, Intel o Texas Instruments, entre otras muchas) es el desarrollo de estándares abiertos para la telefonía móvil como medida para incentivar su desarrollo y para mejorar la experiencia del usuario. La plataforma Android constituye su primera contribución en este sentido.

Cuando en noviembre de 2007 Google anunció su irrupción en el mundo de la telefonía móvil a través de Android, muchos medios especializados catalogaron este novedoso producto como un nuevo sistema operativo, libre y específico para teléfonos móviles. Sin embargo, los responsables del proyecto se han esforzado desde entonces en destacar que la motivación de Android es convertirse en algo más que un simple sistema operativo.

Con Android se busca reunir en una misma plataforma todos los elementos necesarios que permitan al desarrollador controlar y aprovechar al máximo cualquier funcionalidad ofrecida por un dispositivo móvil (llamadas, mensajes de texto, cámara, agenda de contactos, conexión Wi-Fi, Bluetooth, aplicaciones ofimáticas, videojuegos, etc.), así como poder crear aplicaciones que sean verdaderamente portables, reutilizables y de rápido desarrollo. En otras palabras, Android quiere mejorar y estandarizar el desarrollo de aplicaciones para cualquier dispositivo móvil y, por ende, acabar con la perjudicial fragmentación existente hoy día.

Además de todo ello, otro aspecto básico para entender la aparición de Android es que pretende facilitar la integración de estos dispositivos con las posibilidades cada día mayores ofrecidas por la Web. Por ejemplo, una aplicación desarrollada en Android podría ser aquella que indicase al usuario, a través de Google Maps, la localización de sus diferentes contactos de la agenda y que avisase cuando éstos se encuentren a una distancia cercana o en una ubicación determinada.

Mejorar el desarrollo y enriquecer la experiencia del usuario se convierte, por tanto, en la gran filosofía de Android y en su principal objetivo.

4.1 La arquitectura Android

Como ya se ha mencionado, Android es una plataforma para dispositivos móviles que contiene una pila de software donde se incluye un sistema operativo, *middleware* y aplicaciones básicas para el usuario. Su diseño cuenta, entre otras, con las siguientes características:

- Busca el desarrollo rápido de aplicaciones, que sean reutilizables y verdaderamente portables entre diferentes dispositivos.
- Los componentes básicos de las aplicaciones se pueden sustituir fácilmente por otros.
- Cuenta con su propia máquina virtual, Dalvik, que interpreta y ejecuta código escrito en Java.
- Permite la representación de gráficos 2D y 3D.
- Posibilita el uso de bases de datos.
- Soporta un elevado número de formatos multimedia.
- Servicio de localización GSM.
- Controla los diferentes elementos hardware: Bluetooth, Wi-Fi, cámara fotográfica o de vídeo, GPS, acelerómetro, infrarrojos, etc., siempre y cuando el dispositivo móvil lo contemple.
- Cuenta con un entorno de desarrollo muy cuidado mediante un SDK disponible de forma gratuita.
- Ofrece un *plug*-in para uno de los entornos de desarrollo más populares, Eclipse, y un emulador integrado para ejecutar las aplicaciones.

En las siguientes líneas se dará una visión global por capas de cuál es la arquitectura empleada en Android. Cada una de estas capas utiliza servicios ofrecidos por las anteriores, y ofrece a su vez los suyos propios a las capas de niveles superiores.

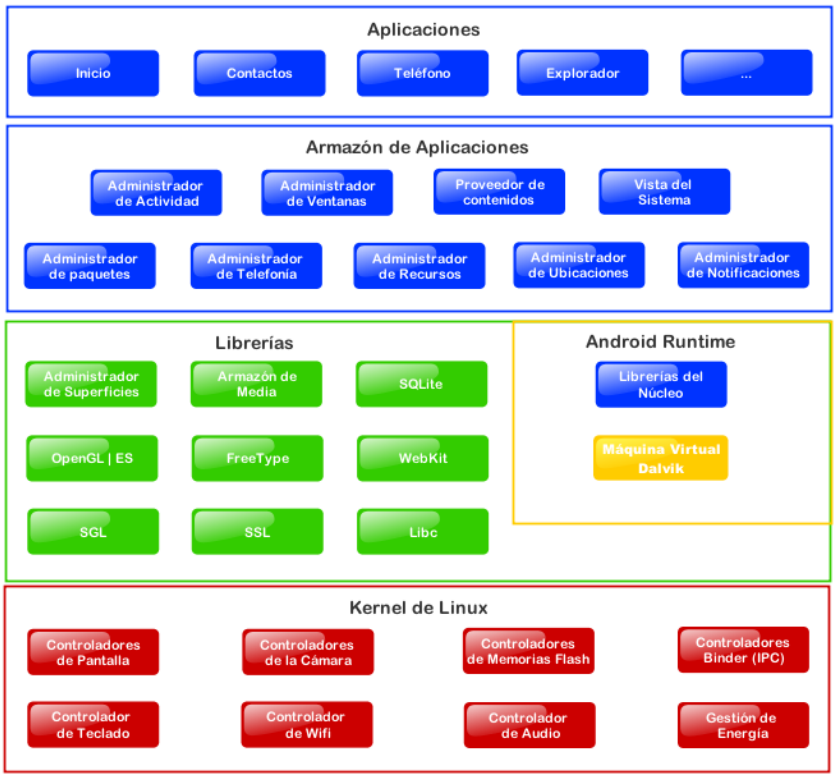


Figura 8. Arquitectura de Android. [7]

4.1.1 Capa Kernel de Linux

La capa más inmediata es la corresponde al **núcleo de Android**. Android utiliza el núcleo de Linux 2.6 como una capa de abstracción para el hardware disponible en los dispositivos móviles. Esta capa contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes.

Siempre que un fabricante incluya un nuevo elemento de hardware, lo primero que se debe realizar para que pueda ser utilizado desde Android es crear las librerías de control o drivers necesarios dentro de este kernel de Linux embebido en el propio Android.

La elección de Linux 2.6 se ha debido principalmente a dos razones: la primera, su naturaleza de código abierto y libre se ajusta al tipo de distribución que se buscaba para Android (cualquier otra opción comercial disponible hoy día hubiera comprometido la licencia de Apache); la segunda es que este kernel de Linux incluye de por sí numerosos drivers, además de contemplar la gestión de memoria, gestión de procesos, módulos de seguridad, comunicación en red y otras muchas responsabilidades propias de un sistemas operativo.

4.1.2 Capa Librerías & Android Runtime

La siguiente capa se corresponde con las **librerías** utilizadas por Android. Éstas han sido escritas utilizando C/C++ y proporcionan a Android la mayor parte de sus capacidades más características. Junto al núcleo basado en Linux, estas librerías constituyen el corazón de Android.

Entre las librerías más importantes de este nivel, se pueden mencionar las siguientes:

- La librería *libc* incluye todas las cabeceras y funciones según el estándar del lenguaje C. Todas las demás librerías se definen en este lenguaje.

- La librería *Surface Manager* es la encargada de componer los diferentes elementos de navegación de pantalla. Gestiona también las ventanas pertenecientes a las distintas aplicaciones activas en cada momento.

- *OpenGL/SL* y *SGL* representan las librerías gráficas y, por tanto, sustentan la capacidad gráfica de Android. *OpenGL/SL* maneja gráficos en 3D y permite utilizar, en caso de que esté disponible en el propio dispositivo móvil, el hardware encargado de proporcionar gráficos 3D. Por otro lado, *SGL* proporciona gráficos en 2D, por lo que será la librería más habitualmente utilizada por la mayoría de las aplicaciones. Una característica importante de la capacidad gráfica de Android es que es posible desarrollar aplicaciones que combinen gráficos en 3D y 2D.

- La librería *Media Libraries* proporciona todos los códecs necesarios para el contenido multimedia soportado en Android (vídeo, audio, imágenes estáticas y animadas, etc.).

- *FreeType*, permite trabajar de forma rápida y sencilla con distintos tipos de fuentes.

- La librería *SSL* posibilita la utilización de dicho protocolo para establecer comunicaciones seguras.

- A través de la librería *SQLite*, Android ofrece la creación y gestión de bases de datos relacionales, pudiendo transformar estructuras de datos en objetos fáciles de manejar por las aplicaciones.

- La librería *WebKit* proporciona un motor para las aplicaciones de tipo navegador, y forma el núcleo del actual navegador incluido por defecto en la plataforma Android.

Al mismo nivel que las librerías de Android se sitúa el **entorno de ejecución**. Éste lo constituyen las *Core Libraries*, que son librerías con multitud de clases de Java, y la máquina virtual Dalvik.

4.1.3 Capa armazón de Aplicaciones

Los dos últimos niveles de la arquitectura de Android están escritos enteramente en Java. El ***framework* de aplicaciones** representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o terceras compañías, o incluso las que el propio usuario cree, utilizan el mismo conjunto de API y el mismo *framework*, representado por este nivel.

Entre las API más importantes ubicadas aquí, se pueden encontrar las siguientes:

- *Activity Manager*, importante conjunto de API que gestiona el ciclo de vida de las aplicaciones en Android (del que se hablará más adelante).

- *Window Manager*, gestiona las ventanas de las aplicaciones y utiliza la librería ya vista *Surface Manager*.

- *Telephone Manager*, incluye todas las API vinculadas a las funcionalidades propias del teléfono (llamadas, mensajes, etc.).

- *Content Providers*, permite a cualquier aplicación compartir sus datos con las demás aplicaciones de Android. Por ejemplo, gracias a esta API la información de contactos, agenda, mensajes, etc. será accesible para otras aplicaciones.

- *View System*, proporciona un gran número de elementos para poder construir interfaces de usuario (GUI), como listas, mosaicos, botones, *check-boxes*, tamaño de ventanas, control de las interfaces mediante tacto o teclado, etc. Incluye también algunas vistas estándar para las funcionalidades más frecuentes.

- *Location Manager*, posibilita a las aplicaciones la obtención de información de localización y posicionamiento, y funcionar según ésta.

- *Notification Manager*, mediante el cual las aplicaciones, usando un mismo formato, comunican al usuario eventos que ocurran durante su ejecución: una llamada entrante, un mensaje recibido, conexión Wi-Fi disponible, ubicación en un punto determinado, etc. Si llevan asociada alguna acción, en Android denominada *Intent*, (por ejemplo, atender una llamada recibida) ésta se activa mediante un simple clic.

- *XMPP Service*, colección de API para utilizar este protocolo de intercambio de mensajes basado en XML.

4.1.4 Capa Aplicaciones

El último nivel del diseño arquitectónico de Android son las **aplicaciones**. Éste nivel incluye tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente, ya sean de terceras empresas o de su propio desarrollo.

Todas estas aplicaciones utilizan los servicios, las API y librerías de los niveles anteriores.

4.2 La máquina virtual Dalvik

En Android, todas las aplicaciones se programan en el lenguaje Java y se ejecutan mediante una máquina virtual de nombre Dalvik, específicamente diseñada para Android.

Esta máquina virtual ha sido optimizada y adaptada a las peculiaridades propias de los dispositivos móviles (menor capacidad de proceso, baja memoria, alimentación por batería, etc.) y trabaja con ficheros de extensión .dex (*Dalvik Executables*).

Dalvik no trabaja directamente con el *bytecode* de Java, sino que lo transforma en un código más eficiente que el original, pensado para procesadores pequeños.

Gracias a la herramienta “dx”, esta transformación es posible: los ficheros .class de Java se compilan en ficheros .dex, de forma que cada fichero .dex puede contener varias clases. Después, este resultado se comprime en un único archivo de extensión .apk (*Android Package*), que es el que se distribuirá en el dispositivo móvil.

Dalvik permite varias instancias simultáneas de la máquina virtual, y a diferencia de otras máquinas virtuales, está basada en registros y no en pila, lo que implica que las instrucciones son más reducidas y el número de accesos a memoria es menor. Así mismo, Dalvik no permite la compilación *Just-in-Time*.

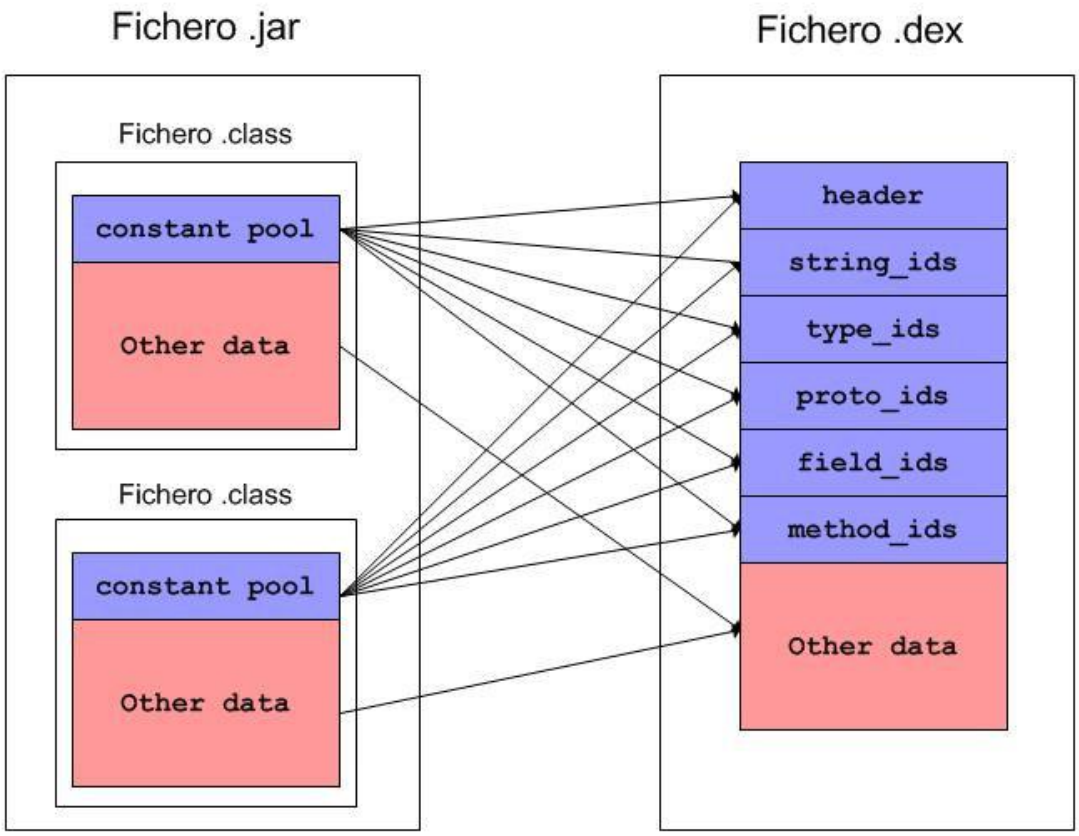


Figura 9. Formato de un fichero .dex. [9]

Según los responsables del proyecto, la utilización de esta máquina virtual responde a un deseo de mejorar y optimizar la ejecución de aplicaciones en dispositivos móviles, así como evitar la fragmentación de otras plataformas como Java ME.

A pesar de esta aclaración oficial, no ha pasado inadvertido que con esta maniobra Android ha esquivado tener que utilizar directamente Java ME, evitando así que los futuros desarrolladores de aplicaciones tengan que comprar ninguna licencia, ni tampoco que publicar el código fuente de sus productos.

El lenguaje Java utilizado en Android no sigue ningún JSR determinado, y Google se afana en recordar que Android no es tecnología Java, sino que simplemente utiliza este lenguaje en sus aplicaciones.

4.3 Política de eliminación de procesos

Cada aplicación Android se ejecuta en su propio proceso. Este proceso se crea cada vez que una aplicación necesita ejecutar parte de su código, y seguirá existiendo hasta que la aplicación finalice o hasta que el sistema necesite utilizar parte de sus recursos para otra aplicación considerada prioritaria.

Por ello, es importante saber cómo los componentes de una aplicación en Android (*Activity*, *Broadcast Intent Receiver*, *Service* y *Content Provider*) determinan e influyen en el ciclo de vida de la aplicación.

No usar los componentes correctamente a la hora de construir una aplicación puede significar que el sistema operativo la termine cuando en realidad está haciendo una tarea importante para el usuario.

Android construye una jerarquía donde evalúa la clase de componentes que están ejecutándose y el estado de los mismos. En orden de importancia, serían los siguientes:

1. **Procesos en primer plano**: aquellos necesarios para lo que el usuario está haciendo en ese momento. Un proceso se encuadra en esa categoría si cumple alguna de las siguientes condiciones:

    a. tiene un componente *Activity* ejecutándose, con la que el usuario está interactuando.

    b. tiene un componente *Broadcast Intent Receiver* ejecutándose.

    c. ha lanzado algún otro proceso que tiene un componente *Service* ejecutándose en el momento Idealmente, sólo debería haber algunos de estos procesos en el sistema, y su eliminación debería darse únicamente en casos extremos en los que la falta de recursos impide siguen ejecutándolos todos.

2. **Procesos visibles**: aquellos procesos que contienen un componente *Activity* visible en la pantalla, pero no con el foco de actividad en ese momento.

3. **Procesos de servicio**: aquellos procesos que tienen un componente *Service* y que están ejecutándose en *background*. Aunque no sean visibles directamente al usuario, desempeñan tareas sí percibidas por este.

4. **Procesos en segundo plano**: procesos con un componente *Activity*, que no son visibles al usuario. Estos procesos no tienen una importancia directa para el usuario en ese momento.

5. **Procesos vacíos**: procesos que ya no ejecutan ninguna actividad, pero que se mantienen en memoria para agilizar una posible nueva llamada por parte del usuario.

Según esta jerarquía, Android prioriza los procesos existentes en el sistema y decide cuáles han de ser eliminados, con el fin de liberar recursos y poder lanzar la aplicación requerida.

4.4 Gestión de la información

A diferencia de lo que ocurre en otros sistemas, en Android cualquier repositorio de datos, como por ejemplo archivos, son privados y únicamente accesibles por la aplicación a la que pertenecen. Sin embargo, esto no implica necesariamente que no puedan ser compartidos o accedidos por otras aplicaciones; Android facilita una serie de mecanismos que posibilitan en ciertas circunstancias el acceso a dicha información.

En los siguientes apartados se explican las distintas formas que tiene una aplicación para almacenar y recuperar información, así como los mecanismos a utilizar en caso de desear compartirla con las demás aplicaciones.

4.4.1 Preferencias de usuario

Las preferencias son todos aquellos valores asociados a una determinada aplicación y que permiten adaptar ésta a los gustos o necesidades del usuario. Solamente se puede acceder a las preferencias dentro de un mismo paquete; Android no permite compartir estos valores con otras aplicaciones.

Para poder compartir las preferencias entre todos los componentes que forman la aplicación, dentro del mismo paquete, debe asignarse un nombre al conjunto de valores que forman las preferencias de usuario. Después, éstas se pueden recuperar a lo largo de todo el paquete haciendo una llamada a `Context.getSharedPreferences()`. Si las preferencias solamente serán accedidas desde un componente Activity y no es necesario compartirlas con los demás componentes, es posible omitir el nombre asociado a las preferencias y recuperarlas simplemente llamando a `Activity.getPreferences()`.

4.4.2 Ficheros

Para leer o escribir ficheros, Android facilita los métodos `Context.openFileInput()` y `Context.openFileOutput()`, respectivamente. Estos ficheros deben ser locales a la aplicación en curso, es decir, al igual que con las preferencias, un fichero "estándar" de datos no puede ser compartido con otras aplicaciones.

Si el fichero es estático y puede ser adjuntado a la aplicación en el momento que se compila, éste puede ser añadido como un recurso más. Su ubicación correcta será en la carpeta de recursos, concretamente en "\res\raw\nombrefichero.extension". Esto permitirá que acompañe al paquete completo y que pueda ser leído directamente con el método `Resources.openRawResource (R.raw.nombrefichero)`.

4.4.3 Bases de datos

Android incluye una librería de SQLite que permite crear bases de datos relacionales, navegar entre las tablas, ejecutar sentencias en SQL y otras funcionalidades propias del sistema SQLite. La base de datos resultante puede ser accedida desde el código de la aplicación como si de un objeto más se tratara, gracias a las clases contenidas en el paquete `android.database.sqlite`.

Cualquier base de datos será accesible desde los demás componentes de la misma aplicación, pero no fuera de ésta.

4.4.4 Acceso por red

Como es lógico, Android permite la comunicación entre dispositivos a través de una infraestructura de red, siempre y cuando ésta esté disponible. Los paquetes *java.net* y *android.net* ofrecen multitud de clases y posibilidades en este sentido.

4.4.5 Content Provider

Hasta ahora, se han mencionado algunas formas para almacenar y recuperar información de forma siempre local y sin poder compartirla con otras aplicaciones del mismo dispositivo. Android ofrece un mecanismo que posibilita el acceso a información de forma compartida: usando un Content Provider o proveedor de contenidos. Ésta es la única forma habilitada para facilitar datos más allá del propio paquete de la aplicación.

Por defecto, Android incluye una serie de componentes Content Provider que permiten publicar todo tipo de datos básicos que pueden resultar útiles entre aplicaciones: información de los contactos, fotografías, imágenes, vídeos, mensajes de texto, audios, etc. Todos estos Content Provider ya definidos y listos para utilizar se pueden encontrar en el paquete *android.provider*. Además, Android ofrece la posibilidad de que el desarrollador pueda crear sus propios Content Provider.

Un Content Provider es un objeto de la clase `ContentProvider`, ubicada en el paquete *android.content*, y que almacena datos de un determinado tipo que pueden ser accedidos desde cualquier aplicación. Cada elemento Content Provider tiene asociado una URI única que lo representa y a través de la cual los otros componentes de una aplicación pueden acceder a él.

Por ejemplo, la cadena "content://contacts/people/" es una URI válida que da acceso a la información de contactos del dispositivo; la cadena "content://media/external/images" es la URI identificativa de otro Content Provider que da acceso a las imágenes de un dispositivo de almacenamiento externo (tarjeta SD, por ejemplo).

Conocida la URI, cualquier componente puede acceder al correspondiente Content Provider. En el siguiente ejemplo, mostrado en el Código 1, se accede a la información de contactos del dispositivo utilizando la URI que identifica el Content Provider de los contactos. La información que se quiere obtener es el ID del contacto, su

nombre y su teléfono. Para hacer una consulta a través de una URI, existen varios métodos disponibles: uno de ellos es `managedQuery()`, de la clase `Activity`. Este método requiere como parámetros la URI a la que se consulta, los campos a seleccionar, y otros valores como las cláusulas WHERE u ORDER BY, como si de una consulta SQL se tratara. El resultado consiste en un conjunto de filas que se puede recorrer a través del objeto `Cursor`, clase ubicada en el paquete *android.database*.

```
// Columnas a consultar
String[] projection = new String[] {
      Contacts.People.NAME,
      Contacts.People.NUMBER,
      Contacts.People._ID
};

// Establecer URI para acceder a los contactos
Uri contacts = "content://contacts/people/";

// Lanzar consulta
Cursor cursor = managedQuery( contacts,
      projection,
      null,
      null,
      Contacts.People.NAME + " ASC");
```

Código 1. Ejemplo de consulta a un Content Provider.

Los mecanismos de actualización e inserción de datos son muy similares a la consulta anteriormente vista. Sin embargo, el mayor potencial de los Content Provider estriba en que el desarrollador cree los suyos propios adaptados a las necesidades de su aplicación.

Para crear un Content Provider, se deben seguir los siguientes pasos:

1. Crear una clase que extienda a `ContentProvider`.

2. Definir una constante denominada `CONTENT_URI` donde quede recogida la URI que identificará a este nuevo Content Provider. Esta cadena **siempre** ha de comenzar por el prefijo "content://" seguida de la jerarquía de nombres que se desee establecer. Por ejemplo "content://miaplicacion.misdatos".

3. Establecer el sistema de almacenamiento deseado. Lo habitual es establecer una base de datos en SQLite, pero se puede utilizar cualquier mecanismo de almacenamiento.

4. Definir también como constantes el nombre de las columnas de datos que pueden ser recuperadas, de forma que el futuro usuario de este Content Provider puede conocerlas.

5. Implementar los métodos básicos de manipulación de datos. Estos son:
   - `query()`, debe devolver un objeto `Cursor`.

- `insert()`
- `update()`
- `delete()`
- `getType()`

6. En el fichero "AndroidManifest.xml" de la aplicación debe añadirse, dentro del elemento `<application>`, la siguiente etiqueta:

```
<application>
      <provider android:name="nombre_del_content_provider"
      android:authorities="uri_asignada"
      />
</application>
```

Código 2. Declaración en el manifiesto de un componente Content Provider.

4.5 Seguridad en Android

Android es una plataforma móvil moderna que fue diseñada para ser de carácter libre. Las aplicaciones de Android hacen uso de hardware y software avanzados, así como de datos tanto locales como servidos, expuestos a través de la plataforma para ofrecer innovación y calidad a los consumidores. Para proteger esa calidad, la plataforma ofrece un entorno de aplicación que garantice la seguridad de los usuarios, datos, aplicaciones, el dispositivo, y la red.

Proporcionar una buena seguridad a una plataforma de código abierto como Android, requiere de una robusta arquitectura de seguridad y de rigurosos programas de seguridad. Android fue diseñado con varias capas de seguridad que proporcionan la flexibilidad necesaria para una plataforma de carácter abierto como esta, mientras que proporcionan la protección para todos los usuarios de la plataforma.

Android fue diseñado teniendo en mente a los desarrolladores. Los controles de seguridad fueron diseñados para reducir la carga de la misma sobre los desarrolladores. La seguridad inteligente hace que los desarrolladores puedan trabajar y confiar en los flexibles controles de seguridad proporcionados en la plataforma. Así mismo, los desarrolladores menos familiarizados con la seguridad estarán protegidos de la falta de esta última.

Android también fue diseñado teniendo en mente a los usuarios de los dispositivos bajo la plataforma. Dichos usuarios disponen de visibilidad en todo momento de cómo funcionan las aplicaciones y el control que tienen sobre las mismas. Este diseño incluye la expectativa de que los atacantes intenten llevar a cabo los ataques más comunes, tales como ataques de ingeniería social cuyo cometido es el de convencer a los usuarios de los dispositivos de instalar malware en los mismos, y los ataques a aplicaciones de terceros. Android fue diseñado tanto para reducir la probabilidad de sufrir dichos ataques como para limitar en gran medida el impacto del ataque en caso de que se haya realizado.

A continuación en este proyecto vamos a describir los objetivos del programa de seguridad de Android y los fundamentos de la arquitectura de seguridad que emplea. En este punto nos centraremos en las características de seguridad del núcleo de la plataforma Android, no abordando los problemas de seguridad únicos para aplicaciones específicas,

tales como los relacionados con el navegador o una aplicación SMS. Las mejores prácticas recomendadas en la construcción de dispositivos Android, el despliegue de dichos dispositivos, o el desarrollo de aplicaciones para Android no es el objetivo de este punto y se proporcionarán en puntos y capítulos posteriores en este mismo proyecto.

4.5.1 Background

Android es una plataforma de código abierto y entorno de aplicaciones para dispositivos móviles. Los principales bloques de construcción de plataforma Android son:

- **Dispositivos de hardware**: Android se ejecuta en una amplia gama de configuraciones de hardware, incluidos los teléfonos inteligentes (o "Smartphones"), "tablets" y "set-top-boxes". Android es el procesador agnóstico, pero se aprovecha de algunas capacidades de seguridad de hardware específicos, tales como ARM v6 eXecute-Never.

- **Sistema operativo Android**: El sistema operativo central se construye en la parte superior del núcleo Linux. Todos los recursos del dispositivo, como funciones de la cámara, los datos GPS, funciones de Bluetooth, funciones de telefonía, conexiones de red, etc… se acceden a través de dicho sistema operativo.

- **Android Application Runtime**: Las aplicaciones para Android son a menudo desarrolladas en el lenguaje de programación Java y se ejecutan en la máquina virtual Dalvik. Sin embargo, muchas aplicaciones, incluyendo los servicios básicos de Android y algunas aplicaciones, son de carácter nativo o incluyen bibliotecas nativas. Ambos, Dalvik y aplicaciones nativas, se ejecutan dentro del entorno de seguridad, contenidas en el "sandbox" para aplicaciones. Cada aplicacion tiene un espacio del sistema de archivos específico en el que puede almacenar datos privados, incluyendo bases de datos y los archivos RAW.

Las aplicaciones de Android extienden del núcleo del sistema operativo Android. Hay dos origenes principales de aplicaciones:

- **Aplicaciones pre-instaladas**: Android incluye un conjunto de aplicaciones pre-instaladas que incluyen las funcionalidades como teléfono, correo electrónico, calendario, navegador web, y un administrador de contactos. Funcionan como aplicaciones de usuario y son clave para proporcionar funcionalidades del dispositivo, se pueden acceder desde otras aplicaciones. Estas aplicaciones pre-instaladas pueden ser parte de la plataforma de código abierto Android, o pueden ser desarrolladas para un dispositivo específico.

- **Aplicaciones instaladas por el usuario**: Android ofrece un entorno de desarrollo abierto para cualquier aplicación de terceros. El Google Play ofrece a los usuarios cientos de miles de aplicaciones.

Google ofrece un conjunto de servicios basados en la "nube" que están disponibles para cualquier dispositivo compatible con Android. Los servicios principales son:

- **Google Play**: Es un conjunto de servicios que permiten a los usuarios buscar, instalar y comprar aplicaciones desde su dispositivo Android o desde la web. Hace fácil para los desarrolladores llegar a los usuarios de Android y clientes potenciales. También proporciona revisiones comunitarias, verificación de las licencias de aplicaciones, y otros servicios de seguridad.

- **Actualizaciones Android**: El servicio de actualización de Android ofrece nuevas capacidades y actualizaciones de seguridad para dispositivos Android, incluyendo actualizaciones a través de la web o por el aire (vía OTA).

- **Servicios de aplicaciones**: marcos que permiten a las aplicaciones Android utilizar las capacidades de la nube, tales como la copia de seguridad de los datos y la configuración de dichas aplicaciones.

Estos servicios aunque no forman parte del proyecto Android de código abierto como tal, son importantes para la seguridad de la mayoría de los dispositivos Android, por lo que en puntos posteriores de este mismo proyecto ahondaremos más en dichos temas.

4.5.1.1 Visión general del programa de seguridad para Android

Desde la fase temprana del desarrollo, el equipo principal de Android reconoció que un modelo de seguridad robusto era necesario para permitir un ecosistema vigoroso de las aplicaciones y dispositivos construidos en y alrededor de la plataforma Android y con el apoyo de servicios en la nube. Como resultado de ello, a través de su ciclo de vida completo de desarrollo, Android ha sido sometido a un programa de seguridad profesional. El equipo de Android ha tenido la oportunidad de observar cómo otros móviles, de escritorio y plataformas de servidor de prevenir y reaccionar a los problemas de seguridad y construyó un programa de seguridad para abordar los puntos débiles observados en otras ofertas.

4.5.1.2 Arquitectura de seguridad de la plataforma Android

Android pretende ser el sistema operativo más seguro y útil para plataformas móviles por proponer controles de seguridad de sistemas operativos tradicionales para:

- Proteger los datos del usuario.

- Proteger los recursos del sistema (incluyendo la conexión de red).

- Proporcionar el aislamiento de aplicaciones.

Para lograr estos objetivos, Android proporciona las siguientes características clave de seguridad:

- Una seguridad robusta a nivel de sistema operativo a través del núcleo de Linux.

- "Sandbox" de aplicación obligatorio para todas las aplicaciones.

- Seguridad en la comunicación entre procesos.
- Firma de aplicaciones.
- Permisos de aplicación y usuarios.

En los puntos siguientes describiremos estas y otras características de seguridad de la plataforma Android.

Figura 10 se resumen los componentes de seguridad y las consideraciones de los distintos niveles de la pila de software Android. Cada componente se asume que los siguientes componentes estén bien sujetos. Con la excepción de una pequeña cantidad de código del sistema operativo Android corriendo como root, todo el código por encima del núcleo de Linux se ve limitado por el recinto de seguridad de aplicaciones.

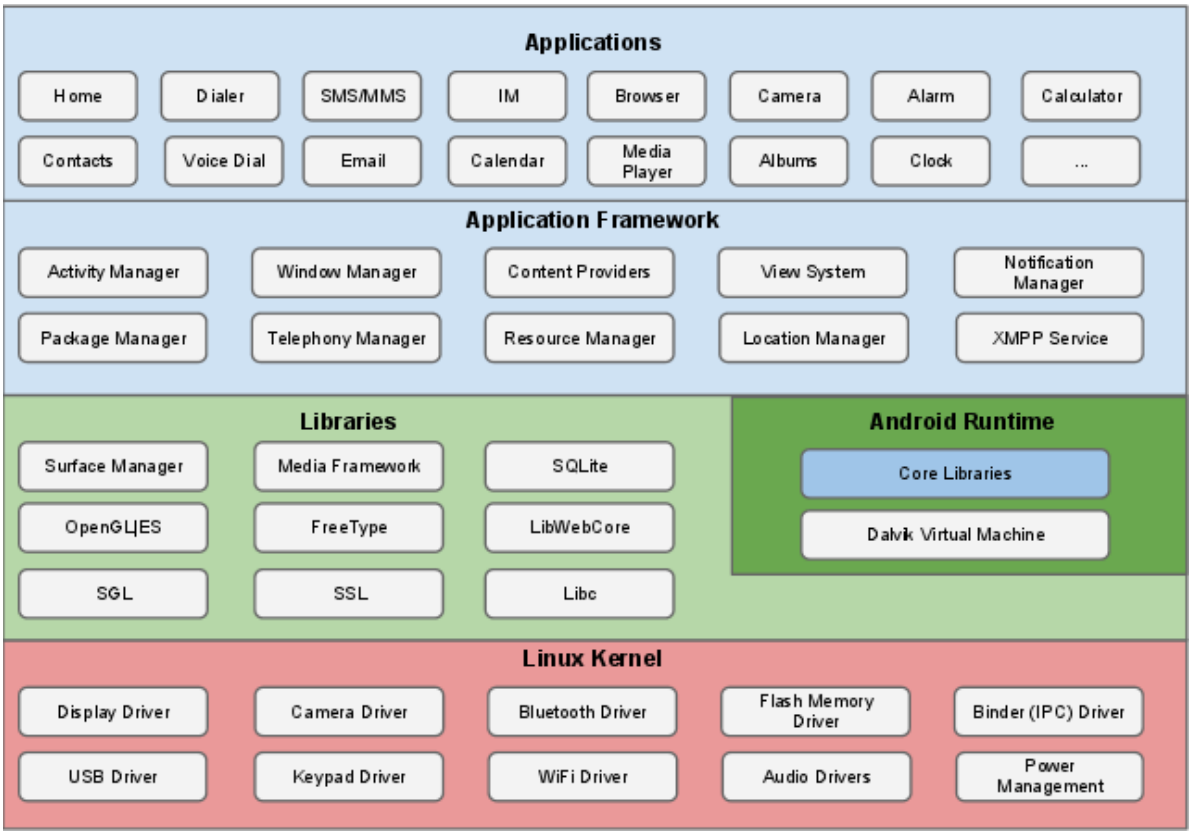


Figura 10. Pila de software Android.

4.5.2 Seguridad a nivel de sistema operativo y núcleo

A nivel de sistema operativo, la plataforma Android ofrece la seguridad del kernel Linux, así como una comunicación segura entre procesos (IPC) permitiendo la posibilidad de comunicación segura entre aplicaciones que se ejecutan en diferentes procesos. Estas características de seguridad a nivel de sistema operativo aseguran que incluso el código nativo del sistema se ve limitado por el recinto de seguridad de aplicaciones. Ya sea este código el resultado del comportamiento de las aplicaciones ya incluidas, o la explotación de una vulnerabilidad de la aplicación, el sistema impedirá a la

aplicación atacante hacer daño a otras aplicaciones, el sistema Android, o el propio dispositivo.

4.5.2.1 La seguridad del Kernel Linux

La base de la plataforma Android es el núcleo de Linux. El kernel de Linux ha sido de uso generalizado durante años, y se utiliza en millones de entornos de seguridad. A lo largo de su historia ha sido constantemente revisado, atacado, y mejorado por miles de desarrolladores, Linux se ha convertido en un núcleo estable y seguro ganándose la confianza de muchas empresas y profesionales de la seguridad.

Como base para un entorno de computación móvil, el kernel de Linux ofrece a Android varias características de seguridad clave, incluyendo:

- Un modelo basado en permisos de usuario.
- Aislamiento de procesos.
- Mecanismo extensible para seguridad IPC.
- La capacidad de eliminar partes innecesarias y potencialmente inseguras del kernel.

Como sistema operativo multiusuario, un objetivo fundamental de seguridad del kernel de Linux es aislar los recursos de un usuario de otros. La filosofía de seguridad de Linux es la protección de recursos de los usuarios entre sí. Por lo tanto, Linux:

- Previene al usuario A de la lectura de los archivos del usuario B.
- Asegura que el usuario A no consuma la memoria del usuario B.
- Asegura que el usuario A no consuma los recursos de la CPU del usuario B.
- Asegura que el usuario A no pueda usar los dispositivos del usuario B (por ejemplo telefonía, GPS, bluetooth).

4.5.2.2 El recinto de seguridad de aplicaciones

La plataforma Android aprovecha las ventajas de protección de usuarios que proporciona Linux, basándose en dicha protección como medio para identificar y aislar los recursos de las aplicaciones. El sistema Android asigna un identificador único de usuario (UID) a cada aplicación de Android y las ejecuta con ese identificador de usuario en procesos separados. Este enfoque es diferente de otros sistemas operativos (incluyendo la tradicional configuración de Linux), donde múltiples aplicaciones se ejecutan con los permisos del mismo usuario.

Este mecanismo establece un recinto de seguridad para las aplicaciones a nivel de kernel. El núcleo refuerza la seguridad entre las aplicaciones y el sistema operativo a nivel de proceso, a través de los estándares que proporciona Linux, como hemos

comentado, son el identificador de usuario e identificador de grupo que se asignan a las aplicaciones.

De forma predeterminada, las aplicaciones no pueden interactuar entre sí y tienen un acceso limitado al sistema operativo. Si una aplicación A intenta hacer algo malicioso, como leer datos de una aplicación B o realizar una llamada telefónica sin permiso (que es una aplicación independiente), el sistema operativo nos protege en contra de este tipo de acciones al no tener posesión la aplicación A de los privilegios de usuario correspondientes. El recinto de seguridad es sencillo, auditable, y está basado en décadas de experiencia en separación de procesos y permisos de archivos mediante el sistema de usuarios de estilo UNIX anteriormente mencionado.

Desde que el mecanismo de recinto de seguridad de aplicación se encuentra en el núcleo, este modelo de seguridad se extiende al código nativo y aplicaciones integradas del sistema operativo. Todo el software por encima del núcleo en la Figura 10, incluidas las bibliotecas del sistema operativo, marcos de aplicación, tiempo de ejecución de aplicaciones, y todas las aplicaciones, se ejecutan dentro del recinto de seguridad de aplicaciones. En algunas plataformas, los desarrolladores se ven obligados a un marco de desarrollo en específico, un conjunto de APIs, o un lenguaje con el fin de reforzar la seguridad. En Android, no hay restricciones de cómo una aplicación debe ser escrita o desarrollada para reforzar la seguridad, en este sentido, el código nativo es tan seguro como código interpretado en Android.

En algunos sistemas operativos, los errores de corrupción de memoria en general, llevan completamente a comprometer la seguridad del dispositivo. Este no es el caso de Android, debido a que todas las aplicaciones y los recursos se encuentran en el recinto a nivel del sistema operativo. Un error de corrupción de memoria sólo permitirá la ejecución de código arbitrario en el contexto de esa aplicación en particular, con los permisos establecidos por el sistema operativo y no afectará a otras.

Al igual que todos los elementos de seguridad, el recinto de seguridad de aplicaciones no es irrompible. Sin embargo, al salir del recinto de seguridad de aplicaciones en un dispositivo configurado correctamente, se puede poner en peligro la seguridad del kernel de Linux.

4.5.2.3 Partición del sistema y el modo seguro

La partición del sistema contiene el kernel de Android, así como las bibliotecas del sistema operativo, “runtime” de aplicación, marco de aplicación, y las aplicaciones. Esta partición se encuentra en modo sólo lectura. Cuando un usuario inicia el dispositivo en modo seguro, sólo las aplicaciones básicas de Android están disponibles. Esto asegura que el usuario pueda arrancar su teléfono en un ambiente que esté libre de software de terceros o malicioso.

4.5.2.4 Permisos del sistema de archivos

En un entorno de tipo UNIX, los permisos del sistema de archivos aseguran que un usuario no puede modificar o leer archivos de otro usuario. En el caso de Android, cada aplicación se ejecuta como su propio usuario. A menos que el desarrollador exponga

explícitamente los archivos a otras aplicaciones, los archivos creados por una aplicación no puede ser leído o modificado por otra aplicación.

4.5.2.5 Cifrado del sistema de archivos

Android 3.0 y versiones posteriores proporcionan un cifrado del sistema de ficheros completo, por lo que todos los datos se pueden cifrar en el kernel mediante la implementación dmcrypt de AES128 con CBC y ESIVV: SHA256. La clave de cifrado está protegido por AES128 utilizando una clave derivada de la contraseña del usuario, evitando el acceso no autorizado a los datos almacenados sin la contraseña del dispositivo del usuario. Para proporcionar resistencia contra los ataques de adivinar la contraseña sistemática (por ejemplo, “rainbow tables” o fuerza bruta), la contraseña se combina con algoritmos “salt” y “hash” aleatoriamente con SHA1 en varias ocasiones con el uso del algoritmo estándar PBKDF2 antes de ser utilizado para descifrar la clave del sistema de archivos. Para proporcionar resistencia contra los ataques de adivinar contraseñas de diccionario, Android proporciona reglas de complejidad de contraseñas que se pueden configurar por el administrador del dispositivo y ejecutadas por el sistema operativo. El cifrado del sistema de archivos requiere el uso de una contraseña de usuario, los patrones basados en el bloqueo de pantalla no son compatibles.

Más detalles sobre la aplicación del cifrado del sistema de archivos los trataremos en capítulos posteriores de este mismo proyecto para ahondar más en estos temas relacionados con seguridad del sistema operativo.

4.5.2.6 Cómo funciona el sistema de cifrado Android

El cifrado de disco en Android se basa en “dm-crypt” (provee cifrado transparente de dispositivos de bloque utilizando la nueva cryptoapi de Linux 2.6), que es una característica del núcleo que funciona en la capa de bloque del dispositivo. Por lo tanto, no se puede utilizar con “YAFFS”, ya que esto trata directamente con chips flash “NAND”, pero sí funciona con “emmc” y dispositivos flash similares que se presentan en el núcleo como dispositivos de bloque. El sistema de archivos actual preferido para usar en estos dispositivos es “ext4”, sin embargo, esto es independiente de si se utiliza el cifrado o no.

Si bien el trabajo de cifrado es una característica estándar del kernel linux, habilitarlo en un dispositivo Android resultó un tanto difícil. El sistema Android trata de evitar la incorporación de componentes GPL, así que usar el comando “cryptsetup” o “libdevmapper” no eran opciones disponibles. Así que realizar la correspondiente llamada “ioctl” (es una llamada de sistema en Unix que permite a una aplicación controlar o comunicarse con un driver de dispositivo, fuera de los usuales read/write de datos) al núcleo era la mejor opción. El demonio de volumen de Android (“vold”) ya hacía esto para favorecer el movimiento de aplicaciones a la tarjeta SD, así que se optó por aprovechar este trabajo para el cifrado de disco completo. El cifrado actual utilizado para el sistema de archivos de la primera versión es el 128 AES con CBC y ESSIV: SHA256. La clave maestra se cifra con 128 bit AES a través de llamadas a la librería OpenSSL.

Una vez que se decidió apostar por “vold”, se hizo evidente que la invocación de las características de cifrado de este, es igual que la invocación de otros comandos de vold, mediante la adición de un nuevo módulo para vold (llamado “cryptfs”) y la implementación de varios comandos. Los comandos son “checkpw”, “restart”, “enablecrypto”, “changepw” y “cryptocomplete”. Que describiremos con más detalle a continuación.

El gran problema era cómo obtener la contraseña del usuario en el arranque. El plan inicial era implementar una mínima interfaz de usuario que pueda ser invocada por “init” en el disco de memoria inicial, y entonces “init” descifra y monta “/data”. Sin embargo, el ingeniero de la interfaz de usuario, dijo que era mucho trabajo, y sugirió que se comunicase con “init” en el arranque para decirle al marco que apareciera una pantalla de ingreso de contraseña, obtener dicha contraseña, y a continuación, cerrarla e invocar el marco de arranque real. Se decidió seguir este método, y esto llevó después a una serie de decisiones que describiremos a continuación. En particular, init envía una propiedad para indicar al marco que debe entrar en el modo especial de ingreso de contraseña, y que envió las bases para la comunicación entre “vold”, “init” y el marco usando las propiedades. Los detalles los describiremos a continuación.

Por último, hubo problemas en torno a matar y reiniciar los diversos servicios para que “/data” pudiera ser desmontado y vuelto a montar. Invocar el marco temporal para obtener la contraseña de usuario requiere que un “tmpfs” como el sistema de archivos /data esté montado, de lo contrario el marco no se ejecutará. Sin embargo, para desmontar el sistema de archivos “tmpfs” “/data” y que el verdadero descifrado del sistema de archivos “/data” pueda ser montado significa que todos los procesos que tenían archivos abiertos en la “tmpfs /data” tuvieran que ser matados y reiniciados en el sistema de ficheros real “/data”. Esta magia se logra al exigir que todos los servicios formen parte de uno de los tres grupos siguientes: “core”, “main” y “late_start”.

- **Core**: estos servicios no se cierran después de inicializarlos.
- **Main**: estos servicios que se cierran y se reinician después de que la contraseña del disco sea introducida.
- **Late_start**: estos servicios no se inician hasta después de que “/data” haya sido descifrado y montado.

La magia para provocar este tipo de acciones reside en el envío de la propiedad “vold.decrypt” con varias cadenas mágicas diferentes, que describiremos a continuación. Además, un nuevo comando init "class_reset" fue inventado para detener un servicio, y para permitir que fuera reanudado el comando "class_start". Si el comando "class_stop" fuera utilizado en vez del nuevo comando "class_reset" la bandera “SVC_DISABLED” sería añadida en el estado de cualquier servicio que estuviera detenido, lo que significa que no se podría iniciar cuando el comando “class_start” hubiera sido usado en esta clase.

4.5.2.7 Protección por contraseña

Android se puede configurar para validar contraseñas introducidas por el usuario antes de proporcionar el acceso de este al dispositivo. Además de prevenir el uso no autorizado del dispositivo, la contraseña protege la clave para el cifrado de sistema de ficheros completo.

El uso de una contraseña y/o reglas de complejidad de contraseñas puede ser requerido por el administrador del dispositivo.

4.5.2.8 Administración de dispositivos

Desde la versión del sistema operativo Android 2.2 y posteriores, se proporciona un API para la administración de dispositivos Android, que proporciona funciones de administración de dispositivos a nivel del sistema. Por ejemplo, la aplicación integrada de correo electrónico utiliza la API de Android para mejorar el soporte “Exchange”. A través de la aplicación de correo electrónico, los administradores de Exchange pueden aplicar políticas de contraseñas (incluyendo contraseñas alfanuméricas o PIN numérico) a través de los dispositivos. Los administradores también pueden borrar de forma remota (es decir, restaurar los valores predeterminados de fábrica) en caso de pérdida o robo del teléfono móvil.

Además de su uso en las aplicaciones incluidas o integradas en el sistema Android, estas API están disponibles para proveedores externos de soluciones de gestión de dispositivos. Los detalles de la API se ofrecen a continuación en este mismo capítulo.

4.5.2.8.1 Administración de dispositivos API general

Estos son algunos ejemplos de los tipos de aplicaciones que podrían usar la API de administración de dispositivos:

- Clientes de correo electrónico.
- Aplicaciones de seguridad que hacen borrados remotos.
- Servicios de gestión de dispositivos y aplicaciones.

4.5.2.8.2 ¿Cómo funciona?

Se utiliza la API de Administración de Dispositivos para desarrollar aplicaciones de administración que los usuarios instalen en sus dispositivos. La aplicación de dicha API en el desarrollo refuerza las políticas de seguridad elegidas. Así es como funciona:

- Un administrador del sistema, desarrolla una aplicación de administración que hace cumplir las políticas de dispositivos remotos/locales de seguridad. Estas políticas pueden ser codificadas de forma estricta en la aplicación, o esta última podría dinámicamente alcanzar dichas políticas mediante un servidor externo.

- La aplicación se instala en los dispositivos de los usuarios (Android actualmente no tiene una solución automatizada de aprovisionamiento). Algunas de las formas en que un administrador del sistema puede distribuir la aplicación a los usuarios son las siguientes:
    - “Google Play”.
    - Habilitando instalaciones ajenas a “Google Play”.
    - La distribución de la aplicación a través de otros medios, tales como correo electrónico, sitios web o Market´s de terceros.
- El sistema solicita al usuario que permita la instalación en el dispositivo de la aplicación de administración. Cómo y cuándo sucede depende de cómo la aplicación esté implementada.
- Una vez que los usuarios permiten la aplicación de administración de dispositivos, están sujetos a sus políticas. Cumplir con las políticas típicamente confiere beneficios, tales como el acceso a los sistemas y datos sensibles.

Si los usuarios no dan permiso a la aplicación de administración de dispositivos, esta permanecerá en el dispositivo, pero en estado inactivo. Los usuarios no estarán sujetos a sus políticas, y por el contrario no obtendrán ninguno de los beneficios de la aplicación, por ejemplo, no podrán ser capaces de sincronizar los datos.

Si un usuario no cumple con dichas políticas, por ejemplo, un usuario establece una contraseña que viola las directrices, dependerá de la aplicación decidir cómo manejar esta situación, normalmente esto se traducirá en que el usuario no será capaz de sincronizar los datos por ejemplo.

Si un dispositivo intenta conectarse a un servidor que requiere de políticas no compatibles con las del API de administración de dispositivos, la conexión no será permitida. La API de administración de dispositivos no permite actualmente aprovisionamiento parcial, en otras palabras, si un dispositivo no es compatible con todas y cada una de las políticas establecidas, no hay manera de permitir que el dispositivo se conecte.

Si un dispositivo contiene varias aplicaciones de administración activas, la política más estricta será la que se haga cumplir. No hay forma de escoger una aplicación de administración en particular.

Para desinstalar una aplicación de administración existente en el dispositivo, los usuarios deben primero cancelar el registro de la aplicación como administradores.

4.5.2.8.3 Políticas

En un entorno empresarial, a menudo se da el caso de que los dispositivos empleados deben cumplir con un estricto conjunto de políticas que rigen su uso. La API

de administración de dispositivos es compatible con las políticas mencionadas en la siguiente tabla. Debemos tener en cuenta que la API de administración de dispositivos actualmente sólo es compatible con las contraseñas de bloqueo de pantalla:

Política	**Descripción**
Contraseña habilitada	Requiere que los dispositivos soliciten PIN o contraseñas.
Longitud mínima de contraseña	Establecer el número mínimo necesario de caracteres para contraseñas. Por ejemplo, puede requerir que el PIN o contraseña tengan al menos seis caracteres.
Contraseña alfanumérica requerida	Requiere que las contraseñas tienen una combinación de letras y números. Se pueden incluir caracteres simbólicos.
Contraseña compleja requerida	Requiere que las contraseñas deben contener al menos una letra, un dígito numérico, y un símbolo especial. Introducido en la versión 3.0 del sistema operativo Android.
Mínimo de caracteres alfabéticos requeridos en la contraseña	El número mínimo de caracteres alfabéticos necesarios en la contraseña para todos los administradores o uno en particular. Introducido en la versión 3.0 del sistema operativo Android.
Mínimo de caracteres alfabéticos en minúscula requeridos en la contraseña	El número mínimo de caracteres alfabéticos en minúscula requeridos en la contraseña para todos los administradores o uno en particular. Introducido en la versión 3.0 del sistema operativo Android.
Mínimo de caracteres no alfabéticos requeridos en la contraseña	El número mínimo de caracteres no alfabéticos requeridos en la contraseña para todos los administradores o uno en particular. Introducido en la versión 3.0 del sistema operativo Android.
Mínimo de dígitos numéricos requeridos en la contraseña	El número mínimo de dígitos numéricos requeridos en la contraseña para todos los administradores o uno en particular. Introducido en la versión 3.0 del sistema operativo Android.
Mínimo de caracteres simbólicos requeridos en la contraseña	El número mínimo de caracteres simbólicos requeridos en la contraseña para todos los administradores o uno en particular. Introducido en la versión 3.0 del sistema operativo Android.
Mínimo de caracteres alfabéticos en mayúscula	El número mínimo de caracteres alfabéticos en mayúscula necesarios en la contraseña para todos los administradores o uno en particular. Introducido en la versión 3.0 del sistema

requeridos en la contraseña	operativo Android.
Tiempo de caducidad de la contraseña	Cuándo caducará la contraseña, expresado en milisegundos desde el momento en que el administrador del dispositivo establece el tiempo de caducidad. Introducido en la versión 3.0 del sistema operativo Android.
Historial de contraseñas	Esta política evita que los usuarios reutilicen las últimas *n* contraseñas. Esta política se suele utilizar junto con “setPasswordExpirationTimeout()” , lo que obliga a los usuarios a actualizar sus contraseñas después de que transcurra un período de tiempo especifico. Introducido en la versión 3.0 del sistema operativo Android.
Máximo de intentos fallidos de contraseña	Especifica cuántas veces el usuario puede introducir una contraseña incorrecta antes de que el dispositivo borre sus datos. La API de administración de dispositivos permite a los administradores reiniciar de forma remota el dispositivo a los valores predeterminados de fábrica. Esto protege los datos en caso de que el dispositivo sea robado o extraviado.
Tiempo máximo de inactividad para bloqueo	Establece el tiempo transcurrido desde que el último usuario toca la pantalla o pulsa un botón antes de que el dispositivo realice el bloqueo de pantalla. Cuando esto ocurre, los usuarios deben introducir su PIN o contraseña de nuevo antes de poder utilizar sus dispositivos y acceder a los datos. El valor puede variar entre 1 y 60 minutos.
Cifrado de almacenamiento requerido	Especifica que el área de almacenamiento debe ser cifrada si el dispositivo lo soporta. Introducido en la versión 3.0 del sistema operativo Android.
Desactivar la cámara	Especifica que la cámara debe estar deshabilitada. Hay que tener en cuenta que esto no tiene por qué ser de forma permanente. La cámara puede ser activada/desactivada de forma dinámica en función del contexto, el tiempo, y así sucesivamente. Introducido en la versión 4.0 del sistema operativo Android.

Tabla 2: Políticas con el apoyo de la API de administración de dispositivos.

Además de apoyar las políticas mencionadas en la tabla anterior, la API de administración de dispositivos nos permite hacer lo siguiente:

- Preguntar al usuario para establecer una nueva contraseña.
- Bloqueo del dispositivo de inmediato.

- Limpiar los datos del dispositivo (es decir, restaurar el dispositivo a sus valores de fábrica predeterminados).

4.5.2.9 Mejoras en la seguridad de la gestión de memoria

Android incluye muchas características que hacen que los problemas más comunes de seguridad resulten más difíciles de explotar. El SDK de Android, los compiladores y el sistema operativo usan herramientas que hacen más difícil explotar los problemas de corrupción de memoria, incluyendo:

- Disposición aleatoria del espacio de direcciones (ASLR) ubica al azar las claves en memoria.
- Hardware basado en la tecnología “No eXecute” (NX) para evitar la ejecución de código en la pila de memoria.
- “ProPolice” para evitar el desbordamiento del buffer de la pila.
- “safe_iop” para reducir el desbordamientos de enteros.
- Ampliaciones de “OpenBSD dlmalloc” para evitar vulnerabilidades “doble free ()” y para prevenir los ataques al fragmento de consolidación. Estos ataques son una forma común de explotar daños en la pila.
- “OpenBSD calloc” para evitar el desbordamientos de enteros durante la asignación de memoria.
- “Linux mmap_min_addr ()” para mitigar la referente escalada de privilegios de puntero nulo.

4.5.2.10 Root de dispositivos

Por defecto, en Android sólo el kernel y un pequeño subconjunto de las aplicaciones principales se ejecutan con permisos de “root”. Android no impide que un usuario o una aplicación con permisos de “root” modifiquen el sistema operativo, el núcleo, o cualquier otra aplicación. En general, el permiso “root” tiene acceso completo a todas las aplicaciones y todos los datos de las mismas. Los usuarios que cambian los permisos de un dispositivo Android para permitir el acceso “root” a las aplicaciones, aumentan el riesgo de aplicaciones maliciosas en la seguridad y posibles defectos.

La capacidad de modificar un dispositivo Android de su propiedad es importante para los desarrolladores que trabajan con la plataforma Android. En muchos dispositivos Android los usuarios tienen la capacidad para desbloquear el “bootloader” con el fin de permitir la instalación de un sistema operativo alternativo. Estos sistemas operativos alternativos pueden permitir al propietario tener acceso de “root” para fines de depuración de las aplicaciones y los componentes del sistema, o para acceder a funciones no presentes en las aplicaciones bajo la API Android.

En algunos dispositivos, una persona con el control físico de un dispositivo y un cable USB es capaz de instalar un nuevo sistema operativo que proporciona privilegios de “root” al usuario. Para proteger los datos de usuario existentes de acuerdo al mecanismo de desbloqueo del gestor de arranque, se requiere que dicho gestor borre todos los datos de usuario ya existentes como parte del procedimiento de desbloqueo. El acceso “root” a través de la explotación de un fallo del núcleo o agujero de seguridad puede pasar por alto esta protección.

El cifrado de datos con una clave almacenada en el dispositivo no protege los datos de aplicación de los usuarios que poseen permisos “root”. Las aplicaciones pueden añadir una capa de protección de datos extra mediante el cifrado con una clave almacenada fuera del dispositivo, como por ejemplo, en un servidor o una contraseña de usuario. Esta propuesta puede proporcionar una protección temporal mientras que la clave no esté presente, pero en algún momento la clave debe ser proporcionada a la aplicación y entonces se convierte en accesible a los usuarios con permisos “root”.

El enfoque más sólido para la protección de datos de los usuarios con permisos “root” es a través del uso de soluciones hardware. Los fabricantes de dispositivos pueden optar por implementar soluciones hardware que limitan el acceso a determinados tipos de contenidos tales como “DRM” para la reproducción de vídeo, o el almacenamiento relacionado con “NFC” de confianza para “Google wallet”.

En el caso de que un dispositivo fuera extraviado o robado, el cifrado del sistema de ficheros completo en los dispositivos Android utiliza la contraseña del dispositivo para proteger la clave de cifrado, por lo que modificar el gestor de arranque o el sistema operativo no es suficiente para acceder a los datos del usuario sin la contraseña del dispositivo del usuario.

4.5.3 Seguridad en las aplicaciones Android

4.5.3.1 Elementos de las aplicaciones

Android es una plataforma de código abierto y entorno de aplicaciones para dispositivos móviles. El núcleo del sistema operativo se basa en el kernel de Linux. Las aplicaciones de Android son a menudo desarrolladas en el lenguaje de programación Java y se ejecutan en la máquina virtual de Dalvik. Sin embargo, las aplicaciones también pueden ser desarrolladas en código nativo. Las aplicaciones se instalan desde un único archivo con la extensión “.apk”.

Los principales bloques de construcción para aplicaciones Android son:

- **AndroidManifest.xml**: es el archivo de control que indica al sistema qué hacer con todos los componentes de alto nivel (en concreto actividades, servicios, receptores de transmisiones y los proveedores de contenido) en una aplicación. También especifica los permisos que son requeridos.
- **Actividades**: la actividad es, en general, el código de una sola tarea centrada en el usuario. Por lo general, incluye la visualización de una interfaz de usuario para

este, pero no tiene por qué (algunas actividades no muestran interfaces de usuario). Por lo general, una de las actividades de la aplicación es el punto de entrada a la misma.

- **Servicios**: Un servicio es el cuerpo del código que se ejecuta en segundo plano. Se puede ejecutar en su propio proceso, o en el contexto del proceso de otra aplicación. Otros componentes se "enlazan" a un servicio e invocan métodos de este a través de llamadas de procedimiento remoto. Un ejemplo de un servicio es el reproductor multimedia, cuando el usuario sale de la interfaz del mismo, el usuario probablemente tiene la intención de seguir escuchando la música todavía, un servicio mantiene la música sonando aún cuando la interfaz de usuario del reproductor se ha cerrado o completado.

- **Receptor de transmisiones**: es un objeto que es instanciado cuando un mecanismo IPC conocido como “Intent” es emitido por el sistema operativo u otra aplicación. Una aplicación puede registrar un receptor para el mensaje de batería baja, por ejemplo, y cambiar su comportamiento en base a esa información.

4.5.3.2 El modelo de permisos Android: acceso a las API protegidas

De manera predeterminada, una aplicación Android sólo puede acceder a una gama limitada de los recursos del sistema. El sistema gestiona el acceso a los recursos de las aplicaciones Android que, si se usan incorrectamente o maliciosamente, pueden afectar desfavorablemente a la experiencia del usuario, la red, o los datos en el dispositivo.

Estas restricciones se aplican en una variedad de formas diferentes. Algunas capacidades se ven limitadas por una falta intencional de la API a la funcionalidad sensible (por ejemplo, no hay API de Android para la manipulación directa de la tarjeta SIM). En algunos casos, la separación de roles proporciona una medida de seguridad, al igual que con el aislamiento por la aplicación de almacenamiento. En otros casos, las API son sensibles para uso de aplicaciones de confianza y protegido a través de un mecanismo de seguridad conocido como permisos.

Estas API protegidas incluyen:

- Funciones de la cámara.
- Datos de localización (GPS).
- Funciones Bluetooth.
- Funciones de telefonía.
- Funciones SMS/MMS.
- Red/Conexiones de datos.

Estos recursos sólo son accesibles a través del sistema operativo. Para hacer uso de la API protegida en el dispositivo una aplicación debe definir las capacidades o funcionalidades que necesita en su archivo “manifest.xml”. En los preparativos para instalar una aplicación, el sistema muestra una interfaz al usuario que indica los permisos solicitados por la aplicación que se va a proceder a instalar y le pregunta si desea continuar con la instalación. Si el usuario continúa con la instalación, el sistema acepta que el usuario ha concedido todos los permisos solicitados. El usuario no puede conceder o denegar permisos individuales, este debe conceder o denegar todos los permisos solicitados en bloque.

Una vez concedida dicha solicitud al usuario se aplican los permisos a la aplicación a medida que esta es instalada. Para evitar la confusión de los usuarios, el sistema no avisa al usuario de nuevo de los permisos concedidos a la aplicación. Las aplicaciones que se incluyen en el núcleo del sistema operativo o por un paquete de OEM (fabricante de equipamiento original) no solicitan los permisos al usuario. Dichos permisos se eliminan si se desinstala la aplicación, por lo que un nueva instalación de la misma aplicación vuelve a mostrar la pantalla de los permisos al usuario.

Dentro de la configuración del dispositivo, los usuarios pueden ver los permisos para las aplicaciones que se han instalado previamente. Los usuarios también pueden desactivar algunas funciones a nivel global cuando lo requieren, por ejemplo, deshabilitar GPS, radio, o Wi-Fi.

En el caso de que una aplicación intentara utilizar una función o característica protegida que no se ha declarado en el archivo “manifest.xml” de la aplicación, la falta de autorización dará lugar a que una excepción de seguridad sea lanzada a la aplicación. Las comprobaciones de permisos API protegidos se aplican en el nivel más bajo posible para evitar la elusión de los mismos. Un ejemplo de la interfaz de usuario cuando se instala una aplicación al tiempo que se le solicita al mismo el acceso a la API protegida se muestra en la Figura 11 que mostraremos en el siguiente punto de este mismo capítulo.

Los permisos por defecto del sistema se describen en puntos posteriores de este mismo capítulo. Las aplicaciones pueden declarar sus propios permisos para usar otras aplicaciones, estos permisos no figuran en la ubicación anterior.

Cuando se define un permiso con el atributo “ProtectionLevel” le indicamos al sistema cómo el usuario debe ser informado de las aplicaciones que requieren de este permiso, o a cuales se le permite mantener este permiso. Detalles sobre la creación y el uso de los permisos de aplicaciones específicas se describen en puntos posteriores de este mismo capítulo.

Hay algunas funcionalidades o capacidades del dispositivo, tales como la función de enviar SMS, que no están disponibles para las aplicaciones externas o de terceros, pero que pueden ser utilizados por las aplicaciones pre-instaladas por el OEM (fabricante de equipamiento original). Estos permisos utilizan la autorización “signatureOrSystem”.

4.5.3.3 ¿Cómo los usuarios comprenden las aplicaciones externas o de terceros?

Android busca dejar claro a los usuarios, cuando interactúan con las aplicaciones externas o de terceros, informales de las capacidades o funcionalidades que tienen las mismas. Antes de la instalación de cualquier aplicación, al usuario se le muestra un mensaje claro sobre los diferentes permisos que la aplicación está solicitando. Después de la instalación, al usuario no se le pedirá de nuevo confirmar dichos permisos.

Hay muchas razones para mostrar dichos permisos inmediatamente antes de instalar la aplicación, es cuando el usuario está revisando activamente la información sobre la aplicación, los desarrolladores y funcionalidad de la misma para determinar si coincide con sus necesidades y expectativas. También es importante que aún no se haya establecido un compromiso (incluso financiero) con la aplicación, y se puede comparar fácilmente esta última con otras aplicaciones alternativas.

Otras plataformas utilizan un enfoque distinto al de notificación de usuario, solicitando los permisos al comienzo de cada sesión o mientras las aplicaciones están en uso. La visión de Android es que los usuarios a su antojo cambien fácilmente entre las aplicaciones. Proporcionar estas confirmaciones cada vez que usásemos una aplicación, haría más lento y privaría a Android de proporcionar una gran experiencia de usuario. Con la revisión de permisos al usuario durante la instalación de la aplicación le proporciona al usuario la opción de no instalar esta si se siente incómodo o no está de acuerdo al cien por cien.

Además, muchos estudios sobre la interfaz de usuario han demostrado que el exceso de preguntar o requerir la confirmación al usuario, hace que este último empiece a pulsar el botón de confirmación para cualquier diálogo que se le muestre, sin entender lo que se le está solicitando en ese momento. Uno de los objetivos de seguridad de Android es el de transmitir adecuadamente la información importante de seguridad al usuario, lo cuál no se puede llegar a conseguir con los cuadros de diálogo que el usuario está predispuesto a ignorar. Mediante la presentación de la información importante una vez, y sólo cuando es importante, es más probable que el usuario esté pensando o sea consciente de lo que está aceptando.

Algunas plataformas optan por no mostrar ningún tipo de información sobre la funcionalidad o capacidades de la aplicación. Este enfoque evita que los usuarios entiendan fácilmente o discutan las capacidades o funcionalidades de la aplicación. Si bien no es posible siempre para todos los usuarios hacer decisiones con pleno conocimiento de las mismas, el modelo de permisos de Android hace que la información sobre las aplicaciones sea de fácil acceso para una amplia gama de usuarios. Por ejemplo, las peticiones de permisos inesperadas pueden inspirar a los usuarios con conocimientos más avanzados a responder preguntas críticas acerca de la funcionalidad de las aplicaciones y compartir sus respuestas en lugares como “Google Play”, donde son visibles para todos los usuarios.

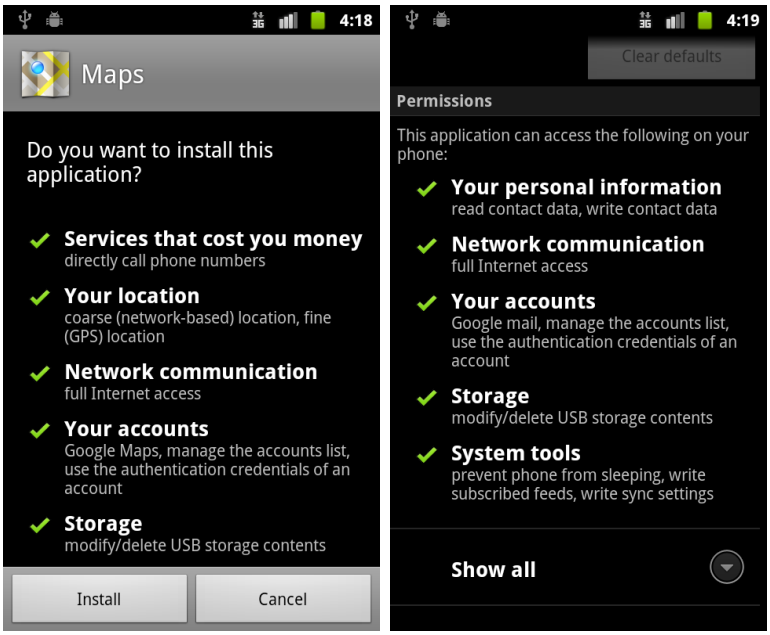


Figura 11.Permisos para instalar las aplicaciones Google Maps y Gmail.

4.5.3.4 Comunicación entre procesos

Los procesos pueden comunicarse con cualquiera de los mecanismos tradicionales de tipo UNIX. Los ejemplos incluyen por ejemplo el sistema de archivos, sockets locales o señales. Sin embargo, los permisos Linux siguen siendo válidos.

Android también proporciona nuevos mecanismos de IPC (comunicación entre procesos - Inter-process Communication):

- **Binder**: una ligera funcionalidad basada en el mecanismo de llamada a procedimiento remoto diseñado para un alto rendimiento cuando se realiza durante el mismo proceso y/o las llamadas entre procesos. "Binder" es implementado utilizando un controlador de Linux personalizado.

- **Services**: pueden proporcionar interfaces de acceso directo usando "Binder".

- **Intents**: Un "Intent" es un simple objeto de mensaje que representa una "intención" de hacer algo. Por ejemplo, si la aplicación desea mostrar una página web, expresa su "intención" para ver la dirección URL mediante la creación de una instancia "Intent" y la entrega fuera del sistema. El sistema detecta alguna otra pieza de código (en este caso, el navegador) que sabe cómo manejar esa intención, y lo ejecuta. Las intenciones también se puede utilizar para transmitir eventos interesantes (como la notificación) para todo el sistema.

- **ContentProviders**: es un almacén de datos que proporciona acceso a los datos en el dispositivo, el ejemplo clásico de "ContentProvider" es el que se utiliza para acceder a la lista de los contactos del usuario. Una aplicación puede acceder a los datos que otras aplicaciones han expuesto a través de un "ContentProvider", al

igual que una aplicación también puede definir sus propios "ContentProviders" para exponer sus propios datos.

Si bien es posible la implementación de IPC a través de otros mecanismos diferentes a los anteriormente mencionados, tales como las conexiones de red o ficheros con permiso de escritura, los explicados con anterioridad son los recomendados para el entorno IPC Android. Los desarrolladores de Android se proponen utilizar las mejores prácticas en torno a la seguridad de los datos de los usuarios y evitar la introducción de vulnerabilidades en la seguridad.

4.5.3.5 API sensible a costos

Un API sensible a costos es una función o funcionalidad que podría generar un coste para el usuario o la red. La plataforma Android ha introducido una API sensible a los costos en la lista de las API protegidas controladas por el sistema operativo. El usuario tendrá que otorgar un permiso explícito para las aplicaciones externas o de terceros que solicitan el uso de las API sensibles a costes. Estas API son:

- Telefonía.
- SMS/MMS.
- Red/Datos.
- Aplicaciones de facturación.
- Acceso NFC (Near Field Communication – tecnología de comunicación de corto alcance para la transmisión de datos).

4.5.3.6 Acceso a la tarjeta SIM

El acceso de bajo nivel a la tarjeta SIM no está disponible para aplicaciones externas o de terceros. El sistema operativo gestiona todas las comunicaciones con la tarjeta SIM, incluyendo el acceso a la información personal (contactos) en la memoria de la tarjeta SIM. Las aplicaciones tampoco pueden acceder a los comandos AT (comandos UNIX que permiten programar la ejecución de uno o varios programas en un momento futuro), ya que estos son administrados exclusivamente por la "Radio Interface Layer" (RIL - es la capa del sistema operativo que proporciona una interfaz de radio para hardware y módem). El EIR (es una base de datos en la que existe información sobre el estado de los teléfonos móviles) no proporciona API de alto nivel para estos comandos.

4.5.3.7 Información personal

Android ha puesto un API que proporciona acceso a los datos del usuario en el conjunto de API protegidos. Con el uso cotidiano, los dispositivos Android acumulan datos personales proporcionados por el usuario para las aplicaciones de terceros instaladas en el dispositivo por los mismos. Las aplicaciones que deciden compartir esta información pueden utilizar las comprobaciones de los permisos del sistema operativo Android proteger los datos de las aplicaciones externas o de terceros.

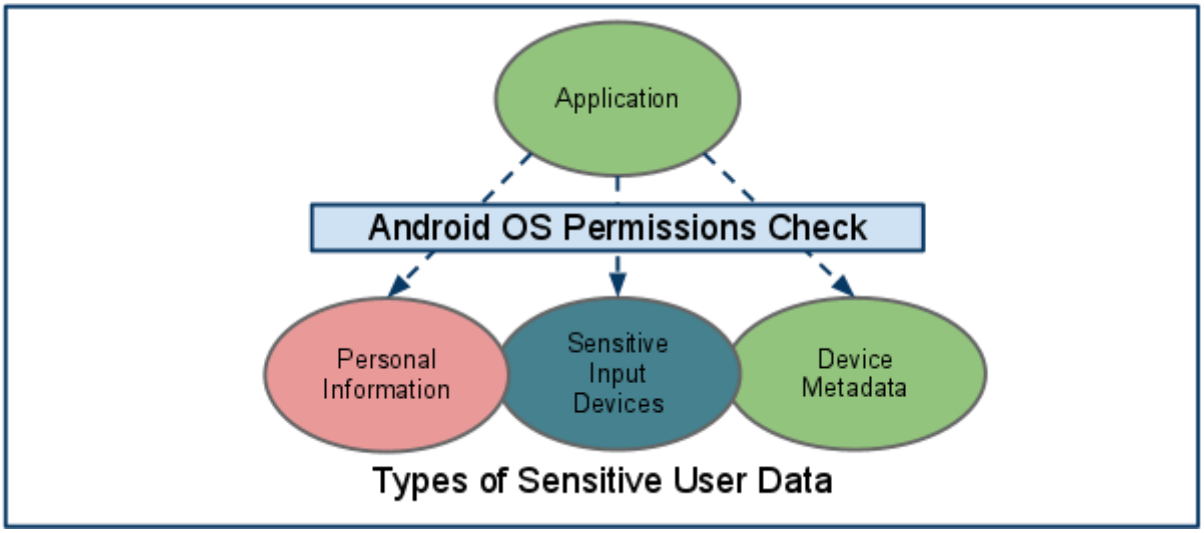


Figura 12. El acceso a la información personal del usuario sólo está disponible a través de las API protegidas.

Los proveedores de contenido del sistema es probable que contengan información personal o de identificación personal, como contactos y/o calendarios que han sido creados con permisos claramente identificados. Esta granularidad proporciona al usuario una clara indicación de los tipos de información que puede ser proporcionada a la aplicación. Durante la instalación, una aplicación ajena o de terceros puede solicitar permiso o varios permisos para acceder a estos recursos. Si le es otorgado dicho permiso o conjunto de permisos, la aplicación podrá ser instalada y tendrá acceso a los datos solicitados en cualquier momento cuando se instale.

Todas las aplicaciones que recopilan información personal tendrán, por defecto, acceso limitado a los datos, únicamente por dichas aplicaciones en específico. Si una aplicación opta por tener sus datos disponibles para otras aplicaciones a través del IPC, la aplicación que concede el acceso a su información puede aplicar los permisos del mecanismo IPC, que son impuestos por el sistema operativo para el intercambio de información entre procesos.

4.5.3.8 Dispositivos de entrada de datos sensibles

Los dispositivos bajo la plataforma Android a menudo proporcionan la entrada de datos sensibles o de alto nivel que permiten a las aplicaciones interactuar con el entorno circundante, tales son como la cámara, el micrófono o el GPS. Para acceder una aplicación externa o de terceros a los dispositivos de este tipo, primero se le debe proporcionar explícitamente el acceso por parte del usuario mediante el uso de los permisos del sistema operativo Android. Tras la instalación, el instalador le pedirá al usuario la solicitud del permiso para el sensor por su nombre.

Si una aplicación quiere saber la ubicación del usuario, la aplicación requiere un permiso para acceder a la ubicación del usuario. Tras la instalación, el instalador le preguntará al usuario si la aplicación puede acceder a la ubicación del usuario. En cualquier momento, si el usuario no quiere que cualquier aplicación pueda acceder a su ubicación, el usuario puede ejecutar la “Configuración” de la aplicación, ir al apartdo

“Ubicación y seguridad”, y desactive la casilla “Utilizar redes inalámbricas” y “Habilitar satélites GPS”. Esto desactivará los servicios de localización para todas las aplicaciones en el dispositivo del usuario.

4.5.3.9 Dispositivos de metadatos

La plataforma Android también se esfuerza por restringir el acceso a datos que no son intrínsecamente sensibles, pero indirectamente puede revelar características del usuario, las preferencias del usuario, y la manera en que se utiliza un dispositivo.

Por defecto las aplicaciones no tienen acceso a los registros del sistema operativo, el historial del navegador, número de teléfono o información de identificación de hardware/red. Si una aplicación solicita acceso a esta información durante la instalación, el instalador preguntará al usuario si la aplicación puede acceder a la información. Si el usuario no permite el acceso, la aplicación no se instalará.

4.5.3.10 Firmado de aplicación

El firmado de código permite a los desarrolladores identificar al autor de la aplicación y actualizar su aplicación sin la creación de interfaces complicadas y permisos. Cada aplicación que se ejecuta en la plataforma Android debe ser firmada por el desarrollador. Las aplicaciones que intenta instalar sin la firma serán rechazadas tanto por Google Play como por el instalador de paquetes del dispositivo bajo la plataforma Android.

En Google Play, la firma de la aplicación proporciona la confianza de Google para con los desarrolladores y la confianza de estos con su aplicación. Los desarrolladores saben que su aplicación es proporcionada, sin modificaciones a los dispositivos Android y estos son los únicos responsables del comportamiento de la aplicación.

En Android, la firma de la aplicación es el primer paso para emplazar la misma en el recinto de seguridad de aplicaciones (“Application Sandbox”). El certificado de la aplicación firmada define qué ID de usuario está asociada a qué aplicación, ya que las diferentes aplicaciones se ejecutan bajo ID de usuario diferentes. La firma de la aplicación asegura que esta no puede acceder a cualquier otra aplicación, excepto a través de los bien definidos permisos IPC.

Cuando una aplicación (archivo “.apk”) se instala en un dispositivo Android, el gestor de paquetes comprueba que el archivo “.apk” ha sido debidamente firmado con el certificado incluido en mismo archivo. Si el certificado (o, más exactamente, la clave pública del certificado) coincide con la clave utilizada para firmar otros archivos “.apk” en el dispositivo, el nuevo archivo tiene la opción de especificar en su “manifest.xml” que compartirá su ID de usuario con los otros archivos firmados “.apk” de manera similar.

Las aplicaciones pueden ser firmadas por un tercero (OEM, operadores, el mercado alternativo) o auto-firmadas. Android ofrece firmado del código usando certificados auto-firmados que los desarrolladores pueden generar sin ayuda o permiso externo. Las aplicaciones no tienen que ser firmadas por una autoridad central,

actualmente no realiza la verificación de certificados de aplicación CA (Autoridad de certificación).

Las aplicaciones también son capaces de declarar los permisos de seguridad a nivel de firma protegida, limitando el acceso únicamente a las aplicaciones firmadas con la misma clave, manteniendo los distintos ID de usuarios y recintos de seguridad de aplicaciones (o "Sandboxes"). Una relación más estrecha o cercana con un recinto de seguridad compartido permite a través de la "función de ID de usuario compartida", en la que dos o más aplicaciones son firmadas con la clave del mismo desarrollador, declarar un ID de usuario compartido en sus archivos "manifest.xml".

4.5.3.11 Gestión de derechos digitales

La plataforma Android proporciona un marco extensible que permite a las aplicaciones DRM, administrar contenido protegido por derechos de autor de acuerdo a las restricciones de licencia que se asocian con el contenido. El marco de DRM es compatible con muchos sistemas de DRM, esta compatibilidad de los sistemas de DRM de un dispositivo se dejan al fabricante de dicho dispositivo.

El marco DRM Android ("Android DRM framework") se aplica en dos capas de la arquitectura (ver figura 13):

- Un marco API DRM, que se aplica a las aplicaciones a través de la estructura de la propia aplicación Android y se ejecuta a través de la máquina virtual Dalvik para aplicaciones estándar.
- Un código nativo gestor DRM, que implementa el marco DRM y expone una interfaz de plug-ins DRM (agentes) para controlar la gestión de derechos y descifrado de los diversos sistemas de DRM.

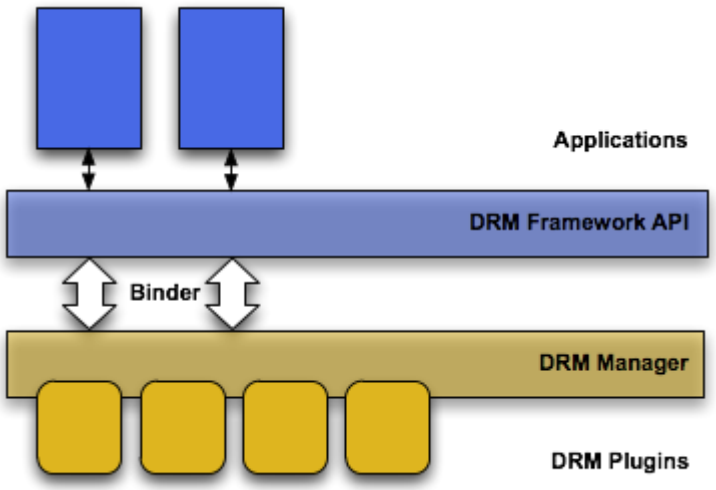


Figura 13. Arquitectura de gestión de derechos digitales en la plataforma Android.

4.5.4 Actualizaciones Android

Android proporciona las actualizaciones del sistema, tanto para propósitos de seguridad como para otras funciones relacionadas.

Hay dos formas de actualizar el código en la mayoría de dispositivos Android: “on-the-air” (actualizaciones OTA) o actualizaciones cargadas. Las actualizaciones OTA se pueden lanza a lo largo de un período de tiempo definido o ser lanzadas a todos los dispositivos a la vez, dependiendo de cómo el OEM y/o distribuidor quieran lanzar las actualizaciones. Las actualizaciones cargadas se pueden proporcionar desde una ubicación central, los usuarios descargan un archivo zip a su equipo o directamente a su dispositivo Android. Una vez que la actualización se ha copiado o descargado a la tarjeta SD del dispositivo, Android reconocerá el archivo de actualización, comprobará su integridad y autenticidad, y actualizará automáticamente el dispositivo.

Si una vulnerabilidad peligrosa es descubierta de forma interna o el responsable informa a Google o al proyecto de código abierto Android, el equipo de seguridad de Android iniciará el siguiente proceso:

1. El equipo de Android notificará a las empresas que han firmado acuerdos de confidencialidad en relación con el problema y comenzará a discutir la solución.
2. Los propietarios de código comenzarán a solucionarlo.
3. El equipo de Android resolverá los problemas relacionados con la seguridad.
4. Cuando hay un parche disponible, será proporcionado a las empresas con acuerdos NDA (acuerdo de confidencialidad).
5. El equipo de Android publicará el parche en el proyecto de código abierto Android.
6. OEM/distribuidor lanzará una actualización a los clientes.

El NDA tiene la obligación de asegurar que el problema de seguridad no se haga público antes de disponer de una solución y, por lo tanto, poner a los usuarios en una situación de riesgo. Muchos miembros de la OHA (alianza comercial de 78 compañías para desarrollar estándares abiertos para dispositivos móviles) ejecutan su propio código en los dispositivos Android, como el gestor de arranque, los controladores de Wi-Fi, y la radio. Una vez que el equipo de seguridad de Android es notificado de un problema de seguridad en este código promocional, se consultará a los miembros de la OHA para encontrar rápidamente una solución al problema en cuestión y/o problemas similares. Sin embargo, el miembro de la OHA, que desarrolló el código defectuoso es responsable en última instancia de solucionar el problema.

Si de una vulnerabilidad peligrosa no se conoce al responsable (por ejemplo, si es enviada a un foro público sin previo aviso), Google y/o el Proyecto de Código Abierto Android trabajarán lo más rápido posible para crear un parche o solución al problema. El parche será lanzado al público (y socios) cuando sea probado y esté listo para su uso.

En Google I/O 2011, muchos de los mayores socios OHA se comprometieron a proporcionar actualizaciones a los dispositivos 18 meses después del lanzamiento de los mismos. Esto proporcionará a los usuarios de los dispositivos el acceso a las características más recientes de Android, así como a las actualizaciones de seguridad.

Cualquier desarrollador, usuario de Android, o investigador de seguridad puede notificar al equipo de seguridad de Android posibles problemas de seguridad mediante el envío de correo electrónico a security@android.com.

4.6 Seguridad en Android para el desarrollo de aplicaciones

En Android cada aplicación se ejecuta en su propio proceso. La mayoría de las medidas de seguridad entre el sistema y las aplicaciones deriva de los estándares de Linux 2.6, cuyo kernel, recuérdese, constituye el núcleo principal de Android.

Android es un sistema operativo de privilegios separados, en el que cada aplicación se ejecuta con un identificador de sistema distinto (ID de usuario de Linux y el ID de grupo). Las partes del sistema también se separan en diferentes identificadores. Linux aísla las aplicaciones entre sí y del sistema.

Las características adicionales de seguridad más concretas y precisas son proporcionadas a través de un mecanismo de "*permisos*" que endurece las restricciones a las operaciones específicas que un proceso determinado puede realizar, y en permisos URI para garantizar el acceso "ad-hoc" a partes específicas de datos.

4.6.1 Arquitectura de seguridad

Un punto central del diseño de la arquitectura de seguridad de Android es la denegación, por defecto, de permisos para realizar cualquier operación que tenga impactos negativos en otras aplicaciones, el sistema operativo, o el usuario. Esto incluye la lectura o escritura de datos privados del usuario (por ejemplo, contactos o mensajes de correo electrónico), leer o escribir archivos de otra aplicación, acceso a la red, mantener en estado "despierto" el dispositivo, habilitación de algún recurso hardware del dispositivo, etc.

Cada proceso en Android constituye lo que se llama un cajón de arena o *sandbox*, que proporciona un entorno seguro de ejecución. Por defecto, ninguna aplicación tiene permiso para realizar ninguna operación o comportamiento que pueda impactar negativamente en la ejecución de otras aplicaciones o del sistema mismo. La única forma de poder saltar estas restricciones impuestas por Android, es mediante la declaración explícita de un permiso que autorice a llevar a cabo una determinada acción habitualmente prohibida.

A causa del sistema "sandbox" (un sistema informático de aislamiento de procesos, mediante el cual, se pueden ejecutar distintos programas con seguridad y de manera separada. A menudo se utiliza para ejecutar código nuevo, o software de dudosa confiabilidad, con objeto de evitar la corrupción de datos del sistema en donde estos se ejecutan) del núcleo que regula la interacción de aplicaciones entre sí, para que las aplicaciones puedan compartir recursos y datos, se declararán los "*permisos"* que necesitan de forma explícita para estas capacidades adicionales, no previstas por el "sandbox" de seguridad base.

Las aplicaciones estáticamente declaran los permisos que requieren, y el sistema Android solicita al usuario su consentimiento en el momento de instalar la aplicación.

Android no tiene ningún mecanismo para la concesión de permisos de forma dinámica (en tiempo de ejecución), ya que complica la experiencia del usuario en detrimento de la seguridad.

El núcleo es el único responsable del sistema "sandbox" de aplicaciones entre sí. En particular, la máquina virtual Dalvik no limita la seguridad puesto que cualquier aplicación puede ejecutar código nativo. Todos los tipos de aplicaciones- Java, nativa e híbridos- hacen uso del mismo sistema sandbox y poseen igual grado de seguridad.

4.6.2 Firma de aplicaciones

Todas las aplicaciones de Android (archivos. APK) deben estar firmadas con un certificado cuya clave privada se encuentre en manos de sus desarrolladores. Este certificado identifica al autor de la aplicación. El certificado "no" necesita ser firmado por una autoridad de certificación: es perfectamente admisible, y típico, para aplicaciones de Android usar certificados auto-firmados. El único cometido del certificado es crear una relación de confianza entre las aplicaciones. Mediante la firma, la aplicación lleva adjunta su autoría.

La finalidad de los certificados en Android es distinguir a los autores de la aplicación. Esto permite que el sistema pueda conceder o denegar las solicitudes de acceso a nivel de firma de las aplicaciones, y otorgar o negar el mismo identificador Linux que otra aplicación.

Niveles de protección: Caracteriza el riesgo potencial que implica la autorización, e indica el procedimiento que el sistema debe seguir, para determinar si concede o deniega el permiso a una aplicación que lo solicite. Hay diferentes niveles de protección:

- Normal: El valor por defecto. Un permiso de poco riesgo que solicita el acceso a las aplicaciones aisladas a nivel de aplicación, con un riesgo mínimo para otras aplicaciones, el sistema, o el usuario. El sistema automáticamente otorga este tipo de permiso para una aplicación que solicita la instalación, sin pedir la aprobación explícita del usuario (aunque el usuario siempre tiene la opción de revisar estos permisos antes de instalar).

- Dangerous: Un permiso de alto riesgo que le daría acceso a la aplicación a solicitar datos privados de usuario o control sobre el dispositivo que puede influir negativamente en el usuario. Debido a que este tipo de permiso implica un riesgo potencial, el sistema automáticamente no lo puede conceder a la aplicación. Los permisos peligrosos solicitados por una aplicación se muestran al usuario y requieren de su confirmación antes de proceder a la instalación, o algún otro método que se pueda tomar para evitar que se permita de forma automática dicha instalación.

- Signature: Sólo si la aplicación solicitante se firma con el mismo certificado que la aplicación que declaró el permiso. Si los certificados coinciden, el sistema automáticamente otorga el permiso sin informar al usuario o solicitar la aprobación explícita del usuario.

- SignatureOrSystem: Sólo las aplicaciones que se encuentran en la imagen del sistema Android o que se suscriban con los mismos certificados que los de la imagen del sistema. El nivel de protección de la firma debe ser suficiente para la mayoría de las necesidades y funciona independientemente de donde están instaladas las aplicaciones. El "signatureOrSystem" se utiliza para ciertas situaciones especiales en que múltiples proveedores tienen aplicaciones integradas en una imagen del sistema y la necesidad de compartir características de forma explícita.

4.6.3 ID de usuarios y acceso a archivos

Android otorga a cada paquete un identificador de usuario Linux (UID). Este identificador se mantiene constante durante la vida del paquete en ese dispositivo. En un dispositivo diferente, el mismo paquete puede tener un UID diferente, lo importante es que cada paquete tiene un UID distinto en un dispositivo determinado.

Debido al refuerzo de seguridad que se aplica a nivel de proceso, el código de dos paquetes cualesquiera normalmente no puede ejecutarse en el mismo proceso, ya que necesitan para funcionar diferentes identificadores de usuarios Linux. Podemos utilizar el atributo "sharedUserId" en el manifiesto de etiquetas "AndroidManifest.xml" de cada paquete que les asigna el mismo ID de usuario. De esta manera, por razones de seguridad los dos paquetes son tratados como la misma aplicación, con el mismo ID de usuario y permisos de archivos. Tenga en cuenta que el fin de mantener la seguridad, sólo dos aplicaciones firmadas con la misma firma (y su interés en la misma sharedUserId) tendrá la misma identificación de usuario.

Todos los datos almacenados por una aplicación se les asignarán el identificador de usuario de la aplicación, normalmente no serán accesibles por otros paquetes. Cuando creamos un nuevo archivo con "`getSharedPreferences (String, int)`", "`openFileOutput (String, int)`", o "`openOrCreateDatabase (String, int, SQLiteDatabase.CursorFactory)`", puede utilizar el "`MODE_WORLD_READABLE`" y / o "`MODE_WORLD_WRITEABLE`", etiquetas que permiten que cualquier otro paquete lea / escriba el archivo. Al establecer estos parámetros, el archivo aún es propiedad de su aplicación, pero su lectura global y / o

permisos de escritura se han establecido adecuadamente por lo que cualquier otra aplicación puede verlo.

4.6.4 Uso de permisos

Una aplicación de Android no tiene permisos asociados a ella por defecto, lo que significa que no puede hacer ninguna cosa que afectaría negativamente la experiencia del usuario o cualquier otro dato en el dispositivo. Para hacer uso de las características protegidas del dispositivo, debe incluir en su AndroidManifest.xml uno o más "`<uses-permission>`", etiquetas para declarar los permisos que cubran las necesidades de la aplicación.

Por ejemplo, una aplicación que necesita controlar los mensajes SMS entrantes. Tendría que especificar:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.myapp" >
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    ...
</manifest>
```

Código 3. Ejemplo de permiso para manipular mensajes SMS. [12]

Durante la instalación, los permisos solicitados por la aplicación son otorgados por el paquete instalador, basado en el chequeo de las firmas de declaraciones de los permisos y/o por la interacción con el usuario. No se realizan comprobaciones con el usuario mientras la aplicación está ejecutándose, o bien, se concede un permiso especial en la instalación, y se puede usar esa funcionalidad como se desee, o bien, el permiso no se concede y cualquier intento de utilizar esa funcionalidad da un error sin avisar al usuario.

En ocasiones al denegar el permiso tendrá lugar una "*SecurityException*" volviendo de nuevo a la aplicación. Sin embargo, esto no garantiza que se produzca todas las ocasiones. Por ejemplo, "*sendBroadcast (intención)*" es un método que comprueba permisos tales como los datos que están siendo entregados a cada receptor (después del retorno de la llamada al método) por lo que no recibirá una excepción si falla el permiso. En la mayoría de los casos, los fallos de autorización se registrarán en el log del sistema.

Los permisos proporcionados por el sistema Android se pueden encontrar en "*Manifest.permission*". Cualquier aplicación puede definir y hacer cumplir sus propios permisos, así que esto no es una lista completa de todos los permisos posibles.

Un permiso en particular puede ser ejecutado en distintas partes durante el funcionamiento del programa:

- En el momento de una llamada en el sistema, para evitar que la aplicación ejecute ciertas funciones.

- Al iniciar un proceso, para evitar que aplicaciones inicien procesos de otras aplicaciones.
- En el caso de procesos de envío y recepción, para controlar quién puede recibir la transmisión, y quién puede enviarla.
- Al acceder y operar en un proveedor de contenidos.
- Accediendo o inicializando un servicio.

4.6.5 Declaración y aplicación de permisos

Para aplicar permisos, primero se han de declarar en "*AndroidManifest.xml*" empleando las etiquetas "<*permission*>" necesarias.

Por ejemplo, una aplicación que quiere controlar quién puede iniciar uno de sus procesos, debería declarar el permiso para esta operación de la siguiente forma:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.me.app.myapp" >
    <permission
android:name="com.me.app.myapp.permission.DEADLY_ACTIVITY"
        android:label="@string/permlab_deadlyActivity"
        android:description="@string/permdesc_deadlyActivity"
        android:permissionGroup="android.permission-group.COST_MONEY"
        android:protectionLevel="dangerous" />
    ...
</manifest>
```

Código 4. Ejemplo de permiso para controlar quién puede iniciar un proceso. [12]

Se requiere el atributo "<*protectionLevel*>", para decirle al sistema cómo el usuario es informado acerca de las aplicaciones que requieren dicho permiso, o a quién se le permite obtener el permiso.

El atributo "<*permissionGroup*>" es opcional y sólo se utiliza para ayudar al sistema a mostrar permisos al usuario. Por lo general, se establece como un grupo estándar del sistema (que figuran en "*android.Manifest.permission_group*") o en casos más concretos lo definiríamos nosotros mismos, sin embargo, se prefiere el uso de grupos ya existentes.

Tengan en cuenta que tanto la etiqueta y descripción debe ser suministrado por el permiso. Hay cadenas de recursos que pueden ser mostradas al usuario cuando están visualizando una lista de permisos ("*Android:label*") o detalles acerca de un único permiso ("*Android:description*"). La etiqueta debe ser breve, en pocas palabras debe describir la pieza clave de la funcionalidad que el permiso está protegiendo. La descripción debe ser un par de líneas que describen lo que permite realizar, la primera

línea, describe el permiso, y la segunda advierte al usuario de las consecuencias de otorgar dicho permiso a una aplicación.

Aquí un ejemplo de una etiqueta y una descripción para el permiso CALL_PHONE:

```
<string name="permlab_callPhone">llamar directamente a los números de
teléfono</string>
<string name="permdesc_callPhone">Permite a la aplicación llamar a
números de teléfono sin nuestra intervención. Aplicaciones
malintencionadas pueden causar llamadas inesperadas en nuestra factura
del teléfono. Tenga en cuenta que esto no permite a la aplicación
realizar llamadas a números de emergencias.</string>
```

Código 5. Ejemplo de una etiqueta. [12]

En la clase `android.Manifest.permission` se especifican todos los posibles permisos que se pueden conceder a una aplicación: utilización de Wi-Fi, Bluetooth, llamadas telefónicas, cámara, Internet, mensajes SMS y MMS, vibrador, etc.

El elemento `<uses-permission>` contempla una serie de atributos que definen y matizan el alcance del permiso dado:

- `android:name`: especificación del permiso que se pretende conceder. Debe ser un nombre de alguno de los listados en la clase `android.Manifest.permission`.
- `android:label`: una etiqueta o nombre convencional fácilmente legible para el usuario.
- `android:permissionGroup`: permite especificar un grupo asociado al permiso. Los posibles grupos se encuentran listados en la clase `android.Manifest.permission_group` y pueden tener valores como `ACCOUNTS` (cuentas válidas de Google), `COST_MONEY` (acciones que llevan vinculadas un pago) o `PHONE_CALLS` (acciones relacionadas con llamadas), entre otros.
- `android:protectionLevel`: determina el nivel de riesgo del permiso, y en función del mismo influye en cómo el sistema otorga o no el permiso a la aplicación. Oscila entre valores desde el 0 hasta el 3.
- `android:description`: descripción textual del permiso.
- `android:icon`: icono gráfico que puede ser asociado al permiso.

4.6.6 La aplicación de permisos en AndroidManifest.xml

Un alto nivel de permisos puede restringir el acceso a los componentes de todo el sistema o la aplicación a través de su AndroidManifest.xml. Todo lo que se requiere es que se incluya el atributo "*android:permission*" en el componente deseado, nombrando el permiso que se utilizará para controlar el acceso a ella.

Permisos de Actividad (aplicando la etiqueta "<activity>"): restringe quién puede iniciar la actividad asociada. El permiso se comprueba durante un "*Context.startActivity ()*" y "*Activity.startActivityForResult ()*" , si la llamada no tiene el permiso necesario entonces se produce una "*SecurityException*" a partir de la llamada.

Permisos de Servicio (aplicando la etiqueta "<service>"): restringe quién puede iniciar o unirse al servicio asociado. El permiso se comprueba durante un "*Context.startService ()*", "*Context.stopService ()*" y "*Context.bindService ()*", si la llamada no tiene el permiso necesario entonces se produce una "*SecurityException*" a partir de la llamada.

Permisos de BroadcastReceiver (aplicando la etiqueta "<receiver>"): restringe quién puede enviar transmisiones al receptor asociado. El permiso se comprueba después de que "*Context.sendBroadcast ()*" sea devuelto, mientras que el sistema intenta entregar la emisión remitida al receptor establecido. Como resultado, un fallo en el permiso no dará lugar a una excepción que será mandada al emisor, sino que simplemente no se entregará el propósito. De la misma manera, un permiso puede ser solicitado con "*Context.registerReceiver ()*" para controlar quién puede transmitir a un receptor registrado previamente. Un permiso puede ser solicitado llamando a "*Context.sendBroadcast ()*" para restringir que tipo de objeto puede recibir la red.

Permisos de ContentProvider (aplicando la etiqueta "<proveedor>"): restringe quién puede acceder a los datos en un ContentProvider (Los proveedores de contenido tienen una protección adicional de seguridad a su disposición, los llamados "*permisos URI*" que describiremos más adelante). A diferencia de los otros componentes, hay dos atributos de permiso que se pueden establecer por separado:

-"*android:readPermission*": restringe quién puede leer desde el proveedor.

-"*android:writePermission*" restringe quién puede escribir en él.

Tenga en cuenta que un proveedor puede estar protegido por ambos permisos, y que si sólo se posee el permiso de escritura no significa que no se pueda leer datos de un proveedor.

Los permisos se comprueban cuando se recuperan datos por primera vez de un proveedor (si no se encuentra el permiso una "*SecurityException*" será lanzada), y realizamos operaciones en el mismo.

Si usamos "*ContentResolver.query()*" requiere la obtención del permiso de lectura, si usamos"*ContentResolver.insert()*", "*ContentResolver.update()*","*ContentResolver.delete(*

)"requiere la obtención del permiso de escritura. En todos estos casos, sin la obtención de los permisos mencionados una "*SecurityException*" será lanzada.

4.6.7 Cumplimiento de permisos en el envío de transmisiones

Además de la validación de los permisos necesarios para enviar datos a un receptor registrado (como se describió anteriormente), también podemos especificar el requerimiento de un permiso en particular para enviar una transmisión.

Al llamar a "*Context.sendBroadcast()*" con una lista de permisos, estamos requiriendo que la aplicación receptora debe soportar la solicitud de permisos con el fin de recibir los datos.

Tenga en cuenta que un receptor y un emisor puede requerir un permiso. Cuando esto sucede, ambas peticiones de permisos deben ser correspondidas.

4.6.8 Cumplimiento de otros permisos

Arbitrariamente los permisos pueden ser aplicados a cualquier llamada de un servicio. Esto se logra con el procedimiento "*Context.checkCallingPermission()*" al cuál le pasamos el permiso deseado y nos devolverá un número entero que indica si ese permiso ha sido concedido al proceso actual.

Tenga en cuenta que esto sólo se puede utilizar cuando se está ejecutando una llamada que viene de otro proceso.

Hay otras formas útiles para comprobar los permisos. Si conocemos el "*pid*" de un proceso, podemos utilizar el procedimiento de contexto "*Context.checkPermission (String, int, int)*" para validar que el permiso ha sido concedido a ese proceso.

Si conocemos el nombre del paquete de otra aplicación, podemos utilizar el procedimiento directo packagemanager "*PackageManager.checkPermission (String, String)*" para saber si para ese paquete en particular ha sido concedido un permiso en específico.

4.6.9 Permisos URI

Un URI es una cadena corta de caracteres que identifica inequívocamente un recurso (servicio, página, documento, dirección de correo electrónico, enciclopedia, etc). Normalmente estos recursos son accesibles en una red o sistema basado en el Protocolo IP.

El URI (Uniform Resource Identifier o Identificador Uniforme de Recursos), es una cadena de caracteres que sirve para identificar recursos de las páginas de la World Wide Web, su principal característica es la de poder reconocerlas en la red o en un servidor mediante los diferentes códigos con los que cuenta respecto a los servicios que ejecuta la página (autor, FTP, correo electrónico, IP, etc.). La sintaxis con la que se presenta al usuario es puesta con normas globales identificadas por todos los servidores,

se basa en una secuencia de caracteres alfabéticos y numéricos, representados en una serie de variaciones. Un URI puede estar representado en una gran variedad de maneras: por ejemplo, lápiz sobre papel, los píxeles en una pantalla, o una secuencia de un conjunto de caracteres codificados de tal manera que puedan ser procesados para su entendimiento. La interpretación de un URI depende de los caracteres utilizados y no de la forma en cómo se les puede representar en un protocolo de la red.

El sistema de permisos estándar descrito hasta ahora a menudo no es suficiente cuando se utiliza con los proveedores de contenido. Un proveedor de contenido puede querer protegerse con permisos de lectura y escritura, mientras que sus clientes directos también tienen a mano URIs específicas de otras aplicaciones para que puedan operar con ellos.

Un ejemplo típico es los archivos adjuntos en una aplicación de correo. El acceso al correo electrónico debe ser protegido por permisos, ya que se trata de datos sensibles del usuario. Sin embargo, si un URI para un archivo adjunto de una imagen se carga a un visor de imágenes, éste último no tiene permiso para abrir el archivo adjunto, ya que no hay un motivo para mantener el permiso de acceso a todo el correo electrónico.

La solución a este problema son los permiso per-URI: al iniciar una actividad o devolver un resultado a una actividad, quien realiza la llamada puede establecer "*Intent.FLAG_GRANT_READ_URI_PERMISSION*" y/o "*Intent.FLAG_GRANT_WRITE_URI_PERMISSION*". Lo anterior permite el acceso al permiso de actividad receptora específica de datos URI, independientemente de si tiene permiso para acceder a los datos en el proveedor de contenido que correspondiente.

Este mecanismo permite a un modelo de capacidad de estilo genérico, donde la interacción del usuario (abrir un archivo adjunto, seleccionar un contacto de una lista, etc) da lugar a la concesión de permisos ad-hoc muy específicos.

Esto puede ser un punto clave para la reducción de los permisos necesarios por las aplicaciones a únicamente aquellos que están directamente relacionados con su comportamiento.

La concesión de permisos URI específicos, sin embargo, requieren algún tipo de cooperación con el proveedor de contenido que soporta dichos URIs. Se recomienda encarecidamente que los proveedores de contenido que implementen este mecanismo, y declaren que soportan dichos URI´s a través del atributo "*android:grantUriPermissions*" o la etiqueta "<grant-uri-permissions>".

Capítulo 5

Firma de aplicaciones Android

5 Firma de aplicaciones

Para la realización de este capítulo de nuestro proyecto fin de carrera, hemos recopilado y estudiado información de las siguientes referencias [13], [25] y [26].

El sistema operativo Android requiere que todas las aplicaciones instaladas deben estar firmadas digitalmente con un certificado cuya clave privada está en posesión del desarrollador de la aplicación. Android utiliza este certificado como medio para identificar al autor de la aplicación y el establecimiento de relaciones de confianza entre las aplicaciones. No se utiliza para controlar las aplicaciones que el usuario puede instalar. No tiene que ser firmado por una autoridad de certificación: es perfectamente admisible, y típico, para las aplicaciones de Android usar certificados con firma personal.

Los puntos importantes para comprender la firma de aplicaciones Android son los siguientes:

- Todas las aplicaciones deben estar firmadas. El sistema no instalará una aplicación que no está firmada.

- Se pueden utilizar certificados con firma personal para firmar sus aplicaciones. No es necesaria una autoridad de certificación.

- Cuando se esté listo para lanzar su aplicación a los usuarios finales, se debe firmar con una clave privada adecuada. No se puede publicar una aplicación que se firma con la clave de depuración generada por las herramientas del SDK de Android.

- El sistema comprueba la fecha de caducidad de un certificado únicamente durante la instalación. Si el certificado de la aplicación expira después de que dicha aplicación esté instalada, la aplicación seguirá funcionando normalmente.

- Se pueden utilizar las herramientas estándar - Keytool y Jarsigner - para generar las claves y firmar su aplicación (archivos .apk).

- Una vez que haya firmado su aplicación, utilice la herramienta “zipalign” para optimizar el paquete APK final.

Android no puede instalar o ejecutar una aplicación que no esté firmada correctamente. Esto se aplica siempre, ya sea en un dispositivo real o en el emulador. Por lo tanto, es necesario configurar la firma antes de ejecutar o depurar la aplicación en un emulador o dispositivo.

La herramienta SDK Android nos ayudará a firmar nuestras aplicaciones durante el desarrollo de la misma. Nos ofrecen dos modos de firma, el modo de depuración y el modo de lanzamiento.

5.1 Estrategia de firmado

Algunos aspectos de la firma de aplicaciones puede afectar a cómo abordar el desarrollo de la aplicación, especialmente si se está planeando lanzar múltiples aplicaciones.

En general, la estrategia recomendada para todos los desarrolladores, es que firmen todas las aplicaciones con el mismo certificado durante todo el ciclo de vida de sus aplicaciones. Hay varias razones por las que se debe hacer:

- Actualización de la aplicación: A medida que actualizamos una aplicación, tendremos que seguir firmando dichas actualizaciones con el mismo certificado o un conjunto de certificados, para que los usuarios actualicen sin problemas la nueva versión de la aplicación. Cuando se instala en el sistema una actualización de una aplicación, se compara el certificado o conjunto de certificados de la nueva versión de la aplicación con los de la versión existente. Si los certificados coinciden exactamente, incluyendo tanto los datos del certificado como el orden de los mismos, el sistema permite la actualización. Si no coinciden, se tendrá que asignar un nombre de paquete diferente a la aplicación, en este caso, es como si se instalase una aplicación completamente nueva.

- Modularidad de aplicaciones: El sistema Android permite a las aplicaciones que están firmadas por el mismo certificado ejecutarse en el mismo proceso si las aplicaciones así lo solicitan, el sistema las trata como si de una sola aplicación se tratase. De esta manera se puede implementar la aplicación en módulos, los usuarios pueden actualizar cada uno de los módulos de forma independiente si fuese necesario.

- Intercambio de código/datos a través de permisos: El sistema Android proporciona la firma de aplicaciones basada en permisos, por lo que una aplicación puede tener la funcionalidad de otra que se firma con un certificado específico. Con la firma de múltiples aplicaciones con el mismo certificado y el uso de la firma basada en las comprobaciones de permisos, sus aplicaciones pueden compartir código y datos de forma segura.

Otra consideración en la determinación de la estrategia de firmado, es el período de tiempo de validez de la clave que va a utilizar para firmar sus aplicaciones:

- Si se tiene intención de aportar actualizaciones de la aplicación, se debe asegurar de que la clave tiene un período de validez que supera la esperanza de vida de dicha aplicación. Cuando el período de validez de la clave expira, los usuarios ya no serán capaces de actualizar a las nuevas versiones de la aplicación.

- Si se va a firmar varias aplicaciones distintas con la misma clave, se debe asegurar de que su período de tiempo de validez de clave sea superior a la esperanza de

vida de todas las versiones de las aplicaciones, incluyendo aplicaciones dependientes que pueden añadirse en el futuro.

- Si se va a publicar la aplicación o conjunto de aplicaciones en “Google Play”, la clave que se debe utilizar para firmar la solicitud o solicitudes deben tener un período de tiempo de validez que termine después del 22 de octubre 2033. Se debe cumplir este requisito para que los usuarios pueden actualizar sin problemas las aplicaciones cuando las nuevas versiones de las mismas estén disponibles.

A medida que se diseña una aplicación, se debe tener en cuenta estos puntos y asegurarse de usar un certificado adecuado para firmar las aplicaciones.

5.2 La firma para el lanzamiento público

Cuando su aplicación está lista para el lanzamiento de otros usuarios, usted debe:

1. Obtener una clave privada adecuada.
2. Compilar la aplicación en modo de lanzamiento.
3. Firmar la aplicación con la clave privada.
4. Construir el paquete APK final.

Si se está desarrollando en Eclipse con el plugin ADT, se puede utilizar el Asistente de Exportación para llevar a cabo la compilación, firma y construcción de los procedimientos. El Asistente de exportación le permite incluso generar un nuevo almacén de claves y una clave privada durante el proceso. Así que si se utiliza Eclipse, se puede ahorrar el compilado y firmado con el plugin ADT.

5.2.1 Obtener una clave privada adecuada

En la preparación para la firma de una aplicación, primero se debe asegurar de que se tiene una clave privada adecuada con la que firmar.

Una clave privada adecuada es aquella que:

- Está en nuestra posesión únicamente.
- Representa a la entidad personal, empresarial o de organización que se identifica con la aplicación.
- Tiene un período de validez que supera la esperanza de vida de la aplicación o conjunto de aplicaciones. Un período de validez de más de 25 años se recomienda.

Si va a publicar la aplicación o conjunto de aplicaciones en “Google Play”, tenga en cuenta que el período de tiempo de validez termine después del 22 de octubre 2033.No se puede cargar una aplicación si se firma con una clave cuya validez expira antes de esa fecha.

- No es la clave de depuración generada por las herramientas SDK de Android.

La clave puede ser auto-firmada. Si no disponemos de una clave adecuada, debemos generar una utilizando la herramienta “Keytool”.

Para generar una clave con la herramienta Keytool, se debe utilizar el comando keytool más algunos de los paramétros que explicaremos a continuación:

- `-genkey`: Genera un par de claves (clave pública y privada).
- `-v`: Activa la salida.
- `-alias <alias_name>`: Define un alias para la clave. Sólo los primeros 8 caracteres del alias se utilizan.
- `-keyalg <alg>`: Define el algoritmo de cifrado a utilizar al generar la clave. Ambos DSA y RSA son compatibles.
- `-keysize <size>`: Define el tamaño de cada clave generada (en bits). Si no se proporciona, Keytool utiliza un tamaño de clave de 1024 bits. En general, se recomienda utilizar un tamaño de clave de 2048 bits o más.
- `-dname <name>`: Define un nombre único para describir el emisor que creó la clave. Este valor se utiliza como el emisor y campo constante en el auto-firmado del certificado. Hay que tener en cuenta que no es necesario especificar esta opción en la línea de comandos. Si no se proporciona, habrá que introducir cada uno de los campos.
- `-keypass <password>`: Define la contraseña para la clave. Como medida de seguridad, es conveniente no incluir esta opción en la línea de comandos. Si no se proporciona, la herramienta “Keytool” requerirá que se introduzca la contraseña. De esta forma, la contraseña no se almacena en el historial de comandos.
- `-validity <valdays>`: Define el período de validez de tiempo de la clave, en días.
- `-keystore <keystore-name>.keystore`: Define el nombre para el almacén de claves que contiene la clave privada.
- `-storepass <password>`: Define una contraseña para el almacén de claves. Como medida de seguridad, es conveniente no incluir esta opción en la línea de comandos. Si no se proporciona, la herramienta “Keytool” requerirá que se introduzca la contraseña. De esta forma, su contraseña no se almacena en el historial de comandos.

5.2.2 Compilar la aplicación en modo lanzamiento

Con el fin de lanzar la aplicación a los usuarios, esta se debe compilar en modo de lanzamiento. En modo de lanzamiento, la aplicación no se firma por defecto y se tendrá que firmar con su clave privada. Se recomienda no divulgar su clave de privada, o firmar con la clave de depuración.

En este capítulo resaltamos el método de firma utilizando Eclipse, y el modo de empleo del mismo, puesto que este Proyecto Final de Carrera se ha desarrollado en dicho entorno.

- Con Eclipse:

Para exportar un . Apk no firmado de Eclipse, se debe utilizar el Explorador de paquetes de aplicaciones sin firmar.

Se puede combinar los pasos de elaboración y firma con el Asistente de Exportación de Eclipse proporcionado por el plugin ADT.

5.2.3 Firmar nuestra aplicación con la clave privada

Cuando tenemos un paquete de aplicaciones listo para ser firmado, podemos firmarlo usando la herramienta jarsigner. Debemos asegurarnos de que el almacén de claves, que contiene la clave privada, esté disponible.

Para firmar la aplicación con Jarsigner, haciendo referencia tanto a la aplicación “.apk” y el almacén de claves que contiene la clave privada con la que firmar el paquete, utilizaremos las siguientes opciones mostradas en la tabla:

- `-Keystore`: El nombre del almacén de claves que contiene la clave privada.
- `-Verbose`: Activar una salida.
- `-Storepass`: La contraseña para el almacén de claves. Como medida de seguridad, si no se proporciona, Jarsigner le pedirá que introduzca la contraseña, de esta forma su contraseña no se almacena en el historial de comandos.
- `-Keypass`: La contraseña de la clave privada. Como medida de seguridad, si no se proporciona, Jarsigner le pedirá que introduzca la contraseña, de esta forma su contraseña no se almacena en el historial de comandos.

A continuación explicaremos un ejemplo usando Jarsigner para firmar un paquete de aplicaciones llamado “my_application.apk”, utilizando como ejemplo de almacén de claves “my-release-key.keystore”.

```
$ jarsigner -verbose -keystore my-release-key.keystore
my_application.apk alias_name
```

Al ejecutar el comando del ejemplo anterior, Jarsigner nos pide las contraseñas de el almacén de claves y de la clave privada. A continuación, se modifica el “.apk”, es decir, la aplicación queda firmada.

Hay que tener en cuenta que podemos firmar una aplicación varias veces con diferentes claves privadas.

Para comprobar que la aplicación queda firmada, se puede utilizar el siguiente comando:

```
$ jarsigner -verify my_signed.apk
```

Si la aplicación se firma correctamente, Jarsigner nos muestra "jar verified".

Si queremos obtener más información sobre la operación de firmado realizada, podemos probar algunos de los siguientes comandos:

```
$ jarsigner -verify -verbose my_application.apk

ó

$ jarsigner -verify -verbose -certs my_application.apk
```

Si añadimos a los comandos de arriba, la opción “`-certs`”, nos mostrará el creador de la clave "CN =".

Si nos apareciera "CN = Android Debug", significaría que la aplicación fue firmada con la clave de depuración generada por el SDK de Android. Si tenemos la intención de lanzar nuestra aplicación al mercado, debemos firmarla con la clave privada en lugar de la clave de depuración.

5.2.4 Construir el paquete APK final

Una vez que hayamos firmado nuestra aplicación con su clave privada, ejecutaremos el comando “`zipalign`” en el archivo de nuestra aplicación. Esta herramienta nos garantiza que todos los datos no comprimidos comiencen con una alineación de bytes en particular, relacionado con el inicio del archivo. Asegurando que dicha alineación sea como máximo de 4 bytes se proporciona una optimización del rendimiento cuando se instala en un dispositivo. Al alinear nuestra aplicación, el sistema Android es capaz de leer archivos con “mmap()”, incluso si contienen datos binarios con restricciones de alineación, en vez de copiar todos los datos del paquete. El beneficio que obtenemos al alinear nuestro paquete de aplicación es la reducción de memoria RAM consumida por la aplicación en ejecución.

La herramienta “zipalign” se proporciona con el SDK de Android, dentro del directorio “tools/”. Para alinear nuestro paquete “.apk” firmado debemos ejecutar:

```
zipalign -v 4 your_project_name-unaligned.apk your_project_name.apk
```

El parámetro “`-v`“ activa la salida detallada (esto es opcional).

El parámetro “`4`” es el byte de alineación que usaremos (no usar otro distinto a 4).

En el primer argumento indicamos nuestro paquete “.apk” firmado (la entrada).

En el segundo argumento indicamos el paquete “.apk” destino (la salida).

Nota: si estamos sobreescribiendo un paquete “.apk” que ya existe, debemos de utilizar el parámetro “`-f`”. Además nuestro paquete “.apk” de entrada debe estar firmado con nuestra clave privada antes de realizar la optimización de dicho paquete con la herramienta “zipalign” . Si lo firmamos después se deshará la alineación de bytes.

5.2.5 Compilar y firmar con el plugin ADT de Eclipse

Como mencionamos anteriormente, en el desarrollo de este Proyecto Final de Carrera cabe destacar la utilización del entorno de trabajo Eclipse, por ello vamos a explicar los pasos anteriores de compilado y firmado de nuestra aplicación utilizando dicho entorno.

Podemos utilizar el Asistente de Exportación para exportar nuestro paquete “.apk” firmado (e incluso crear un nuevo almacén de claves, si fuera necesario). El Asistente de Exportación realiza toda la interacción con la herramienta “Keytool” y “Jarsigner” por nosotros, nos permite firmar el paquete de la aplicación con una interfaz gráfica de usuario en lugar de realizar manualmente los procedimientos para compilar, firmar y alinear dicho paquete.

Una vez que el asistente ha compilado y firmado nuestro paquete de aplicación, también realizará la alineación con la herramienta "zipalign", ya que el Asistente de Exportación utiliza tanto "Keytool" como "Jarsigner".

Debemos asegurarnos de que dichas herramientas sean accesibles desde nuestro equipo, como se describió anteriormente.

Los pasos a seguir para firmar y alinear nuestro paquete ".apk" con Eclipse son:

1. Seleccione el proyecto en el Explorador de Paquetes y seleccione **Archivo> Exportar**.
2. Abrimos la carpeta de Android, seleccionamos Exportar Aplicación Android, y hacemos clic en **Siguiente**.

El Asistente de Exportación de Aplicaciones Android se iniciará, nos guiará en el proceso de firma de la aplicación, incluidos los pasos para la selección de la clave privada con el que firmar el paquete ".apk" (O crear un nuevo almacén de claves y clave privada).

3. Complete el Asistente de Exportación y la aplicación será compilada, firmada, alineada y estará lista para su distribución.

5.3 Asegurar nuestra clave privada

Mantener la seguridad de nuestra clave privada es de suma importancia, tanto para nosotros como para el usuario. Si permitimos que alguien use nuestra clave, o si dejamos nuestro almacén de claves y contraseñas en un lugar no seguro de tal manera que un tercero pueda encontrarlas y utilizarlas, nuestra autoría y por tanto la confianza de el usuario se verá comprometida.

Si un tercero utilizase nuestra clave sin nuestro consentimiento o conocimiento, esa persona podría firmar y distribuir aplicaciones maliciosas reemplazando nuestras aplicaciones o corrompiéndolas.

También podría firmar y distribuir aplicaciones haciéndose pasar por nosotros para atacan a otras aplicaciones o al sistema en sí, y también corromper o robar datos del usuario.

Nuestra reputación como desarrolladores depende de la seguridad de nuestra clave privada propiamente dicha, en todo momento, hasta que la clave expire.

Estos son algunos consejos para mantener la seguridad de nuestra clave:

- Seleccionar contraseñas sólidas para el almacén de claves y la clave.
- Cuando generemos la clave con la herramienta "Keytool", *no* utilizaremos los parámetros "`-storepass`" y "`-keypass`" en la línea de comandos. Si los

utilizáramos, nuestras contraseñas estarían disponibles en el historial de la consola y cualquier usuario de nuestro equipo podría tener acceso a ellas.

- Por el mismo motivo, al firmar nuestras aplicaciones con la herramienta “Jarsigner”, *no* utilizaremos los parámetros “`-storepass`” o “`-keypass`” en la línea de comandos.
- No comunicar o prestar a nadie nuestra clave privada, y no permitir que personas no autorizadas conozcan las contraseñas del almacén de claves y la clave.

En general, si utilizamos el sentido común y tenemos precaución cuando generamos, utilizamos y/o almacenamos nuestra clave, esta seguirá siendo segura.

Capítulo 6

Versionar aplicaciones Android

6 Versionando nuestras aplicaciones

Para la realización de este capítulo de nuestro proyecto fin de carrera, hemos recopilado y estudiado información de las siguientes referencias [12], [14], [25] y [26].

El control de las versiones de nuestras aplicaciones es un componente crítico para la estrategia de actualización/mantenimiento de las mismas.

- Los usuarios necesitan disponer de información específica acerca de la versión de la aplicación que se instala en el dispositivo y las versiones de actualización disponibles para su instalación.
- Otras aplicaciones, incluyendo aplicaciones que se publican como una “suite”, necesitan consultar al sistema la versión de las mismas, para determinar la compatibilidad e identificar las dependencias.
- Los servicios a través de los cuales se va a publicar nuestra aplicación o conjunto de aplicaciones también tienen que consultar la versión o versiones de las mismas, de modo que puedan mostrar dichas versiones a los usuarios. A un servicio de publicación también le puede ser necesario comprobar la versión de la aplicación

para determinar la compatibilidad y establecer las relaciones de actualización o desactualización.

El sistema operativo Android en sí, *no siempre verifica* la información de la versión de la aplicación como para hacer cumplir las restricciones de las actualizaciones, compatibilidad, etc. En su lugar, sólo los usuarios o las mismas aplicaciones son responsables de hacer cumplir dichas restricciones de las versiones de aplicaciones.

Por el contrario el sistema operativo Android *verifica* la compatibilidad con cualquier versión del sistema, expresada por la aplicación en el archivo “manifest.xml” con el atributo `minSdkVersion`. Esto permite a una aplicación especificar el nivel de API mínimo del sistema con el que es compatible.

6.1 Configurar la versión de nuestra aplicación

Para configurar la información de la versión de nuestra aplicación, utilizamos los atributos del archivo “manifest.xml” de nuestra aplicación. Tenemos dos atributos disponibles para ello, y siempre se deben asignar los valores para ellos:

- `android:versionCode`: Es un entero que representa la versión del código de la aplicación, en relación a otras versiones.

  El valor es un número entero para que otras aplicaciones mediante su programación lo puedan evaluar, por ejemplo, para comprobar una actualización o una relación con versiones anteriores. Podemos establecer cualquier valor numérico entero que desee, sin embargo, debemos asegurarnos de que en cada versión sucesiva de la aplicación utilizaremos un valor mayor al último asignado. El sistema no nos exige este comportamiento, pero el aumento del valor de las sucesivas versiones es normativo.

  Normalmente, lanzaríamos la primera versión de nuestra aplicación con el `versionCode` establecido en 1, entonces aumentaríamos este valor con cada nueva versión, independientemente de que la actualización constituya un mayor o menor aumento de las funcionalidades de la aplicación. Esto quiere decir que el valor del atributo `android:versionCode` no necesariamente tiene una relación con la versión de la aplicación lanzada para el usuario. Las aplicaciones y servicios de publicación no deberían mostrar este valor de la versión de la aplicación a los usuarios.

- `android:versionName`: Es una cadena de caracteres que representa la versión del código de la aplicación lanzada, el valor de dicha cadena describe la versión de la aplicación que mostramos a los usuarios.

  Al igual que el atributo `android:versionCode` el sistema operativo Android no utiliza este valor para propósitos internos, únicamente muestra la información de

la aplicación a los usuarios. Los servicios de publicación también pueden recoger el valor del atributo `android:versionName` para posteriormente mostrarlo a los usuarios.

Definimos estos atributos relacionados con la versión de nuestra aplicación dentro de la etiqueta `<manifest>` del archivo “manifest.xml”.

He aquí un ejemplo de lo mencionado anteriormente:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.example.package.name"
      android:versionCode="2"
      android:versionName="1.1">
    <application android:icon="@drawable/icon"
android:label="@string/app_name">
        ...
    </application>
</manifest>
```

En este ejemplo, hay que tener en cuenta que el valor del atributo `android:versionCode` indica que el actual “.apk” contiene la segunda versión del código de la aplicación, que corresponde a una de las versiones de valor menor lanzadas al usuario, como lo demuestra la cadena de caracteres del atributo `android:versionName`.

El framework de Android proporciona una API para que las aplicaciones puedan consultar la información de la versión del sistema de nuestra aplicación.

Para obtener dicha información sobre la versión, las aplicaciones utilizan el método “*getPackageInfo (int java.lang.String)*”, de la clase “*PackageManager*”.

6.2 Especificando los requerimientos del sistema API de nuestra aplicación

Si nuestra aplicación requiere de una versión específica mínima de la plataforma Android, o estamos desarrollando sólo para un cierto rango de versiones del sistema operativo, podemos especificar estos requisitos en los identificadores del nivel del API utilizado en el archivo “manifest.xml” de nuestra aplicación.

Si lo hacemos, nos aseguramos de que nuestra aplicación sólo se puede instalar en dispositivos que tienes instalada una versión compatible del sistema operativo Android.

Para especificar los requisitos de nivel de API, agregaremos la etiqueta `<uses-sdk>` en el archivo “manifest.xml” de nuestra aplicación, con uno o más de los siguientes atributos:

- `android:minSdkVersion`: Indica la versión mínima del sistema Android en el que se ejecutará la aplicación, especificado por el identificador de nivel del API.
- `android:targetSdkVersion`: especifica el nivel de API en el que se ha desarrollado la aplicación a ejecutar. En algunos casos, esto permite que la aplicación pueda utilizar elementos o comportamientos definidos en el nivel del API objetivo, en lugar de limitarse a usar sólo las definidas para el nivel mínimo de API.
- `android:maxSdkVersion`: La versión más avanzada de la plataforma Android para la que se ha diseñado la aplicación a ejecutar, especificado por el identificador de nivel del API.

Al prepararse para instalar la aplicación, el sistema comprueba el valor de estos atributos y los comparará con la versión del sistema. Si el valor del atributo `android:minSdkVersion` es mayor que la versión del sistema, este último abortará la instalación de la aplicación. Del mismo modo, el sistema instalará la aplicación sólo si el valor del atributo `android:maxSdkVersion` es compatible con la versión de la plataforma.

Si no especificamos estos atributos en nuestro archivo “manifest.xml”, el sistema asume que nuestra aplicación es compatible con todas las versiones de la plataforma, sin ningún nivel máximo del API.

Para especificar una versión mínima de la plataforma para nuestra aplicación, agregaremos la etiqueta `<uses-sdk>` dentro de la etiqueta `<manifest>`, y a continuación, definiremos el atributo `android:minSdkVersion`.

6.3 Niveles de API Android

A medida que desarrollamos nuestra aplicación para la plataforma Android, nos será útil comprender el enfoque general de la plataforma sobre la gestión de cambios de los niveles de API. Es importante entender el identificador del nivel de API y el papel que desempeña en la compatibilidad de la aplicación con los dispositivos en los que se podrá instalar.

En los puntos siguientes proporcionaremos información acerca del nivel del API y cómo afecta a nuestras aplicaciones.

6.3.1 Definición de nivel del API

El nivel del API es un valor entero que identifica de forma exclusiva la versión del framework API que ofrecen distintas versiones de la plataforma Android.

La plataforma Android proporciona un framework API que las aplicaciones pueden utilizar para interactuar con la base del sistema Android.

El API se compone de:

- Un conjunto de paquetes y clases.
- Un conjunto de etiquetas y atributos XML para la utilización en el archivo “manifest.xml” de nuestra aplicación.
- Un conjunto de etiquetas y atributos XML para declarar y tener acceso a los recursos.
- Un conjunto de “intents” para nuestras actividades de la aplicación.
- Un conjunto de permisos que pueden solicitar nuestras aplicaciones, así como refuerzos de permisos incluidos en el sistema.
- Cada versión sucesiva de la plataforma Android puede incluir cambios al framework API de Android que ofrece.

Los cambios en el framework API se diseñan para que la nueva API sigua siendo compatible con versiones anteriores de la misma. Es decir, la mayoría de los cambios en la API son aditivos e introducen o reemplazan nuevas funcionalidades.

Como hay partes del API que se actualizan, las partes más viejas son reemplazadas al quedar obsoletas pero no se eliminan, de modo que las aplicaciones existentes se puedan seguir utilizando.

En un número muy pequeño de ocasiones, algunas partes del API pueden ser modificadas o eliminadas, aunque por lo general estos cambios sólo son necesarios para asegurar la robustez del API y la aplicación, o la seguridad del sistema. En todas las otras partes de versiones anteriores del API no se llevan a cabo

ninguna modificación.

El framework API que nos ofrece una plataforma de Android se especifica mediante un identificador entero llamado “nivel del API”.

Cada versión de la plataforma Android soporta exactamente un nivel del API, aunque por lo anteriormente mencionado quedan implícitos los niveles anteriores de dicho API (hasta llegar al nivel del API 1). La versión inicial de la plataforma Android proporciona un nivel de API y en versiones posteriores se han ido incrementado el nivel del API. La siguiente tabla especifica el nivel de API con el apoyo de cada versión de la plataforma Android:

Versión de la plataforma	**Nivel del API**
Android 4.3	18
Android 4.2.2	17
Android 4.2	
Android 4.1.1	16
Android 4.1	
Android 4.0.4	15
Android 4.0.3	
Android 4.0	14
Android 3.2	13
Android 3.1.x	12
Android 3.0.x	11
Android 2.3.4	10
Android 2.3.3	

Android 2.3	9
Android 2.2.x	8
Android 2.1.x	7
Android 2.0.1	6
Android 2.0	5
Android 1.6	4
Android 1.5	3
Android 1.1	2
Android 1.0	1

Tabla 3. Versiones de la plataforma y APIs [12].

6.3.2 Como utilizar los niveles del API para la plataforma Android

Como ya hemos mencionado en capitulo anteriores, el identificador de nivel del API desempeña un papel clave para asegurar la mejor experiencia posible para los usuarios y desarrolladores de aplicaciones, en este punto profundizaremos más sobre el tema tratado:

- Permite a la plataforma Android describir la versión máxima del framework del API que soporta.

- Permite a las aplicaciones describir la versión del framework del API que requieren.

- Esto permite que el sistema de negociar la instalación de aplicaciones en el dispositivo del usuario, de tal manera que es incompatible la versión de las aplicaciones no están instaladas.

Cada versión de la plataforma Android almacena su identificador de nivel del API internamente, en el propio sistema operativo Android.

Las aplicaciones utilizan una etiqueta en su archivo "manifest.xml" que proporciona el framework del API, `<uses-sdk>`, para describir el nivel mínimo y máximo del API en las que son capaces de ejecutarse, así como el nivel específico del API para el que han sido diseñadas para soportar. La etiqueta cuenta con tres atributos clave:

- `android:minSdkVersion`: especifica el nivel mínimo del API en el que la aplicación es capaz de ejecutarse. El valor por defecto es "1".

- `android:targetSdkVersion`: especifica el nivel del API en el que se ha diseñado la aplicación para ejecutarse. En algunos casos, esto permite que la aplicación utilice etiquetas del archivo “manifest.xml” o comportamientos definidos en el nivel del API especificado, en lugar de limitarse a usar sólo las definidas para el nivel mínimo del API.

- `android: maxSdkVersion`: especifica el nivel máximo del API en el que la aplicación es capaz de ejecutarse.

Por ejemplo, para especificar el nivel mínimo del API que una aplicación necesita para funcionar, la aplicación incluye en su archivo “manifest.xml” una etiqueta `<uses-sdk>` con el atributo `android:minSdkVersion`.

El valor de dicho atributo sería un entero correspondiente al primer nivel del API de la plataforma Android para el cual la aplicación se puede ejecutar.

Cuando el usuario instala una aplicación, o cuando se revalida una aplicación después de una actualización del sistema, este último comprueba primero los atributos de la etiqueta `<uses-sdk>` del archivo “manifest.xml” de la aplicación y se comparan los valores con respecto a su propio nivel del API interno. El sistema permite que comience la instalación sólo si se cumplen estas condiciones:

Si el atributo `android:minSdkVersion` es declarado, su valor debe ser menor o igual al valor entero del nivel del API del sistema. Si no se declara dicho atributo, el sistema asume que la aplicación requiere un nivel del API 1.

Si el atributo `android:maxSdkVersion` es declarado, su valor debe ser igual o mayor al valor entero del nivel del API del sistema. Si no se declara dicho atributo, el sistema asume que la aplicación no tiene un nivel máximo de dicho API.

A continuación mostraremos un ejemplo de cómo quedaría un archivo “manifest.xml” con la etiqueta y atributos anteriormente mencionados:

```
<manifest>
  <uses-sdk android:minSdkVersion="5" />

  <uses-sdk android:maxSdkVersion="10"/>
  ...
</manifest>
```

La razón principal de que en una aplicación se declare un nivel del API con el atributo `Android:minSdkVersion` es indicarle al sistema operativo Android que se está utilizando las APIs que se introdujeron en ese nivel del API especificado.

Si la aplicación pudiera ser instalada de alguna manera en una plataforma con un nivel inferior de la API, posteriormente, nos daría un error en tiempo de ejecución cuando al acceder a las APIs se encontrara con que no existen. El sistema impide que se dé este error al no permitir que la aplicación se instale si el nivel más

bajo del API que se requiere es mayor que el de la versión de la plataforma Android del dispositivo destino.

Por ejemplo, el paquete android.appwidget (responsable de los widget de las aplicaciones) se introdujo en el API de nivel 3. Si una aplicación utiliza esa API, se debe declarar un atributo `android:minSdkVersion` con un valor de "3". La aplicación será instalable en plataformas como Android 1.5 (Nivel API 3) y Android 1.6 (Nivel API 4), pero no en el Android 1.1 (Nivel 2 del API) y las plataformas Android 1.0 (Nivel 1 del API).

6.3.3 Consideraciones para el desarrollo de la aplicación

En los siguientes apartados proporcionaremos información relacionada con el nivel del API que debemos considerar al desarrollar nuestra aplicación.

6.3.3.1 Futura compatibilidad de aplicaciones

Las aplicaciones de Android son generalmente compatibles con las nuevas versiones de la plataforma Android.

Debido a que casi todos los cambios en el framework del API son aditivos, una aplicación desarrollada para alguna versión determinada del API de Android(como se especifica en su nivel API) es compatible con las versiones posteriores de la plataforma y los niveles más altos del API. La aplicación deben ser capaces de funcionar en todas las versiones posteriores de la plataforma, excepto en casos aislados en que la aplicación utiliza una parte del API que fuera posteriormente eliminada por alguna razón.

La compatibilidad es importante porque muchos dispositivos con Android reciben vía “on-the-air” (OTA) actualizaciones del sistema. El usuario puede instalar la aplicación y usarla con éxito, y más tarde recibir una actualización OTA a una nueva versión de la plataforma Android. Una vez instalada la actualización, la aplicación se ejecutará en un nuevo tiempo de ejecución de la nueva versión del sistema, porque este último tiene las funcionalidades del API y del sistema de las que depende la aplicación.

En algunos casos, los cambios por debajo del API, como las del sistema subyacente en sí, puede afectar a nuestra aplicación cuando se ejecuta en el nuevo entorno. Por esa razón es importante que nosotros, como desarrolladores de las aplicaciones, entendamos cómo la aplicación se muestra y se comporta en cada entorno del sistema. Para ayudar a probar la aplicación en varias versiones de la plataforma Android, el SDK de Android incluye múltiples plataformas que se pueden descargar. Cada plataforma incluye una imagen compatible con el sistema que se puede ejecutar en el plugin AVD, para probar la aplicación.

6.3.3.2 Retro-compatibilidad de aplicaciones

Las aplicaciones de Android no son necesariamente compatibles con versiones de la plataforma más antiguas que la versión para las que fueron compiladas.

Cada nueva versión de la plataforma Android puede incluir nuevas APIs, estas últimas dan acceso a las aplicaciones a utilizar las capacidades de la nueva plataforma o sustituir los actuales componentes del API.

Las nuevas APIs son accesibles a las aplicaciones cuando se ejecutan en la nueva plataforma y, como se mencionó anteriormente, también cuando se ejecuta en versiones posteriores de dicha plataforma, como se especifica en el nivel del API.

En el caso contrario, las versiones anteriores de la plataforma no incluyen las nuevas APIs, las aplicaciones que utilizan las nuevas APIs no son capaces de funcionar en estas plataformas.

Aunque es poco probable que un dispositivo con Android pueda ser rebajado a una versión anterior de la plataforma, es importante darse cuenta de que es más que probable que haya muchos dispositivos en el mercado con versiones anteriores de la plataforma. Incluso entre los dispositivos que reciben actualizaciones OTA, algunos podrían sufrir un retraso y no recibir una actualización en una cantidad significativa de tiempo.

6.3.3.3 Selección de una versión de la plataforma Android y el nivel del API

Cuando estemos desarrollando la aplicación, tendremos que elegir la versión de la plataforma contra el cual se compilará la aplicación. En general, se debe compilar la aplicación con la versión más baja posible de la plataforma que nuestra aplicación puede soportar.

Se puede determinar la versión más baja posible de la plataforma mediante la compilación de la aplicación progresivamente contra varios tipos.

Después de determinar la versión más baja, se debe crear una AVD con la versión de la plataforma correspondiente (y el nivel del API) y comprobar que toda nuestra aplicación funcione correctamente.

Por último debemos asegurarnos de declarar el atributo `android:minSdkVersion` en el archivo “manifest.xml” de nuestra aplicación y establecer su valor como el nivel del API de la versión de la plataforma probada.

6.3.3.4 Declaración del nivel del API mínimo

Si desarrollamos una aplicación que utiliza las características o funcionalidades de las APIs o del sistema introducidas en la última versión de la plataforma, debemos configurar el atributo `android:minSdkVersion` al nivel del API de la versión más reciente de la plataforma.

Esto nos asegura que los usuarios sólo podrán instalar la aplicación si sus dispositivos tienen instalada una versión compatible de la plataforma Android. A su vez, esto garantiza que la aplicación pueda funcionar correctamente en sus dispositivos.

Si nuestra aplicación utiliza las APIs introducidas en la última versión de la plataforma pero no declaramos un atributo `android:minSdkVersion`, entonces nuestra aplicación se ejecutará correctamente en dispositivos que tienen instalada la última versión de la plataforma, pero no en los dispositivos que tengan versiones anteriores de la misma.

En este último caso, la aplicación nos dará un error en tiempo de ejecución cuando trate de utilizar las APIs que no existen en dichas versiones anteriores.

6.3.3.5 Probando los niveles más altos del API

Después de compilar nuestra aplicación, debemos asegurarnos de que haremos las pruebas de la misma en la plataforma Android que corresponde al atributo `android:minSdkVersion` que hemos especificado en el archivo “manifest.xml”.

Para ello, crearemos un AVD que utilice la versión de la plataforma que requiere la aplicación. Además, para asegurar la compatibilidad con versiones posteriores, debemos ejecutar y probar nuestra aplicación en todas las plataformas que utilizan un nivel API superior al especificado en nuestra aplicación.

El SDK de Android incluye varias versiones de la plataforma que se pueden utilizar, incluida la última versión, y proporciona una herramienta de actualización que puede utilizar para descargar otras versiones de la plataforma cuando sea necesario.

Para tener acceso a las actualizaciones, utilizaremos la herramienta de la línea de comandos Android, esta se encuentra en el directorio “<`sdk`>/tools”. Podemos iniciar el programa de actualización mediante el comando sin especificar ninguna otra opción. También podemos hacer doble clic en el archivo android.bat (Windows) o el archivo android (OS X / Linux). Con el ADT, también podemos acceder a las actualizaciones, seleccionando “Ventana> Android SDK y AVD Manager”.

Para ejecutar la aplicación en diferentes versiones de la plataforma en el emulador, crearemos un AVD para cada versión de la plataforma en la que deseemos probar. Si por el contrario estamos usando un dispositivo físico para realizar las pruebas, debemos asegurarnos de que conocemos el nivel del API de la plataforma Android que tenemos instalada en dicho dispositivo. Podemos consultar la tabla descrita anteriormente en este documento (punto 5.3.1) para obtener una lista de las versiones de la plataforma y sus niveles de API.

6.3.4 Usando un nivel API provisional

En algunos casos, versiones de prueba de una plataforma pueden estar disponible en el SDK de Android. Para que podamos comenzar a desarrollar para esa plataforma a pesar de que el API no sea el definitivo.

El entero que denota el nivel del API de la plataforma no debemos especificarlo. En su lugar, debemos utilizar un nivel del API provisional de la plataforma en nuestro archivo “manifest.xml” de la aplicación, con el fin de crear aplicaciones sobre dicha plataforma. Estableceremos un nivel del API que no será un entero, sino una cadena que coincida con el nombre en clave de la versión de la plataforma aún inacabada.

El uso de estos niveles provisionales del API está diseñado para proteger a desarrolladores y usuarios de los dispositivos de publicaciones o instalaciones de aplicaciones basadas en versiones de prueba del API, que no se ejecuten correctamente en dispositivos con versiones reales de la imagen final de la plataforma.

El nivel API provisional sólo nos será válido mientras se use en el SDK y sólo se puede utilizar para ejecutar aplicaciones en el emulador. Una aplicación que utilice un nivel de API provisional no se puede instalar en un dispositivo Android.

Para instalar nuestra aplicación en la versión final de la plataforma, se debe reemplazar todas las instancias del nivel provisional del API de nuestro archivo “manifest.xml” con valores enteros del nivel del API de la plataforma final.

Capítulo 7

Publicar aplicaciones Android

7 Publicación de nuestras aplicaciones

Para la realización de este capítulo de nuestro proyecto fin de carrera, hemos recopilado y estudiado información de las siguientes referencias [15], [25] y [26].

En este proyecto resaltaremos la publicación de aplicaciones en “Google Play” ya que es la forma más extendida y fiable de hacer públicas nuestras aplicaciones.

Publicar una aplicación significa que ha sido probada, compilada y firmada correctamente para ponerla a disposición de los usuarios de dispositivos móviles Android.

Si vamos a publicar nuestra aplicación para su instalación en dichos dispositivos, hay varias cosas que necesitamos hacer para tener la aplicación preparada. A continuación resaltaremos algunos puntos importantes de la preparación de la aplicación para tener un lanzamiento exitoso.

Antes de considerar nuestra aplicación lista para el lanzamiento :

1. Probar nuestra aplicación, en gran medida, en un dispositivo real y no sólo en el emulador.

2. Considerar la adición de un Acuerdo de Licencia de nuestra aplicación.
3. Considerar la posibilidad de añadir soporte de licencias.
4. Especificar un icono y una etiqueta en el archivo “manifest.xml” de la aplicación.
5. Desactivar el registro y depuración, y limpiar los datos/ficheros.

Antes de realizar el compilado final de nuestra aplicación:

1. Versionar nuestra aplicación.
2. Obtener una clave criptográfica adecuada.
3. Registrarnos para obtener una clave API de Google Maps, si nuestra aplicación utiliza elementos “MapView”.

Compilar nuestra aplicación.

Después de compilar la aplicación:

1. Firmar la aplicación.
2. Poner a prueba la aplicación ya compilada.

A continuación en posteriores apartados, detallaremos en qué consisten los puntos clave anteriormente mencionados.

7.1 Antes de considerar nuestra aplicación lista para lanzarla

1. Probar nuestra aplicación, en gran medida, en un dispositivo real y no sólo en el emulador:

Es importante probar la aplicación lo más ampliamente posible, en tantas áreas como nos sea posible. Para ayudarnos en esta tarea, Android nos ofrece una variedad de clases de pruebas y herramientas muy útiles.

Podemos utilizar la clase `instrumentation` para ejecutar "JUnit" y otros test de pruebas, también podemos utilizar herramientas de pruebas tales como "UI/Application Exerciser Monkey".

- Para garantizar que nuestra aplicación se ejecutará correctamente a los usuarios, debemos hacer todo lo posible para obtener uno o más dispositivos móviles físicos del tipo en el que se espera que la aplicación se ejecute. A continuación, debemos probar la aplicación en los dispositivos reales, bajo condiciones de red reales. Las pruebas de la aplicación en el dispositivo físico son muy importantes, ya que nos permiten comprobar que los elementos de nuestra interfaz de usuario son del tamaño correcto (sobre todo para la interfaz de usuario de pantalla táctil) y que el rendimiento de nuestra aplicación y eficiencia con la batería son aceptables.

- Si no podemos obtener un dispositivo móvil del tipo idóneo para la aplicación, puede utilizar las opciones que nos proporciona el emulador mediante los parámetros `-dpi`, `-device`, `-scale`, `-netspeed`, `-netdelay`, `-cpu`, `-delay` y otros para el emulador pantalla, rendimiento de la red, y otros atributos para que coincida con el dispositivo deseado en la mayor medida de lo posible. Así, podemos comprobar la interfaz de usuario de la aplicación y el rendimiento. Sin embargo, es recomendable que probemos la aplicación en un dispositivo real antes de publicarla.

2. Considerar la adición de un Acuerdo de Licencia de nuestra aplicación.

Para proteger nuestra persona, organización, y la propiedad intelectual, es posible que deseemos proporcionar un "End User License Agreement" (EULA) con nuestra aplicación.

3. Considerar la posibilidad de añadir soporte de licencias.

Si vamos a publicar una aplicación de pago a través de Google Play, tenemos que considerar la adición de soporte para las licencias de Google Play. Estas licencias nos permiten controlar el acceso a nuestra aplicación en función de si el usuario actual tiene que comprarla o no. Estas licencias de uso de Google Play son opcionales.

4. Especificar un icono y una etiqueta en el archivo "manifest.xml" de la aplicación.

El icono y la etiqueta que se especifiquen en el archivo “manifest.xml” de la aplicación son importantes porque se muestra a los usuarios como el icono de la aplicación y el nombre de la misma. Se muestran en la pantalla principal del dispositivo, así como en la administración de aplicaciones, Mis descargas y otros lugares. Además, los servicios de publicación pueden mostrar el icono y la etiqueta a los usuarios.

Para especificar un icono y la etiqueta, debe definir los atributos `android:icon` y `android:label` en la etiqueta `<application>` del archivo “manifest.xml”.

5. Desactivar el registro y depuración, y limpiar los datos/ficheros.

Para la publicación, debemos asegurarnos de que las herramientas de depuración han sido detenidas, y que datos innecesarios de depuración y otros archivos han sido eliminados de nuestro proyecto de la aplicación.

- Eliminaremos el atributo `android:debuggable="true"` de la etiqueta `<application>` del archivo “manifest.xml”.
- Eliminaremos archivos de registro, copias de seguridad, y otros archivos innecesarios del proyecto de la aplicación.
- Verificaremos los datos privados o de propiedad y los eliminaremos si es necesario.
- Desactivar todas las llamadas a los métodos de `Log` en el código fuente.

7.2 Antes de realizar el compilado final de la aplicación

1. **Versionar nuestra aplicación.**

Antes de compilar nuestra aplicación debemos asegurarnos de que hemos definido un número de versión para nuestra aplicación, especificando un valor apropiado tanto para el atributo `android:versionCode` como para `android:versionName` de nuestra etiqueta `<manifest>` en nuestro archivo “manifest.xml” de la aplicación.

Tenemos que considerar cuidadosamente nuestro versionamiento siguiendo algún plan de numeración derivado de nuestra estrategia global de actualizaciones para la aplicación.

Si ya hemos lanzado una versión de nuestra aplicación debemos asegurarnos de que se aumente el número de versión de la aplicación actual. Debemos incrementar tanto el atributo `android:versionCode` como el atributo `android:versionName` de la etiqueta `<manifest>` en el archivo “manifest.xml” de la aplicación, usando los valores apropiados como hemos descrito mas detalladamente en puntos anteriores de este proyecto fin de carrera que tratan sobre este mismo tema.

2. **Obtener una clave criptográfica adecuada.**

Si hemos leído y seguido todos los pasos de preparación hasta el momento, nuestra aplicación se debería haber compilado y estar lista para ser firmada. Dentro del paquete .APK, la aplicación está correctamente versionada, y hemos limpiado los archivos innecesarios y los datos privados, como se describió anteriormente.

Antes de firmar nuestra aplicación, debemos asegurarnos de que tenemos una clave privada adecuada. Una vez que hayamos obtenido (o generado) dicha clave privada, la vamos a usar para:

- Registrarnos para obtener una clave API de Google Maps, si la aplicación utiliza elementos “MapView”.
- Firmar nuestra aplicación para el lanzamiento después en el proceso de preparación.

3. **Registrarnos para obtener una clave API de Google Maps, si nuestra aplicación utiliza elementos “MapView”.**

Si nuestra aplicación utiliza uno o más elementos “Maps view”, tenemos que registrar la aplicación en el servicio “Google Maps”, aparte tenemos que obtener una clave API Google Maps para que nuestros MapView´s sean capaces de recuperar datos de Google Maps. Para ello, se nos proporcionará una firma digital MD5 de nuestro certificado firmante con el servicio Google Maps.

Durante el desarrollo podemos obtener una clave temporal API Google Maps mediante el registro de la clave de depuración generada por las herramientas del SDK Android. Sin embargo, antes de publicar nuestra aplicación, debemos registrarnos para obtener una nueva clave API Google Maps basada en nuestra clave privada.

Si la aplicación utiliza elementos “MapView”:

1. Debemos obtener la clave de API de Google Maps antes de compilar la aplicación para el lanzamiento, ya que debemos agregar la clave a un atributo especial para cada elemento “MapView” (atributo `android:apiKey`) en los archivos de diseño de la aplicación. Si por el contrario instanciamos objetos de tipo “MapView” directamente desde el código fuente, debemos pasar la clave del API de Google Maps como un parámetro en el constructor de dicha clase.

2. La clave de API de Google Maps que referencia los elementos MapView en nuestra aplicación debe estar registrada (por Google Maps) en el certificado usado para firmar la aplicación. Esto último es importante cuando publiquemos la aplicación ya que los elementos MapView deben hacer referencia a una clave que esté registrada en el certificado de lanzamiento que se utilizará para firmar nuestra aplicación.

3. Si tenemos una clave temporal API Google Maps por haber registrado el certificado de depuración generado por las herramientas del SDK Android, *debemos* recordar obtener una nueva clave API Google Maps mediante el registro de nuestro certificado de lanzamiento. A continuación, debemos cambiar la referencia de los elementos MapView utilizados a la nueva clave obtenida, en lugar de la antigua clave asociada con el certificado de depuración. Si no lo hacemos así, los elementos MapView no tendrán permiso para la descarga de datos “Maps”.

4. Si cambiamos la clave privada que vamos a utilizar para firmar la aplicación, *debemos* obtener una nueva clave API Google Maps del servicio Google Maps. Si no recibimos la nueva clave API y la aplicamos a todos los elementos “MapView”, todos los elementos estarán referenciados a la clave vieja y no tendrán permiso para descargar datos “Maps”.

7.3 Compilado final de la aplicación

Una vez que hayamos preparado nuestra aplicación como se describe en las secciones anteriores, podemos compilarla para el lanzamiento de la misma.

7.4 Después de compilar la aplicación

1. **Firma de la aplicación.**

Firmaremos nuestra aplicación utilizando la clave privada y luego la alinearemos con la herramienta “zipalign”. La firma de la aplicación correctamente es muy importante para la posterior distribución de la misma.

2. **Poner a prueba la aplicación firmada y compilada.**

Antes del lanzamiento de la aplicación compilada, debemos probarla en el dispositivo móvil elegido (con la red disponible, si es posible). En particular, debemos asegurarnos de que todos los elementos “MapView” de la interfaz de usuario están recibiendo datos Maps correctamente. Si no es así, habrá que volver a registrar la clave API de Google Maps y corregir el problema.

También debemos asegurarnos de que la aplicación funciona correctamente con todos los servicios del servidor, con los datos que se proporcionan o se basan en la aplicación, y que se manejan todos los requisitos de autenticación correctamente.

Después de estas pruebas estamos listos para publicar las aplicaciones a los usuarios de los dispositivos móviles, para ello nos ayudaremos de los servicios que nos ofrece Google Play de Google.

7.5 Publicación en Google Play

Si hemos seguido los pasos descritos en los apartados anteriores de éste capítulo, el resultado del proceso de compilado, firmado y alineado es un paquete .APK, archivo firmado con su clave privada de lanzamiento. Nuestra aplicación estaría lista para ser divulgada a los usuarios y que estos puedan instalarla en sus dispositivos móviles.

Podemos publicar la aplicación y permitir a los usuarios instalarla de la forma que elijan, incluyendo desde nuestro propio servidor web.

A continuación proporcionaremos información sobre la publicación de nuestra aplicación mediante el servicio Google Play de Google que tratamos en este proyecto.

7.5.1 Acerca de Google Play

Google Play es un servicio que facilita a los usuarios encontrar y descargar aplicaciones Android para sus dispositivos móviles bajo la misma plataforma, sea desde la aplicación Google Play en el dispositivo o desde el sitio web de Google Play (market.android.com). Como desarrolladores, podemos usar Google Play para distribuir nuestras aplicaciones a los usuarios de todo tipo de dispositivos Android en todo el mundo.

Para la publicación de aplicaciones con Google Play, primero debemos registrarnos en dicho servicio con una cuenta de Google y aceptar los términos del servicio. Una vez registrados, podemos subir nuestras aplicaciones al servicio siempre que lo deseemos, actualizarlas tantas veces como se desee, y posteriormente publicarlas cuando estén listas. Una vez publicadas, los usuarios pueden ver nuestras aplicaciones, descargarlas, y puntuarlas según su grado de satisfacción.

Para registrarnos como desarrolladores en Google Play y poder realizar publicaciones, visitaremos la Web de publicación de Google Play:

“*http://market.android.com/publish*”

Para publicar nuestras aplicaciones en Google Play debemos asegurarnos de que cumplen con los siguientes requisitos, estos nos lo impone el servidor Market al cargar o subir la aplicación.

Requisitos impuestos por el servidor Google Play:

1. Nuestra aplicación debe estar firmada con una clave criptográfica privada cuyo período de validez termina después del día 22 de octubre 2033.

2. La aplicación debe definir tanto un atributo `android:versionCode` como uno `android:versionName` atributo en la etiqueta `<manifest>` de nuestro archivo “manifest.xml”. El servidor utilizará el primero como base para la identificación interna de la aplicación y manejo de las actualizaciones, y muestra el segundo a los usuarios como la versión de la aplicación.

3. Por último nuestra aplicación debe definir tanto un atributo `android:icon` como un `android:label` en la etiqueta `<application>` de nuestro archivo “manifest.xml”.

7.5.2 Publicación de actualizaciones en Google Play

En cualquier momento posterior a la publicación de una de nuestras aplicaciones en Google Play podemos cargar y publicar una actualización del paquete de la misma aplicación. Cuando publicamos una actualización de una aplicación, los usuarios que ya han instalado dicha aplicación pueden recibir una notificación de que hay una nueva actualización disponible para ella. A continuación, pueden optar por actualizarla a la última versión.

Antes de cargar la aplicación actualizada, debemos asegurarnos de que hemos incrementado el atributo `android:versionCode` y el `android:versionName` en la etiqueta <`manifest`> del archivo “manifest.xml”.

Además, el nombre del paquete debe ser el mismo que el de la versión existente y el archivo .APK debe estar firmado con la misma clave criptográfica privada. Si el nombre del paquete o el certificado de la firma *no coinciden* con los de la versión actual, el servicio Google Play va a considerarlo como una nueva aplicación y publicarlo como tal, por lo tanto, no lo ofrecerá a los usuarios como una actualización de la aplicación.

7.5.3 Utilización del servicio de licencias Google Play

Google Play nos ofrece un servicio de licencias que nos permite cumplir con las políticas de concesión de licencias para las aplicaciones de pago que se publican a través de Google Play.

Con la licencia de Google Play, nuestras aplicaciones pueden consultar el servicio en tiempo de ejecución y obtener el estado de la licencia para el usuario actual, y luego permitir o no el uso adicional de la aplicación, según corresponda. Con este servicio, podemos aplicar una política flexible de licencias aplicación por aplicación, cada una de ellas puede imponer su estado de licencia de la manera más apropiada para ellas.

Cualquier aplicación publicada a través de Google Play puede utilizar el servicio de licencias del mismo. Este servicio no utiliza ningún framework API en específico, por lo que se pueden agregar licencias a cualquier aplicación que utilice un nivel API mínimo de 3 o superior.

7.5.4 Vinculación de nuestras aplicaciones en Google Play

Para ayudar a los usuarios a descubrir las aplicaciones publicadas, se pueden utilizar dos URI´s especiales Google Play que dirigen al usuario a la página de detalles de nuestra aplicación o realizar una búsqueda de todas las aplicaciones publicadas en Google Play. Podemos utilizar estos identificadores URI para:

- Crear un botón en la aplicación o un enlace en una página web que nos redirija a la página de detalles de nuestra aplicación en la propia aplicación Google Play o en su sitio web.
- Crear un botón en la aplicación o un enlace en una página web que busca todas las aplicaciones publicadas en la aplicación Google Play o en el sitio web.

Podemos iniciar la aplicación Google Play o en el sitio web de la siguiente manera:

- Iniciar un `Intent` en nuestra aplicación que ejecute la aplicación Google Play en el dispositivo del usuario. La intención debe utilizar el `Action` ACTION_VIEW, e incluir los datos en el Intent con el esquema Google Play URI adecuado.
- Proporcionar un hipervínculo en una página web que nos redirija al sitio web de Google Play.

En ambos casos, es necesario que creemos un URI para indicar a la aplicación lo que nos gustaría que nos mostrara en Google Play o la búsqueda que deseamos realizar. Dicha URI es muy similar si deseamos abrir la aplicación o abrir el sitio web. La única diferencia es el prefijo URI.

Para abrir la aplicación Google Play en el dispositivo, el prefijo de URI de los datos del Intent es la siguiente:

"market://"

Para redirigirnos al sitio web de Google Play, el prefijo del URI es:

"http://market.android.com/"

Para completar cada URI, debemos añadir una cadena que especifica la aplicación para la que se desea ver o la búsqueda a realizar. Los puntos siguientes de este capítulo describen cómo crear un URI completo para cada caso.

Si creamos un vínculo para que nos redirija al sitio web de Google Play y el usuario lo selecciona desde un dispositivo Android, la aplicación Google Play se activará para resolver el evento, el usuario utilizará la aplicación nativa en lugar del sitio web. Además, debido a que la aplicación Google Play lee URI´s "http://", también podemos utilizarlos en un Intent, pero por lo general usaremos URI´s del tipo

“market://” para este mismo propósito, de modo que la aplicación nativa se ejecutará por defecto. Debemos usar URI´s del tipo “http://” sólo al crear enlaces en una página web.

7.5.4.1 Abrir la página de detalles de la aplicación

Como describimos anteriormente, podemos cargar la página de detalles de una aplicación específica, ya sea en la propia aplicación Google Play o su sitio web. La página de detalles permite al usuario ver la descripción de la aplicación, capturas de pantalla, críticas y demás, también le da la opción para instalar la aplicación.

El formato del URI que utilizaremos para cargar dicha página de detalles es la siguiente:

```
<URI_prefix>details?id=<package_name>
```

En el identificador <package_name> indicaremos el nombre del paquete completo de la aplicación destino, según lo declarado en el atributo de paquete de la etiqueta <manifest> del archivo “manifest.xml” de la aplicación.

7.5.4.1.1 Abrir los detalles en la aplicación Google Play

Para abrir la página de detalles en la aplicación Google Play, creamos un Intent con la Action ACTION_VIEW e incluimos unos datos URI con el siguiente formato:

```
market://details?id=<package_name>
```

Como ejemplo, mostramos el código de cómo se puede crear un Intent para cargar la página de detalles de una aplicación en la aplicación Google Play:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("market://details?id=com.android.example");
startActivity(intent);
```

7.5.4.1.2 Abrir los detalles en la web oficial Google Play

Para cargar la página de detalles en el sitio web Google Play, creamos un enlace con un URI con el siguiente formato:

```
http://market.android.com/details?id=<package_name>
```

Por ejemplo, mostramos el código para crear un enlace que cargue la página de detalles de una aplicación en el sitio web Google Play:

```
<a href="http://market.android.com/details?id=com.android.example">App
Link</a>
```

7.5.4.2 Realizar una búsqueda

Para iniciar una búsqueda en Google Play, el formato del URI es el siguiente:

```
<URI_prefix>search?q=<query>
```

El identificador <query> nos indica la consulta de búsqueda para ejecutar en Google Play. Dicha consulta puede ser una cadena de texto plano o podemos incluir un parámetro para realizar una búsqueda basada en el nombre de publicación:

- Para realizar una búsqueda de texto plano, añadiremos la cadena a la consulta:

  ```
  <URI_prefix>search?q=<search_query>
  ```

- Para realizar una búsqueda basada en el nombre de publicación, utilizaremos el parámetro “pub:” de la consulta, seguido por el nombre de publicación:

  ```
  <URI_prefix>search?q=pub:<publisher_name>
  ```

  Podemos utilizar este tipo de búsqueda para mostrar todas nuestras aplicaciones publicadas.

7.5.4.2.1 Buscar en la aplicación Google Play

Para realizar una búsqueda en la aplicación Google Play, creamos un Intent con la Action ACTION_VIEW e incluimos unos datos URI en este formato:

```
market://search?q=<query>
```

La consulta puede incluir el parámetro “pub:” anteriormente descrito.

Por ejemplo, mostramos el código para iniciar una búsqueda en la aplicación Google Play, basada en el nombre de publicación:

```
Intent intent = new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("market://search?q=pub:Your Publisher
Name"));
startActivity(intent);
```

El resultado de la búsqueda muestra todas las aplicaciones publicadas por el mismo desarrollador y las cuales son compatibles con el dispositivo actual.

7.5.4.2.2 Buscar en la web oficial Google Play

Para realizar una búsqueda en el sitio web de Google Play, creamos un enlace con un URI con el siguiente formato:

```
http://market.android.com/search?q=<query>
```

La consulta puede incluir el parámetro “pub:” anteriormente descrito.

Por ejemplo, mostramos el código para crear un enlace que inicie una búsqueda en el sitio web AndroidMarket, basada en el nombre de publicación:

```
<a href="http://market.android.com/search?q=pub:Your Publisher
Name">Search Link</a>
```

El resultado de la búsqueda muestra todas las aplicaciones publicadas por el mismo desarrollador.

7.5.4.3 Sumario de formatos URI

La siguiente tabla presenta un resumen de los URI actualmente soportados por Google Play (tanto en la Web como en la aplicación Android), como se mencionó en apartados anteriores.

Para estos resultados	**URI utilizados en un enlace de página web**	**URI utilizados en el ACTION_VIEW del Intent**
Mostrar la pantalla de detalles de una aplicación específica	`http://market.android.com/details?id=<package_name>`	`market://details?id=<package_name>`
Búsqueda de aplicaciones usando una cadena general	`http://market.android.com/search?q=<query>`	`market://search?q=<query>`
Búsqueda de aplicaciones por el nombre de publicación	`http://market.android.com/search?q=pub:<publisher_name>`	`market://search?q=pub:<publisher_name>`

Tabla 4. Resumen URI.

Capítulo 8

Ataques a la plataforma Android

8 Ataques sobre la plataforma

Para la realización de este capítulo de nuestro proyecto fin de carrera, hemos recopilado, estudiado y ampliado la información de las siguientes referencias [16], [17], [18], [27], [28], [29], [30], [31] y [32].

En este capítulo vamos a tratar tanto el panorama actual de la plataforma desde el punto de vista de los ataques producidos a la misma, como el comienzo de estos mismos en los siguientes puntos.

Si bien los sistemas de seguridad son capaces de bloquear los ataques de virus informáticos sobre los dispositivos móviles, la industria móvil ve nuevos riesgos en las plataformas hechas con código abierto, como el sistema Android de Google.

Desde el año 2004, hay virus que son capaces de desactivar teléfonos o aumentar las facturas de su dueño con mensajes costosos y llamadas involuntarias, que llevaron a la creación de un nuevo mercado para la tecnología de la seguridad.

Si Android se convierte en una plataforma totalmente abierta su uso se hará cada vez más común, y los riesgos serán mayores que los que existen con las actuales plataformas, como Symbian o IOS, esto hoy en día es una realidad.

Al conocido error que envía mensajes mensajes SMS a diferentes destinatarios del inicialmente elegido por el usuario, se le suma una vulnerabilidad existente que permite, de forma similar al troyano conocido como “Geinimi”, sustraer información o archivos debido a un fallo de seguridad en el navegador de Internet nativo.

Para colmo de males, ambos problemas son conocidos desde años anteriores por la comunidad de desarrolladores y los ingenieros de Google encargados de la evolución de la plataforma Android, sin que por ello se le haya dado una solución, sobreviviendo a la última versión aparecida de la plataforma, el “Android 2.3 Gingerbread”.

La vulnerabilidad en cuestión, posibilita el envío de determinados datos al autor de un hipotético ataque realizado a través de un archivo JavaScript descargado a través del navegador de Internet de Android sin que éste pida autorización.

El código malicioso se ejecuta en el mismo contexto del propio explorador, haciéndose pasar por un proceso de usuario, lo que le posibilita pasar inadvertido a la hora de acceder a los diferentes archivos y espacios de almacenamiento a los que el propio navegador tiene permiso, encontrándose la tarjeta de memoria entre ellos.

Lugar en donde habitualmente se almacenan archivos de configuración, datos del usuario o incluso historiales, pudiendo llegar a ofrecer información sobre sus hábitos de navegación.

Se dio a conocer el fallo de seguridad de la plataforma y ya se habla de una solución definitiva para este y otros errores disponible en la versión 2.3.1 del sistema operativo. Lo cual está aún por ver tras la reciente liberación de “Android 2.3 Gingerbread”.

Lo que también, por otro lado, pone de manifiesto otro de los problemas a los que se enfrenta Google con el tan cacareado problema de la fragmentación que sufre su sistema operativo para dispositivos móviles, ya que los usuarios deberán esperar a que los de Mountain View, fabricantes y compañías operadoras se pongan de acuerdo para facilitar las actualizaciones del sistema operativo. Eso siempre y cuando nuestro terminal sea actualizable.

Por el momento, las recomendaciones facilitadas por los expertos pasan por deshabilitar la ejecución de código “JavaScript” en el navegador del sistema o utilizar programas de terceros para explorar la Red.

Como podemos observar la rápida evolución de los dispositivos móviles y el éxito que estos últimos están teniendo en el mercado de la telefonía, está promoviendo cada vez más un caldo de cultivo idóneo para la aparición de malware y ataques a esta plataforma, cuyo éxito se debe en parte por su carácter de software libre y su versatilidad de desarrollo.

A continuación, ahondaremos punto por punto en este capítulo en la historia contemporánea que nos proporciona este panorama, desde los comienzos de las primeras aplicaciones de carácter malicioso como las más “famosas” y a la vez complejas, metiéndonos un poco en profundidad en estas últimas.

Geinimi

Geinimi, así se llama el nuevo código malicioso para sistemas Android que se está propagando entre usuarios de China. Aunque el malware para esta y otras plataformas móviles no es ninguna novedad, esta amenaza en concreto es la más sofisticada que se ha visto hasta el momento en sistemas Android y no se conforma solo con enviar información confidencial del usuario, sino que también puede instar al usuario a instalar o desinstalar aplicaciones de su dispositivo móvil e incluso se sospecha que el terminal infectado pueda entrar a formar parte de una “botnet”.

Los descubridores de esta nueva amenaza han realizado un extenso análisis de la misma, descubriendo así que, cada cinco minutos, este troyano intenta conectarse a un servidor remoto con diferentes nombres de dominio que ya vienen incluidos en el código de la amenaza. Si la conexión se realiza con éxito, el malware envía toda la información que ha ido recopilando al servidor remoto.

Las investigaciones realizadas hasta el momento parecen confirmar que los usuarios que se han visto afectados por esta amenaza se descargaron aplicaciones infectadas desde mercados de aplicaciones de terceros presentes en China. En ningún caso las aplicaciones fueron descargadas desde el Andriod Market oficial, aunque las aplicaciones presentes en ese repositorio tampoco están exentas de resultar dañinas. Está técnica, ya vista con anterioridad en otros sistemas operativos para móviles como “Windows Mobile” puede resultar altamente efectiva si las aplicaciones infectadas son populares y se descargan desde sitios no oficiales para evitar pagar por ellas.

Para evitar que nuestro dispositivo móvil se infecte, recomendamos prestar atención ante toda aplicación que descarguemos. Es importante revisar si se realizan conexiones no autorizadas o se piden permisos excesivos. Las llamadas y envío de SMS ocultos a números extraños o la aparición de nuevas aplicaciones que no hemos instalado nosotros también pueden ser un síntoma de que nuestro dispositivo ha sido infectado.

8.1 El primer troyano para Android

En este primer punto vamos a destacar el comienzo de software malicioso conocido para la plataforma Android.

Los dispositivos Android de Google recibieron el primer ataque de su primer troyano, se llamaba “Trojan-SMS.AndroidOS.FakePlayer.a” el cuál se propaga por SMS, se hace pasar por una aplicación de medios inofensiva y una vez instalada

realmente envía mensajes de texto por SMS sin conocimiento ni aprobación por parte de los usuarios.

Este troyano penetra en dispositivos Android haciéndose pasar por una aplicación normal. A los usuarios se les solicita que instalen un pequeño archivo de unos 13 KB que tiene la extensión estándar de Android, .APK. Sin embargo, una vez que la “aplicación” se instala en el dispositivo, el troyano que va incluido en ella comienza a enviar mensajes a números de teléfono de pago. Los delincuentes son los que llevan estos números en realidad, por lo que terminan cobrando el dinero por medio de cobros en la cuenta de las víctimas.

La categoría de troyano de SMS del malware es relativamente habitual en el ecosistema de dispositivos móviles, pero éste es el primero que se dirige específicamente a dispositivos bajo la plataforma Android. Sin embargo, “FakePlayer” no es el primer malware diseñado para dicha plataforma, ya que ha habido incidentes aislados de infecciones de dispositivos Android con spyware, el primero de los cuales se produjo en 2009.

La decisión de dirigirlo contra dispositivos Android en particular no debería sorprender a quienes estén al tanto de las tendencias de la industria de los dispositivos móviles. Teniendo en cuenta el ascenso meteórico de Android en cuota de mercado.

“La organización de estudios y análisis del mercado de la informática IDC ha señalado que los vendedores de dispositivos con Android son los que están experimentando el mayor crecimiento en ventas entre los fabricantes de smartphones”. Como resultado de esto, esperamos ver un aumento equivalente en la cantidad de malware dirigido a esta plataforma.

Según una declaración de Google, el proceso de instalación está diseñado para proteger a los usuarios de ataques como éstos, ya que menciona la información y los recursos del sistema a los que tiene permiso para acceder la aplicación, como, por ejemplo, enviar un SMS. “Los usuarios deben aprobar explícitamente este acceso para poder continuar con la instalación”, afirma la declaración. “Aconsejamos siempre a los usuarios que sólo instalen aplicaciones de confianza. En particular, los usuarios deberían tener cuidado al instalar aplicaciones ajenas al Google Play.”

Sin embargo, el lanzamiento de un troyano disfrazado de aplicación es un modo creativo de llevar el malware a los dispositivos móviles. En este caso, el troyano aprovecha la falta de atención de los usuarios en el proceso de instalación, así como la apertura de Android, ya que este sistema operativo no está asociado a una tienda de aplicaciones administrada y “publicada” bajo controles estrictos, como sucede en “iPhone” con su “iTunes”. Aunque Google sí se pone en funcionamiento para retirar aplicaciones del Market cuando se presentan peligros para la seguridad, no hay nada que evite a los desarrolladores saltarse los canales oficiales y publicar sus propias aplicaciones en otra parte y engañar a los usuarios para que las instalen.

Sin embargo, incluso si el troyano procediese de canales secundarios, como mínimo resulta un pequeño golpe para un sistema operativo que resalta la seguridad en su diseño.

8.2 El Aumento de los ataques sobre Android

La popularidad de la plataforma de Google y el progresivo aumento de los dispositivos bajo dicha plataforma ha hecho que los ataques a Android se estén incrementando rápidamente. Advertimos de la propagación de “Pjaaps” (aplicaciones maliciosas para nuestros dispositivos móviles), un ataque que ha avanzado al siguiente nivel y que se disfraza “bajo una aplicación popular y legítima para Android”.

Podemos asegurar que los ataques a la plataforma Android se están incrementando de manera considerable. Es más, podemos advertir de una “tercera evolución de un rápido incremento de malware dirigido a la plataforma de Google, Android”.

Así, y días después de que se informara sobre la aparición de las “Pjapps”, los ataques han avanzado al siguiente nivel con aplicaciones que han sido descubiertas en la página oficial de Android, además debemos de advertir de que la importancia de este hecho se centra en que es la primera vez que malware se encuentra en un sitio oficial. “Nunca antes se había encontrado en el mercado oficial, sino sólo en sitios de alojamiento para terceros”.

Consciente del problema, Google ya ha tomado medidas al eliminar estas aplicaciones de la tienda oficial de Android (Google Play). Además, han publicado una aplicación, denominada “herramientas de seguridad de la tienda Android”, que borra los efectos secundarios que haya podido causar el virus “Android.Rootcager”.

Recomendamos a los usuarios que, para evitar ser víctima de este tipo de ataques, se utilicen solamente:

- Vías oficiales para descargar e instalar aplicaciones Android.
- Se ajusten los parámetros del SO Android de nuestros dispositivos para detener la instalación de aplicaciones no comerciales.
- Se lean comentarios de otros usuarios sobre una u otra aplicación.
- Se verifiquen, *siempre*, los permisos de acceso que se soliciten para una instalación.
- Y por último se utilicen soluciones de seguridad para dispositivos móviles.

8.3 Google elimina aplicaciones de Google Play

A tenor de todo lo mencionado en el punto anterior sobre el incremento de malware para Android, se han cuestionado las medidas de seguridad que se aplican en Google Play. Parece ser que las medidas de control que se aplican en Google Play son escasas o inexistentes como han puesto de manifiesto un grupo de expertos en seguridad con el experimento que mencionaremos a continuación.

El experimento ha consistido en crear dos programas que contenían malware, registrar las aplicaciones y subirlas a la tienda virtual durante unos días a ver qué pasaba. Parece ser que no ha ocurrido nada, las aplicaciones han estado en Google Play varios días, incluso han sido descargadas por usuarios sin haber mediado publicidad y ningún responsable de la supuesta seguridad de Google Play se ha percatado de su existencia.

Lógicamente y por tratarse de un experimento, el software que ha burlado la seguridad era inocuo y no ha causado ningún daño a los usuarios que han instalado las aplicaciones. El problema es que, como ya se ha puesto en evidencia en más de una ocasión, los desarrolladores que han subido aplicaciones con fines ilícitos sí se están aprovechando de esta aparente falta de control.

Muchas empresas de seguridad llevan tiempo alertando de que en esta segunda década del SigloXXI, el destino del malware va a ser el terminal móvil debido a su masificación. Cabe señalar que Android es el objetivo de las amenazas, con un aumento del 400% de malware para esta plataforma debido al éxito que está obteniendo en el mercado y su carácter de software libre.

De esto podemos deducir que tanto empresas como particulares estamos expuestos a un número récord de amenazas de seguridad, incluyendo principalmente los ataques dirigidos a las redes Wi-Fi. Uno de los hallazgos más inquietantes es la fórmula para distribuir malware para móviles, la descarga de aplicaciones.

El riesgo está presente también en los mensajes SMS, cabe señalar que el 17% de todas las infecciones notificadas fueron a través de SMS, textos con reclamos para llamar a números de tarificación adicional, con el correspondiente gasto para las víctimas.

En la misma línea, también cabe destacar que “Facebook”, utilizado desde Android, ha experimentado un notable incremento de ataques de seguridad, concretamente tres veces mayor que el correspondiente al año pasado.

Windows ha sido estos años el objetivo de la mayoría de ataques de seguridad, ahora parece que va a cambiar el escenario, o al menos los esfuerzos por arruinar nuestros equipos se van a diversificar.

El malware para Android sigue creciendo, esta vez son entre 30.000 y 120.000 posibles afectados. Las noticias sobre malware destinado a vulnerar la seguridad de este sistema operativo se suceden una tras otra. Primero, saltó a la palestra el agujero de

seguridad en el navegador de Android 2.2 Froyo. A continuación, en Google Play tuvieron que eliminar varias aplicaciones infectadas que podrían haber perjudicado a 200.000 usuarios.

Después, podemos hablar del incremento de software dañino para Android, que se estimaba en un 400%. Y por último, el fallo detectado en el sistema operativo cuando opera en redes Wi-Fi abiertas. Si bien es cierto que Google pone remedio de forma casi inmediata a estos problemas, los usuarios no paran de recibir malas noticias. En esta ocasión, se han detectado hasta 26 aplicaciones en Google Play que podrían haber infectado hasta 120.000 terminales.

“Droid Dream Light”, variante del ya conocido *“Droid Dream”*, es el responsable. Se han detectado y eliminado hasta 26 aplicaciones infectadas en Google Play por este código maligno, que puede haber comprometido los datos personales de decenas de miles de usuarios.

La popularidad de Android, con una cuota global del 36% del mercado de terminales móviles, le convierte en objetivo preferido de este tipo de acciones. El problema es que este momento dulce que vive Android, puede verse truncado por noticias de esta índole y Google ha de adoptar una posición más vigilante, la competencia viene muy fuerte. Windows Phone e iOS podrían aprovechar estas circunstancias para hacerse con una buena parte del mercado.

8.3.1 Droid Dream

El código que llevó a Google a retirar más de 50 aplicaciones de Google Play es “*DroidDream*”, que convierte los dispositivos Android en zombis. Se trata de un programa diseñado para cargar código que afecte a la seguridad de los dispositivos móviles, según expertos en seguridad.

Así, “*DroidDream*” sacaría provecho de dos problemas para obtener privilegios en los dispositivos Android al romper el área de seguridad (sandbox) diseñada para limitar lo que puede hacer cada aplicación en dichos dispositivos móviles.

Aunque las vulnerabilidades fueron solucionadas por Google, la mayoría de los teléfonos aún no tienen dicha actualización, permitiendo que más de 260.000 teléfonos se hayan visto afectados.

Después del primer ataque, el programa reenvía determinada información del teléfono (como identificadores de hardware, software y servicios) a un servidor, que puede hacer que el teléfono se conecte a determinados momentos para descargar funciones desde una URL, según estos análisis de seguridad.

La segunda parte del ataque sería más interesante, porque es como un cheque en blanco, así, permitiría manipular las valoraciones y comentarios de Marketplace.

8.3.2 Droid Dream Light

El virus, bautizado como “*Droid Dream Light*”, habría infectado a un total de 25 aplicaciones, que a estas alturas ya han sido eliminadas de Google Play. Este software infeccioso no es más que una modificación de un malware que ya había atacado a los usuarios de Android con anterioridad, de manera que Google ya estaría al tanto de cómo dar solución (más o menos inmediata) al problema que está atravesando, y por ende, sus usuarios. Se podría hablar de varias decenas de miles de afectados.

El malware se habría infiltrado en un total de 25 aplicaciones que ya han sido detectadas y eliminadas de la tienda oficial de Google, el Google Play. En este sentido, todas las personas que hubieran descargado cualquiera de estas aplicaciones, tendrían durmiendo en su teléfono un virus que se activa cada vez que el teléfono registra un movimiento, esto es, una llamada o un mensaje. Según las estimaciones, el total de usuarios de Android afectados podría cifrarse entre 30.000 y 120.000 usuarios. Para dar solución a esta incidencia, los teléfonos comprometidos con esta incidencia tendrán que efectuar un borrado remoto, procedimiento que garantizaría la eliminación completa del virus.

Según los expertos, este virus es más peligroso que su antecesor, puesto que es capaz de ejecutarse de forma automática, sin necesidad de que el usuario haga nada para activarlo. Sabemos, sin embargo, que todas estas aplicaciones han sido eliminadas y que ya no están disponibles para descargar.

8.4 Google mejora la seguridad de Google Play

Como hemos mencionado en puntos anteriores una de las formas de atacar la plataforma Android ha sido mediante el servicio Google Play de Google, al tratarse de un servicio para la libre distribución de aplicaciones resulta idóneo para la distribución de malware o aplicaciones maliciosas, ya sea mediante aplicaciones en principio legales que han sido modificadas al obtener si clave de cifrado o aplicaciones que de por sí ya tenían código malicioso en un principio.

Google ha decidido mejorar la seguridad de Google Play después de descubrirse más de 50 aplicaciones peligrosas. Aseguran haber identificado y retirado estos programas, que sacaban partido de vulnerabilidades que no afectan a Android 2.2.2 y versiones superiores.

Además, Google está retirando de manera remota las aplicaciones instaladas en los teléfonos móviles. Para ello, utiliza una “kill switch”, denominada “Google Play Security Tool March 2011”. Los usuarios afectados recibirán un mensaje de correo electrónico del remitente “android-market-support@google.com” en un plazo máximo de 72 horas notificándoles que se les ha instalado la mencionada aplicación.

Con esta utilidad, se eliminarán los programas malignos y se evitará que los atacantes puedan acceder a información y datos personales de los usuarios.

En muchos casos, estas aplicaciones malignas eran versiones clonadas de programas legítimos que habían sido modificados para incluir malware. Según la firma de seguridad móvil “Lookout”, se habrían llegado a descargar entre 50.000 y 200.000 copias de las aplicaciones.

Es la segunda vez que Google lleva a cabo una operación similar. En enero de 2010, se retiraron un par de aplicaciones que fueron publicadas por un investigador de seguridad para tergiversar de manera intencionada su objetivo para animar a los usuarios a descargarlas. Sin embargo, las aplicaciones no estaban diseñadas para un uso malicioso y no pedían permiso para acceder a datos personales.

Como hemos mencionado anteriormente, además de retirar estas aplicaciones, Google está promocionando una utilidad propia, “Google Play Security Tool March 2011”, para los teléfonos Android que se hayan visto afectados. Esta aplicación se instalará de manera automática en todos aquellos terminales que hayan descargado una aplicación maligna y con ella se evitará que los atacantes puedan obtener más información de los usuarios.

Además, Google añade que se están tomando diversas medidas para evitar que en el futuro se puedan descargar desde Google Play aplicaciones malignas que también saquen partido de similares errores.

Tras el desastre de Google Play y la difusión de virus a gran escala que afectaba a más de 50 aplicaciones, el equipo de Android ha puesto las cartas sobre la mesa.

Desde el blog de dispositivos móviles de Google se nos informa de que dicho “percance” no volverá a repetirse y ha tomado una serie de medidas al respecto:

- Se han eliminado las aplicaciones maliciosas del Market, además de la suspensión de las cuentas de los desarrolladores de dichas aplicaciones y el aviso inmediato a las autoridades.
- Están eliminándose de manera remota las aplicaciones de los dispositivos infectados. Dicha tarea la realizan mediante el control de eliminación de aplicaciones a distancia, uno de los controles de seguridad del equipo de Android para proteger a los dispositivos de ataques maliciosos como el que se ha sufrido.
- Difusión de actualizaciones de seguridad para Google Play con el fin de acabar con los fallos de seguridad que generan problemas de este tipo. Además, están desarrollando actualizaciones que evitarán situaciones similares en el futuro.

Según el equipo de Android la única información que pueden haber sustraído de los terminales es el IMEI y el IMSI (códigos que sirven para identificar el tipo de móvil)

además de la versión del Market usada. Si nuestro dispositivo ha sido uno de los infectados no debes preocuparte ya que el propio equipo solucionará el problema y se te avisará de que este ha sido solucionado por medio de correo electrónico.

La seguridad es una prioridad para la plataforma Android, y están trabajando para desarrollar nuevas barreras de seguridad que ayuden a evitar y prevenir estos tipos de ataques en el futuro.

8.5 Consejos para evitar malware en Android

La lista de aplicaciones malignas en Android crece. He aquí una serie de consejos para evitar el malware en nuestro dispositivo móvil Android y los nombres de algunas de las aplicaciones retiradas de Google Play.

Además de tener en cuenta consejos básicos de seguridad que pueden (y deben) aplicarse a cualquier otro producto tecnológico, ofreceremos cinco consejos. Cinco medidas con las que poder evitar que el malware nos afecte a nuestro dispositivo.

Buscar y rebuscar información sobre el autor de la aplicación. ¿Qué otras aplicaciones ofrece? ¿Parece sospechosa? Si es así, quizá deberíamos evitarla.

1. Lee las críticas y las reseñas. Las que puedes encontrar en Google Play pueden no ser sinceras, por lo que recomendamos hacer una búsqueda más global, especialmente en páginas web de prestigio, para saber qué se dice de una aplicación antes de descargarla e instalarla.

2. Consultar siempre los permisos de la aplicación. Tanto si instalas como si actualizas una aplicación, estás dando una serie de permisos a dicho programa. Si tienes una utilidad despertador, no tiene porqué ver todos tus contactos. Desconfía de aquellas aplicaciones que piden permisos para más cosas de las que necesiten.

3. Evita instalar archivos Android Package (.APK) que sólo puedes disponer a través de un tercero. Es lo conoce como “sideloading” o instalar aplicaciones utilizando un archivo .APK. Se recomienda no instalar este tipo de archivos, puesto que en muchos casos no se sabe qué contienen hasta que no se instala… y para entonces puede ser demasiado tarde.

4. Instala un escáner antivirus y/o antimalware en tu dispositivo móvil. Muchos usuarios siguen pensando que estas herramientas no son útiles en los teléfonos, pero puede que la proliferación de programas malignos nos haga cambiar de opinión. Muchas empresas de seguridad y de prestigio en el mundo PC tienen ya soluciones para estos dispositivos, algunas de ellas incluso gratuitas.

8.6 Herramientas de seguridad para la plataforma Android

En este punto vamos a tratar herramientas de seguridad implementadas para la plataforma Android, vamos a empezar hablando sobre uno de los últimos avances en lo que a seguridad se trata para la plataforma de Google.

La División de Seguridad de EMC amplía las opciones de los autenticadores RSA SecurID® para las aplicaciones y los dispositivos móviles habilitados para Android™ como un componente conveniente de la seguridad empresarial.

RSA, la División de Seguridad de EMC (NYSE: EMC) anunció la disponibilidad de "RSA SecurID®Software Token for Android™" que está diseñado para permitir el uso del dispositivo basado en Android como un autenticador RSA SecurID y, así, ofrecer una autenticación conveniente y rentable de dos factores para las aplicaciones empresariales y los recursos.

Asimismo, RSA está lanzando un nuevo kit de desarrollo de software (SDK) para la plataforma Android que está diseñado para permitir a los desarrolladores incorporar la autenticación RSA SecurID de dos factores directamente a las aplicaciones de Android y obtener una ventaja competitiva al ofrecer esta capa adicional de seguridad. Las aplicaciones móviles que integran directamente la tecnología RSA SecurID ofrecen a las organizaciones la garantía de que sus recursos están diseñados con protección contra el acceso no autorizado sin ningún impacto en la capacidad de uso del usuario final. El SDK está disponible sin cargo para todos los partners de RSA Secured®.

Poder ofrecer los tokens RSA SecurID a nuestros usuarios en la mayoría de las plataformas móviles más populares, como Android, es una manera conveniente y rentable de implementar la autenticación sólida en nuestra empresa, La implementación en las plataformas de teléfonos inteligentes se realiza de forma electrónica, de modo que el aprovisionamiento es simple y rápido para nuestra organización de TI, lo que elimina los retrasos si un usuario final necesita obtener un nuevo token.

El nuevo "RSA SecurID Software Token for Android" está diseñado para generar una contraseña única que cambia cada 60 segundos, lo que permite un acceso seguro a los recursos corporativos. La solución complementa la amplia gama de métodos de autenticación ofrecidos por RSA, y ofrece a los clientes una opción en los métodos de autenticación basada en riesgo, costo y conveniencia.

"RSA SecurID Software Token for Android" está diseñado para los usuarios empresariales cuyas organizaciones hayan implementado el sistema RSA SecurID. El token puede instalarse directamente en los dispositivos habilitados para Android sin costo mediante una simple descarga desde Google Play™. Con una intervención mínima del

departamento de TI, los usuarios pueden habilitar la aplicación con un registro de origen de token único y, así, crear un autenticador RSA SecurID conveniente, seguro y rentable.

El teléfono inteligente está revolucionando el modo en que los consumidores y las organizaciones comerciales realizan los negocios. El teléfono inteligente se convertirá en el autenticador sólido predeterminado de los usuarios en el corto plazo; esto significa que el usuario deberá llevar un dispositivo menos. Es importante que los métodos de autenticación sólidos, como los dispositivos de contraseña única, sean compatibles con los teléfonos inteligentes y que los desarrolladores tengan facilidad para incorporar este método de autenticación de gran calidad a las aplicaciones móviles.

El uso de tokens de software RSA SecurID ayuda a reducir el costo total de propiedad de las organizaciones, ya que no se requiere el envío físico y pueden revocarse y volver a implementarse automáticamente si un empleado abandona la empresa con su dispositivo habilitado Android, lo cual elimina la necesidad de reemplazar los tokens.

Asimismo, disponer del autenticador de software en teléfonos inteligentes importantes para el negocio reduce la cantidad de llamadas costosas al soporte técnico por tokens perdidos.

No es ningún secreto que la informática móvil ha aumentado de manera vertiginosa recientemente, y no esperamos que la tendencia disminuya en absoluto. Este enorme crecimiento y proliferación nos ofrece una gran oportunidad de aprovechar los dispositivos como autenticadores y permitir nuevas formas de autenticación para que nuestros clientes establezcan una identidad.

El aprovechamiento de los dispositivos móviles que se ejecutan en la plataforma Android para implementar la tecnología RSA SecurID permite a los clientes ofrecer la autenticación de dos factores ininterrumpida en la nube o en las aplicaciones de las instalaciones.

El token de software RSA SecurID para la plataforma móvil Android está disponible para su descarga gratuita desde el 22 de diciembre de 2010 en Google Play. Está habilitado para los usuarios con un registro de origen de token de software único comprado por organizaciones de TI que hayan implementado RSA® Authentication Manager.

8.6.1 Acerca de RSA SecurID

Con un legado de innovación de más de 25 años, la tecnología RSA SecurID® es uno de los sistemas de autenticación de dos factores que son líderes del mercado, usado por más de 40 millones de personas en más de 30,000 organizaciones de todo el mundo.

La tecnología RSA SecurID está compuesta por una amplia gama de autenticadores de hardware y software diseñados para ayudar a prevenir el acceso no autorizado a las aplicaciones y los recursos corporativos.

Los tokens RSA SecurID están diseñados para ofrecer a los usuarios finales una contraseña única diseñada para cambiar cada 60 segundos, así como su motor de software, RSA® Authentication Manager, que logra brindar soporte a millones de usuarios y se integra con más de 350 productos.

Los autenticadores RSA SecurID incluyen tokens de hardware, autenticador híbrido con chip inteligente, token de SMS según demanda, tokens de software para teléfonos inteligentes que incluyen los dispositivos BlackBerry®, Apple iPhone®, Java ME y dispositivos con la plataforma móvil Google AndroidTM, plataformas de software para escritorios Microsoft® Windows® y Mac OS X y un token de barra de herramientas del navegador de Internet.

A continuación describiremos las principales herramientas de software libre disponibles para nuestros dispositivos móviles.

8.6.2 Antivirus Free

Aplicación rápida y ligera que protegerá tu dispositivo móvil Android de aplicaciones maliciosas. Este Antivirus detectará nuevas aplicaciones que se instalen en su terminal Android y realizará una búsqueda en bases de datos de aplicaciones maliciosas para comprobar si la aplicación es legítima, impidiendo de esta forma instalalar apps dañinas.

Más fácil de usar que AVG antivirus, más rápido que LookOut y muy ligero (utiliza menos del 0,1% de la batería), así se presenta la solución antivirus para Android, Antivirus Free. DroidDream, Geinimi y todos los últimos virus detectados por antivirus.

8.6.3 DroidWall

Droidwall es un front-end para el potente firewall de iptables de Linux gracias al cual podrás restringir las aplicaciones que están autorizadas a acceder a sus redes de datos (2G/3G y/o Wi-Fi).

Se trata de una solución perfecta si no dispones de un plan de datos ilimitado y quieres controlar las conexiones realizadas desde el terminal. Dispone de una interfaz sencilla desde la cual podrás marcar o desmarcar aquellas aplicaciones que se conecten a

Internet, dándonos la opción de bloquear o permitir sí queremos evitar que se conecten mediante WiFi o por la conexión de datos de nuestra operadora.

8.6.4 DropBox

Dropbox es un servicio de alojamiento de archivos en la nube que permite compartir y sincronizar ficheros entre diferentes plataformas. Gracias a su cliente los usuarios podrán sincronizar todos los ficheros que se encuentren en un directorio con el servicio web de Dropbox y cualquier otro equipo o dispositivo que presente dicho cliente. Además, existe una alternativa web para acceder a los archivos manualmente a través de su navegador.

Dropbox proporciona un servicio ideal para mantener una copia de seguridad de nuestros ficheros más importantes y disponer de los mismos en cualquier momento. Además presenta un historial de revisiones, por lo que los archivos eliminados del directorio de Dropbox pueden ser recuperados de cualquiera de los equipos sincronizados.

Dropbox utiliza el sistema de almacenamiento de Amazon S3 para almacenar los archivos y emplea SSL para sus transferencias. Además almacena los datos mediante cifrado AES 256.

8.6.5 LookOut

Herramienta gratuita para Android, Blackberry y Windows Mobile formada por un Antivirus, un Backup de datos y un sistema de búsqueda para localizar el móvil. Esta aplicación permite proteger su terminal móvil contra diversas amenazas, incluyendo la descarga de aplicaciones que puedan contener malware o las aplicaciones que violan la privacidad. Además, se actualiza automáticamente con las últimas actualizaciones de seguridad sin necesidad de complejas configuraciones.

Gracias a su sistema de Backup, puede programar copias de seguridad automáticas de sus contactos, fotos y llamadas y acceder a ellas vía web pudiendo eliminar de forma remota los datos del terminal en caso de robo o pérdida del terminal

LookOut permite también la localización del terminal mediante un mapa utilizando el GPS del propio dispositivo y llevar a cabo un seguimiento de los sucesos de la aplicación (exige previo registro).

8.6.6 NetQin Mobile Security

NetQin Mobile Security está diseñado para proteger dispositivos móviles Android contra virus y malware manteniendo el sistema funcionando a velocidad óptima.

NetQin Mobile ofrece múltiple funcionalidades adicionales, entre ellos, un backup online para guardar una copia de seguridad de nuestros contactos o una función anti-perdida para localizar el teléfono por medio del GPS.

8.6.7 RedPhone

RedPhone proporciona cifrado extremo a extremo para las llamadas telefónicas asegurando la confidencialidad de las conversaciónes. Utiliza ZRTP (extensión de Real-time Transport Protocol), un estándar abierto desarrollado por Philip Zimmermann, para configurar un stream SRTP (Secure Real-time Transport Protocol) entre dos dispositivos.

Puede utilizarse mediante una conexión Wifi o 3G y actualmente está disponible para dispositivos Android 1.6 y superiores.

8.6.8 TextSecure

TextSecure se presenta como un reemplazo a la mensajería de texto estándar, permitiendo enviar y recibir mensajes de texto de forma segura. Todos los mensajes de texto enviados o recibidos con TextSecure se almacenan en una base de datos cifrada en el teléfono. De la misma forma, los mensajes serán cifrados durante la transmisión de los mismos siempre y cuando la otra persona utilice TextSecure también.

8.6.9 SandBox para Android

El aumento de aplicaciones maliciosas que han ido apareciendo para Android, (no solo en “markets” alternativos, sino también en el oficial) ha motivado la creación de la primera sandbox para Android.

Una sandbox permite recrear un escenario virtual seguro donde analizar las aplicaciones de forma dinámica, pudiendo detectar las acciones que realiza a distintos niveles y registrándolas para un análisis más en profundidad. Existen desde hace mucho tiempo para PC.Organizaciones públicas y privadas permiten el envío de un fichero y se devuelve un informe de su actividad.

La virtualización ha permitido facilitar mucho el proceso, aunque también los atacantes han sabido aprovecharse de esta circunstancia. Buena parte del malware de PC actual sabe detectar si se encuentra en un entorno virtual o en una sandbox y puede o bien no ejecutarse o bien modificar su comportamiento. Con esto dificultan su estudio.

Aunque ya existían algunas sandbox privadas para Android, el proyecto “HoneyNet” supone la primera sandbox gratuita por todo el que quiera montar su propio laboratorio en casa. La versión actual es alfa, lo que implica que, aunque inmadura, es funcional.

DroidBox, como ha sido bautizada, hace uso de TaintDroid, un proyecto para la monitorización en tiempo real creado por varias universidades estadounidenses e Intel. Por ahora, DroidBox crea un informe tras la ejecución de una aplicación que devuelve la siguiente información:

- Operaciones de lectura y escritura de ficheros.
- La actividad de las API criptográficas.
- Conexiones de redes abiertas.
- Salida de tráfico.
- Fuga de información a través de ficheros SMS, o redes.
- Intentos de envío de SMS.
- Llamadas realizadas.

8.6.10 DroidBox

En los últimos tiempos ha habido un aumento de aplicaciones maliciosas para Android apareciendo tanto en mercados oficiales como en secundarios. Si tuviésemos una herramienta que a través de la técnica de sandboxing esta nos podría proporcionar una perspectiva inicial sobre el comportamiento de nuestro paquete de aplicación, y así podríamos reducir el riesgo de exposición a este tipo de malware.

Para lograr esto, el sandbox utilizará un pre-check estático, un análisis dinámico de corrupción y el control de la API. Las fugas de datos pueden ser detectados por contaminar los datos sensibles y la colocación de los sumideros de corrupción a través de la API. Además, el registro de los parámetros relevantes de la función API y valores de retorno, un malware potencial puede ser descubierto y reportado para su posterior análisis.

8.6.11 TaintDroid

Algo olía feo entre las más de 70.000 aplicaciones descargables para Android. Investigadores de las universidades de Duke y Penn State, junto con Intel Labs desarrollaron un software para monitorear la actividad de algunas aplicaciones y detectar qué datos de los usuarios le envían a los servidores remotos.

TaintDroid es un nuevo fiscalizador de aplicaciones para dispositivos móviles Android que fue desarrollado por investigadores de universidades estadounidenses. Concretamente en Duke y Penn State crearon un programa de seguimiento para ver qué hacen las aplicaciones Android por detrás de lo mostrado al usuario. Al parecer el

resultado fue el tristemente esperado, aproximadamente más de la tercera parte de las aplicaciones gratuitas para móviles con la plataforma Android que se pueden descargar en Google Play, extraen información con objetivos de recabar datos para fines publicitarios.

No se trata de virus ni troyanos propiamente dichos, pero sí del estudio de comportamiento de altas cifras de personas que utilizan nuestros dispositivos móviles. Entre los datos emitidos por estas aplicaciones está el envío de Coordenadas GPS y el número de teléfono, entre otros.

TantDroid, es una extensión muy inteligente que incluso puede llegar a renovar el concepto de los sistemas de protección de datos privados porque permite ver a los usuarios lo que las aplicaciones que han descargado están haciendo momento a momento gracias al uso de una versión similar al “Dalvik VM” (Java para SO Android) y un módulo de Kernel que intercepta la actividad del sistema en tiempo real. Cuando la aplicación comienza el proceso de envío de datos privados a una red externa, aparece una ventana emergente que advierte al usuario de tal maniobra.

En el estudio, en el que colaboraron además investigadores de IntelLab, se sometieron 30 aplicaciones muy populares de amplia descarga y se presentó el resultado de la investigación en la Conferencia Usenix de Vancouver.

Los resultados fueron bastante concluyentes y preocupantes, ya que un número importante de algunas populares aplicaciones gratuitas presentes en el Google Play están extrayendo y filtrando información de quienes las descargan sin su consentimiento ni conocimiento, principalmente a redes publicitarias. Realizaron un seguimiento a 30 aplicaciones y descubrieron que la mitad le estaba enviando información privada a servidores de publicidad, que incluye la ubicación geográfica y el número telefónico, incluso con coordenadas GPS cada 30 segundos y aún cuando no necesariamente está enviando publicidad al equipo.

Ante la duda que todo usuario se estará preguntando en este punto acerca de cuántos recursos podría llegar a consumir una aplicación de protección como TaintDroid en modo de ejecución, los responsables afirmaron que sobrecarga el CPU en un 14% y la memoria RAM en un 4.4%.

Falta saber la identidad de las aplicaciones que caen en esta actividad por parte del equipo investigativo y que la muestra fuese ampliada de las apenas 30 seleccionadas a el total de las 70000, o esperar que Google tome cartas en el asunto por un tema de transparencia mínimo.

8.7 La NSA publica una versión fortificada de Android

Con objeto de limitar el ámbito de explotación de aplicaciones vulnerables y maliciosas en terminales Android, la Agencia de Seguridad Nacional de los Estados Unidos ha hecho pública una versión de Android con un gran número de mejoras de seguridad.

Dicha versión de Android, se basa en SELinux (Security-Enhanced Linux), conjunto de políticas de seguridad y controles de acceso implementados en el propio kernel del Sistema Operativo por medio de Linux Security Modules (LSM), y que también fue desarrollado por la NSA. SELinux aplica controles de acceso que proporcionan a los programas de usuario y servicios del sistema la mínima cantidad de privilegios necesarios para llevar a cabo sus funciones. Esto reduce o elimina la capacidad de estos programas de vulnerar el sistema al completo en el caso de ser comprometidos (por medio, por ejemplo, de desbordamientos de búfer o configuraciones incorrectas). Este sistema de seguridad funciona independientemente de los mecanismos tradicionales de control de acceso en los que se basa Linux.

Según se describe en la propia Wiki de proyecto, SE Android también se refiere a la implementación de referencia creada por el propio “SE Android Project” destinada a proporcionar ejemplos prácticos de como habilitar y aplicar SELinux en las capas inferiores de Android, además de mostrar su eficacia frente a exploits y aplicaciones vulnerables. El público objetivo, por tanto, al que está destinado este proyecto son desarrolladores de aplicaciones móviles, expertos en seguridad, fabricantes de dispositivos, así como usuarios avanzados de Android que deseen un mayor nivel de seguridad y privacidad en sus smartphones. Algunas de las características de SE Android se muestran a continuación:

Sistemas de ficheros yaffs2 y ext4:

- Control de permisos del Kernel (Binder IPC).
- Etiquetado de sockets de servicio/ficheros creados por init.
- Etiquetado de nodos de dispositivos creados por ueventd.
- Control de permisos del espacio de usuario (Zygote socket commands).
- Soporte SELinux para Android toolbox.
- Uso de categorías MLS para aislar apps.

8.7.1 SELinux (Security-Enhanced Linux)

Security-Enhanced Linux o *SELinux*, es una arquitectura de seguridad integrada en el kernel 2.6.*x* usando los *módulos de seguridad linux* (o *linux security modules*, LSM). Este es un proyecto de la Agencia de Seguridad Nacional (NSA) de los Estados Unidos y de la comunidad SELinux. La integración de SELinux en Red Hat Enterprise Linux fue un esfuerzo conjunto entre al NSA y Red Hat.

SELinux proporciona un sistema flexible de *control de acceso obligatorio* (*mandatory access control*, MAC) incorporado en el kernel. Bajo el Linux estándar se utiliza el *control de acceso a discreción* (*discretionary access control*, DAC), en el que un proceso o aplicación ejectutándose como un usuario (UID o SUID) tiene los permisos y de ese usuario en los objetos, archivos, zócalos y otros procesos. Al ejecutar un kernel SELinux MAC se protege al sistema de aplicaciones maliciosas o dañadas que pueden perjudicar o destruir el sistema. SELinux define el acceso y los derechos de transición de cada usuario, aplicación, proceso y archivo en el sistema. SELinux gobierna la interacción de estos sujetos y objectos usando una política de seguridad que especifica cuán estricta o indulgente una instalación de Red Hat Enterprise Linux dada debería de ser.

En su mayor parte, SELinux es casi invisible para la mayoría de los usuarios. Solamente los administradores de sistemas se deben de preocupar sobre lo estricto que debe ser una política a implementar en sus entornos de servidores. La política puede ser tan estricta o tan indulgente como se requiera, y es bastante detallada. Este detalle le dá al kernel SELinux un control total y granular sobre el sistema completo.

Cuando un sujeto, tal como una aplicación, intenta acceder a un objeto tal como un archivo, el servidor de aplicación de políticas verifica un *caché de vector de acceso* (AVC), donde se registran los permisos de objeto y del sujeto. Si no se puede tomar una decisión basado en los datos en el AVAC, la petición continua al servidor de seguridad, el cuál busca el *contexto de seguridad* de la aplicación y del archivo en una matriz. Los permisos son entonces otorgados o negados, con un mensaje de "avc: denied" detallado en "/var/log/messages". Los sujetos y objetos reciben su contexto de seguridad a partir de la política instalada, que también proporciona información para llenar la matriz de seguridad del servidor.

Además de ejecutarse en un modo *impositivo*, SELinux puede ejecutarse en un modo *permisivo*, donde el AVC es verificado y se registran los rechazos, pero SELinux no hace cumplir esta política.

8.7.2 Linux Security Modules (LSM)

LSM fue diseñado para proporcionar las necesidades específicas con todo lo necesario para implementar con éxito un control de acceso obligatorio del módulo, al tiempo que impone la menor cantidad posible de cambios en el kernel de Linux. LSM evita el enfoque de la interposición de llamadas del sistema que se utiliza en “Systrace” porque no se adapta a núcleos multiprocesadores y está sujeta a ataques TOCTTOU (“Time of Check to Time of Use”, el sistema comprueba la identidad al principio de la sesión pero utiliza esta información más tarde para tomar decisiones de control de acceso). En su lugar, inserta LSM “hooks” (upcalls al módulo) en cada punto del núcleo donde una llamada al sistema a nivel de usuario está a punto de devolver un acceso a un objeto del núcleo interno importante como “inodes” y bloques de tareas de control.

El proyecto está estrechamente relacionado para resolver el problema del control de acceso evitando la imposición de un parche grande y complejo en el núcleo principal. No está pensado como un “hook” general o un mecanismo de “upcalls”, ni tampoco apoya al nivel de virtualización del sistema operativo.

El objetivo del control de acceso LSM está muy estrechamente relacionado con el problema de la auditoría del sistema, pero es sutilmente diferente. La auditoría requiere que cualquier intento de acceso debe ser registrado. LSM no puede ofrecer eso, porque requeriría una gran cantidad de “hooks” más, a fin de detectar los casos en que las llamadas del sistema al núcleo fallan y devuelven un código de error antes de conseguir los objetos significativos.

El diseño de LSM se describe en el documento “*módulos de seguridad de Linux: Apoyo de Seguridad General para el kernel de Linux”* presentado en “USENIX Security 2002”.

8.7.3 YAFFS (Yet Another Flash File System)

Es el primer sistema de ficheros que fue diseñado específicamente para “Memorias Flash NAND”.

Es un sistema de ficheros de registros con soporte a transacciones que automáticamente provee "wear-levelling" y robustez ante fallos de energía. Además funciona bien sobre grandes dispositivos Flash, en términos de tiempo de inicio y uso de RAM. Se usa actualmente en productos como Linux o WinCE, que ha probado ser realmente fiable. Una variante llamada "*YAFFS/Direct*" se usa en situaciones donde no hay sistema operativo, sino que hay un sistema operativo empotrado o son sistemas empotrados sin SO: tiene el mismo núcleo como sistema de ficheros pero una interfaz mucho más simple para el sistema operativo y el hardware NAND de la memoria flash.

El diseño de YAFFS tienen como prioridades las siguientes cuestiones:

- Memoria Flash NAND como soporte fundamental.
- Robustez a través de las estrategias de registro.
- Reducir significativamente la sobrecarga de RAM y los tiempos de inicio derivados de JFFSx.

Los datos de un fichero son almacenados en "trozos" consistentes con el tamaño de una página (por ej. 512B). Cada página es marcada con un identificador de fichero y un número de trozo, los números de trozo se numeran como 1, 2, 3, etc., siendo 0 la cabecera. Estas etiquetas son almacenadas en la región de *datos dispersos* de la memoria Flash. El número de trozo se determina dividiendo la posición del fichero por el tamaño de trozo.

Cada página dentro de un bloque debe escribirse en orden secuencial, y cada página se debe escribir una sola vez.

Cuando los datos son sobrescritos, los trozos relevantes son reemplazados por nuevas páginas escritas conteniendo los nuevos datos pero las mismas etiquetas. Los datos sobrescritos se marcan como “descartados”. Las cabeceras de los ficheros son almacenadas en una única página, marcada para ser diferenciada de las demás. Un bloque con sólo páginas descartadas es un bloque sucio candidato para la recolección de basura. En otro caso, las páginas válidas pueden ser copiadas a un solo bloque para que todo un bloque sucio pueda ser utilizado en la recolección de basura.

8.7.4 Multilevel security or multiple levels of security (MLS)

Seguridad de niveles múltiples (también definido como seguridad de niveles múltiples o abreviado como MLS) es el uso de un sistema informático para procesar la información con diversas sensibilidades (es decir. en diversos niveles de la seguridad), permite el acceso simultáneo de los usuarios con diferentes habilitaciones de seguridad, y evita que los usuarios obtengan el acceso a la información para la cual carecen de autorización.

MLS permite el acceso fácil a la información menos sensible de los individuos con alta habilitación, y permite a estos últimos compartir fácilmente documentos esterilizados con los individuos de menos habilitación. Un documento esterilizado es uno que se ha corregido para quitar la información que no se permite conocer al individuo de menor habilitación.

Capítulo 9

La aplicación Agenda Personal

9 La aplicación Agenda Personal

En este capítulo se explica paso a paso el desarrollo completo de *Agenda Personal*, una aplicación cuyo objetivo es localizar y mostrar en una lista los contactos almacenados en el dispositivo, añadiendo además otra serie de funcionalidades que explicaremos en puntos posteriores.

9.1 Introducción a Agenda Personal

El objetivo básico de *AgendaPersonal* es mostrar al usuario un listado de identificadores, números de teléfonos y direcciones de correo electrónico de los contactos que tenga almacenados en el propio dispositivo móvil, permitir el alta, baja, modificación de los mismos además de realizar llamadas telefónicas, envío de SMS, envío de Emails y realización de una copia de seguridad en la tarjeta SD de dicho dispositivo. Para ello utilizamos fundamentalmente cuatro actividades y una sub-actividad:

- Actividad *Carga*, es la encargada de buscar los datos de los contactos almacenados en el dispositivo, de crear la base de datos de nuestra aplicación y de almacenar dichos datos en la misma.

- Actividad *Inicio*, es la encargada de recoger la información guardada en nuestra base de datos y listarla en la pantalla del dispositivo móvil, también es la encargada de gestionar las opciones que damos al usuario tanto en el menú contextual del contacto como en el menú de opciones de nuestra aplicación.

- Actividad *Anadir*, es la encargada de añadir contactos nuevos a la base de datos de nuestra aplicación insertando dichos datos proporcionados por el usuario en la misma.

- Actividad *Editar*, es la encargada de editar los contactos de nuestra aplicación y posteriormente actualiza nuestra base de datos.

- Sub-actividad *MostrarContacto*, es la encargada de mostrar al usuario la información existente en nuestra base de datos referente al contacto seleccionado.

El usuario visualiza la pantalla de carga de nuestra aplicación por un breve periodo de tiempo, mientras tanto, sin que el usuario sea consciente de ello, nuestra aplicación en el caso de que sea la primera vez que la ejecutamos en el dispositivo, crea la base de datos con sus respectivas tablas, realiza las consultas pertinentes para la obtención de la información sensible de los contactos de el dispositivo móvil, tales como el nombre del contacto, número de teléfono y la dirección de correo electrónico, a continuación realiza las operaciones de inserción de los datos anteriormente mencionados.

Después de realizar las operaciones anteriormente mencionadas la actividad Carga da paso a la actividad Inicio y posteriormente finaliza.

La actividad Inicio es la encargada de mostrar al usuario la información obtenida en la actividad anterior, la mostramos en forma de lista ordenada alfabéticamente, y desde esta actividad podemos realizar las distintas operaciones para las que se ha creado la aplicación, disponemos de un menú de aplicación y un menú contextual para cada contacto seleccionado de manera prolongada.

En el menú general de la aplicación podemos realizar las operaciones de añadir nuevos contactos, realizar una copia de seguridad o “backup” de los contactos en la tarjeta SD del dispositivo telefónico y la operación de salir de la aplicación.

En el menú contextual de cada contacto podemos realizar las operaciones de realizar una llamada al número de teléfono del contacto, enviar un mensaje SMS al contacto, enviar un email a su dirección de correo electrónico, editar los datos del contacto y por último borrar el contacto de nuestra base de datos.

La actividad Anadir es la que permite al usuario la inserción de un nuevo contacto en la base de datos de la aplicación, mostramos al usuario una pantalla con tres campos a rellenar (nombre del contacto, número de teléfono y dirección de correo electrónico) y

dos botones, uno de guardado que realiza la inserción en la base de datos y otro para cancelar la operación, volver a la actividad Inicio y posteriormente cerrar la actividad Anadir.

La actividad Editar es la que permite al usuario la actualización de los datos de cualquier contacto seleccionado, podemos cambiar cualquiera de los tres datos relevantes del mismo, mostramos al usuario una pantalla con los tres campos de datos del contacto y dos botones, uno para guardar la actualización en la base de datos y otro para cancelar la operación que devuelve el control a la actividad Inicio y cierra posteriormente la actividad Editar.

Y por último la sub-actividad MostrarContacto es la encargada de mostrar al usuario la información sensible del contacto seleccionado tal como nombre, número de teléfono y dirección de correo electrónico, en la pantalla se recoge la información que se muestra al usuario y un botón para volver a la actividad Inicio y cerrar la sub-actividad posteriormente.

9.2 Análisis y diseño de la aplicación

Mediante este desarrollo se busca ilustrar de una forma más práctica las características principales que ofrece Android y pretende además servir como ejemplo para la creación de otras aplicaciones.

Algunos de los aspectos más interesantes de Android y que se explican con esta aplicación son los siguientes:

- Uso de componentes *Activity*.
- Uso de componentes *Service*.
- Solicitudes a través de *Intents*.
- Uso de la clase *JavaMail*.
- Uso de la base de datos *SQLite*.
- Composición del archivo “*AndroidManifest.xml*”.
- Acceso a la información de los contactos.
- Composición de interfaces de usuario, tanto con código como con XML.
- Composición gráfica de elementos en pantalla.
- Gestión de opciones de menú.

- Llamadas telefónicas.
- Envío de SMS.
- Envío de correo electrónico.
- Almacenamiento de archivos en tarjeta SD.

9.3 Casos de uso

En este apartado se utilizan los casos de uso como forma de acercar al lector las funcionalidades que la aplicación *AgendaPersonal* va a desempeñar frente al usuario. Sin embargo, no se va a profundizar demasiado en las posibilidades que ofrece esta técnica, sino que se limitará solamente en aquellos aspectos que son más ilustrativos y concisos para el lector y que le ayuden a comprender mejor qué es lo que realiza la aplicación.

Como recordatorio se dirá que, en ingeniería del software, un caso de uso representa un uso típico que se le da al sistema. La técnica de los casos de uso permite capturar y definir los requisitos que debe cumplir una aplicación, y describe las típicas interacciones que hay entre el usuario y esta. Dicha técnica es utilizada con frecuencia por los ingenieros del software para, entre otras cosas, mostrar al cliente de forma clara y sencilla qué tipo de acciones podrá realizar su futuro sistema.

A continuación se muestra un diagrama con los casos de uso asociados a *AgendaPersonal*, utilizando el estándar UML 2.0 [19].

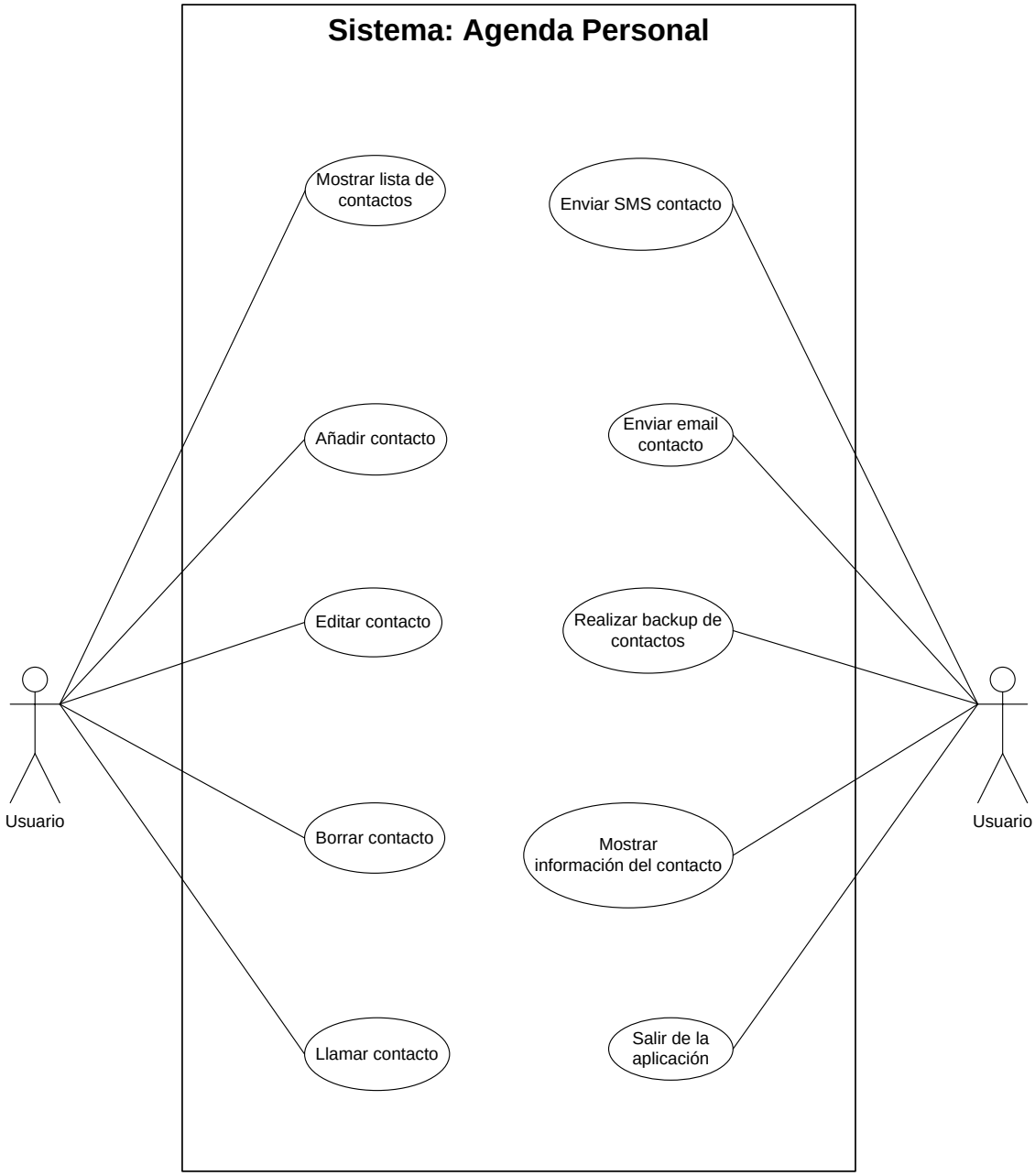


Figura 14. Diagrama casos de uso.

En diagrama mostrado en la Figura 14, el sistema, (es decir, la aplicación *AgendaPersonal*) está representado por una caja que contiene los casos de uso. Cada caso de uso consiste en un óvalo con un nombre descriptivo en su interior. Fuera del sistema se encuentra los actores que pueden interactuar con él. En este caso, a pesar de estar representado por dos figuras por motivos de espacio y legibilidad, existe un **único actor** de nombre “usuario” que es el que realiza todos los casos de uso.

A continuación se ofrece una breve descripción del cometido de cada uno de los casos de uso mostrados en el diagrama:

- **Mostrar lista de contactos:** muestra al usuario una lista en orden alfabético de los contactos almacenados en la base de datos de la aplicación.

- **Añadir contacto:** realiza la inserción de un nuevo contacto en la base de datos de la aplicación.

- **Editar contacto:** realiza la actualización de un contacto ya existente en la base de datos de la aplicación.

- **Borrar contacto:** realiza el borrado de un contacto existente en la base de datos de la aplicación.

- **Llamar contacto:** realiza una llamada telefónica al contacto seleccionado por el usuario.

- **Enviar SMS contacto:** realiza el envío de un mensaje de texto al contacto seleccionado por el usuario.

- **Enviar email contacto:** realiza el envío de un email a la dirección de correo electrónico del contacto seleccionado por el usuario.

- **Realizar back-up de contactos:** realiza una copia de seguridad en un archivo de texto plano de los contactos de la base de datos de la aplicación y lo almacena en la tarjeta SD del dispositivo.

- **Mostrar información del contacto:** muestra al usuario la información sensible del contacto seleccionado tal como nombre del contacto, número de teléfono y dirección de correo electrónico.

- **Salir de la aplicación:** cierra la aplicación evitando que quede ejecutándose en segundo plano y consuma recursos del dispositivo.

Es importante recalcar que los casos de uso simplemente expresan el punto de vista del usuario sobre cómo debe funcionar la aplicación y qué puede realizar a través de ella, y son una forma de facilitar su comprensión. No tiene porqué existir ninguna

correspondencia entre los casos de uso y la clases finalmente implementadas, más allá de que las clases en su conjunto, como sistema completo, realizan aquello que los casos de uso expresan.

9.4 Modelo de clases

Las clases utilizadas para resolver un determinado problema, sus atributos y métodos, la visibilidad que de estos tienen las demás clases, así como las relaciones que existen entre ellas y sus colaboraciones, constituyen el modelo de clases. Mediante este tipo de modelos se expresa, con mayor o menor nivel de detalle, la futura implementación del sistema, así como permite dar una idea bastante cercana a la forma en la que se ha abordado el problema.

En el presente apartado se ofrece al lector una breve descripción de cuál es el modelo de clases utilizado en la aplicación *AgendaPersonal*. Nuevamente, el objetivo principal es ayudar a comprender cómo se soluciona el problema planteado y pretende servir de anticipo a la explicación más detallada de la implementación de la aplicación, que se da en secciones posteriores. Por todo ello, el modelo de clases ofrecido no incluye más que clases, atributos, métodos, y la visibilidad de estos, con vistas a no perder al lector con aspectos demasiado complejos.

En la Figura 15 se enseña el diagrama correspondiente al modelo de clases utilizado, siguiendo el estándar UML 2.0 [19].

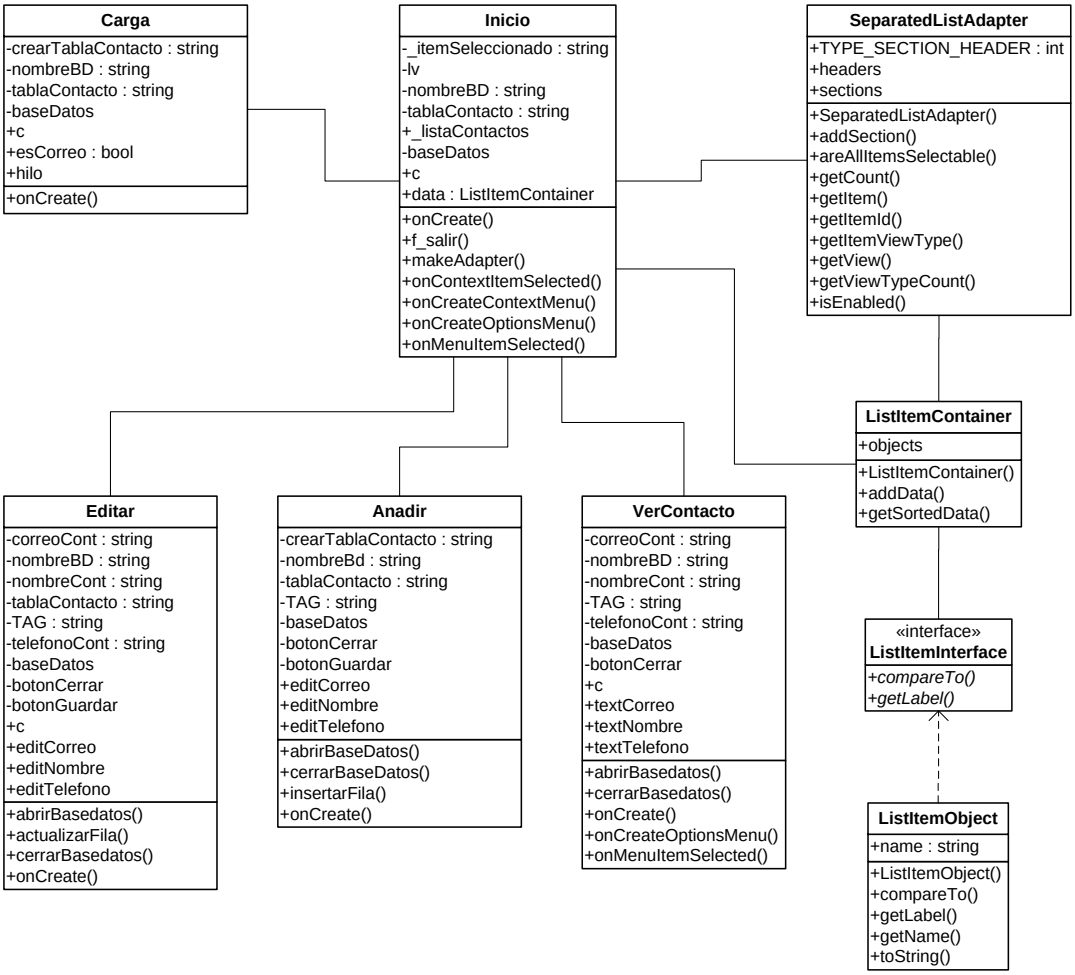


Figura 15. Diagrama de clases.

9.5 Estructura y arquitectura de la aplicación

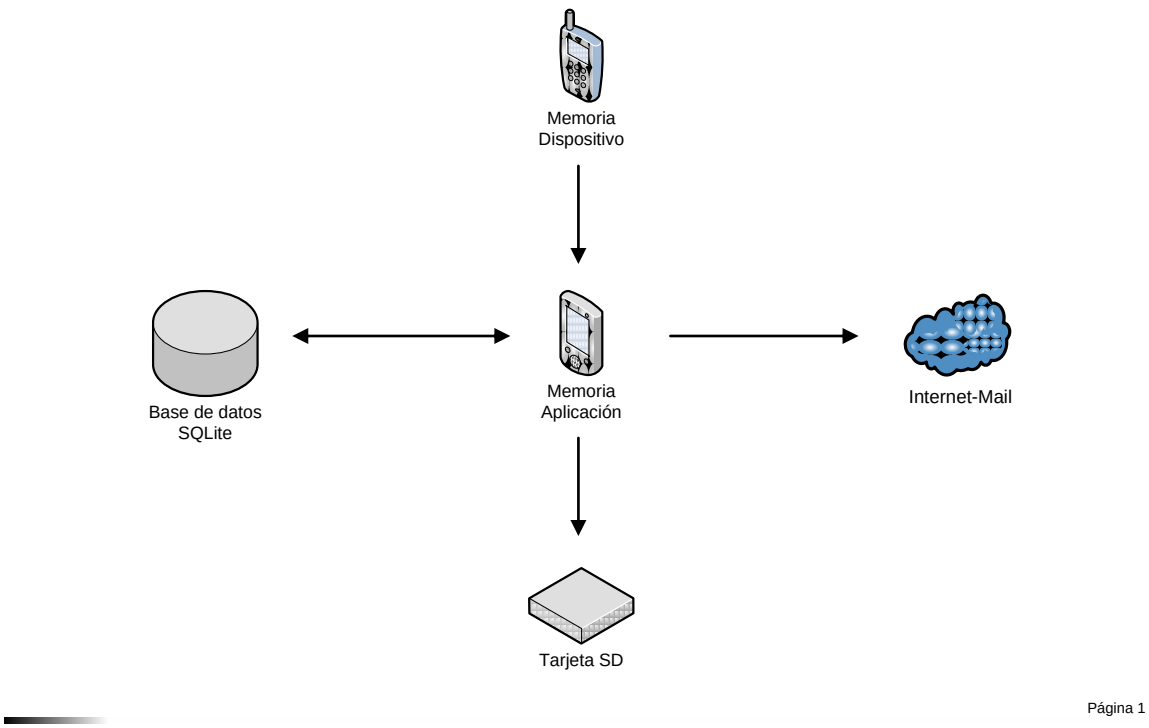


Figura 16. Estructura del proyecto.

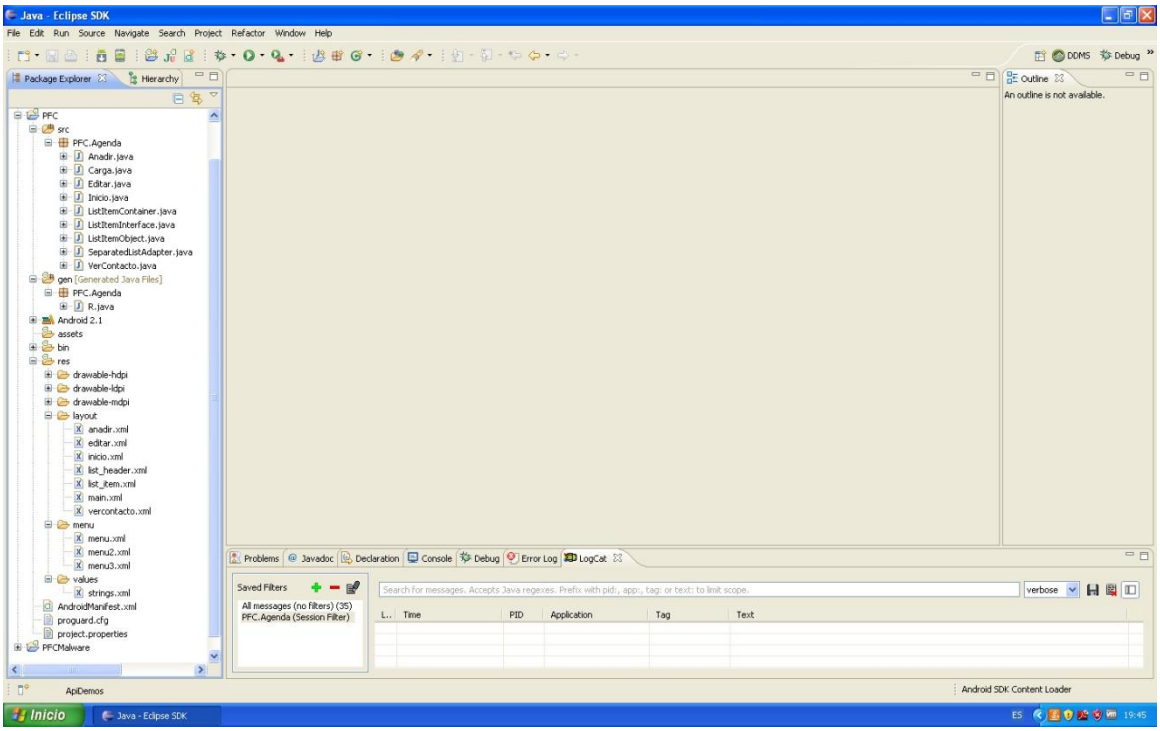


Figura 17. Arquitectura de la aplicación.

9.6 Desarrollo e Implementación

Una vez expuestos los objetivos y características principales de la aplicación, así como las decisiones tomadas en cuanto a su diseño, a continuación se desgranan en este apartado los aspectos relacionados con su implementación.

El fin perseguido en las siguientes líneas no es sólo mostrar el código más relevante de *AgendaPersonal*, sino ofrecer también una explicación general sobre cómo funcionan las aplicaciones para la plataforma Android, de forma que a su vez se consigan mostrar las peculiaridades de este sistema operativo. Así mismo, los detalles de implementación mencionados esperan poder ayudar a esclarecer y justificar las conclusiones de la investigación realizada en este proyecto.

Se advierte al lector de que los fragmentos de código fuente a continuación mostrados no son una copia literal del código fuente de *AgendaPersonal*, sino que en algunos casos se reducen o modifican por motivos de limitación de espacio, pero sobre todo por simplificar y facilitar la comprensión de la aplicación desarrollada para tal investigación.

9.6.1 Representación y acceso de los contactos en memoria

Para cada contacto, la aplicación necesita conocer su nombre, número de teléfono y su dirección de correo electrónico. Toda esta información se obtiene desde diferentes fuentes: los contactos almacenados en el dispositivo móvil o la base de datos SQLite de la aplicación.

Para reunir todos estos datos se utilizaremos la variable global *c* de la clase Cursor, realizando previamente la consulta pertinente indistintamente a la base de datos SQLite de la aplicación o al almacenamiento de contactos del propio dispositivo. Se representará en memoria a través de una lista llamada *_listaContactos* que en realidad es variable global de la clase String ya que lo que nos interesa es mostrar al usuario un listado de los contactos almacenados en el dispositivo, en particular el nombre de los mismos.

Para representar a un contacto utilizaremos las variables globales *nombreCont*, *telefonoCont* y *correoCont* de la clase String en la actividad VerContacto, hemos elegido esta forma de representación con tres variables simples ya que estos dispositivos tienen una limitación de recursos considerable.

Realizaremos una primera recuperación de contactos para la carga de la aplicación por primera vez en el dispositivo, en esta ocasión los contactos recuperados tienen su origen en la memoria del dispositivo, a continuación la aplicación crea la lista de contactos y crea nuestra base de datos y almacena los contactos obtenidos.

Cada contacto tiene tres atributos declarados como hemos mencionado anteriormente:

- nombreCont: para el nombre.
- telefonoCont: para el número de teléfono del contacto.
- correoCont: para la dirección de correo electrónico del contacto.

La base de datos de nuestra aplicación y la lista de contactos en memoria con los nombres de los respectivos contactos, se actualiza en cada adición de un nuevo contacto o en el borrado de los mismos.

9.6.2 Dibujar y gestionar elementos de la Agenda

La interfaz principal de *AgendaPersonal* consiste en la vista completa de un listado de contactos donde el usuario visualiza el nombre de los contactos, puede desplazarlo verticalmente libremente, realizar la selección de un contacto, o bien mantener pulsado un contacto para centrar el foco en él.

Además, al tocar sobre el contacto este muestra una pantalla con la información sensible a dicho contacto.

Por lo tanto, los elementos que se deben dibujar sobre la interfaz principal vista por el usuario son los siguientes:

- Un *Listview*, que representa el listado de los distintos contactos.
- El nombre de los contactos.
- Un menú contextual que se activa cuando se mantiene pulsado un contacto.
- Un menú general de la aplicación que se muestra al usuario cuando este presiona la tecla “menú” del dispositivo.

La pantalla con la información debe ser pintada en caso de que el usuario pulse sobre un contacto. Por lo tanto, es necesario también tomar el control de los elementos dibujados a fin de saber cuándo el usuario ha pulsado sobre uno de ellos.

El menú contextual es dibujado al mantener presionado uno de los contactos, por tanto, también es necesario como en el caso anterior tomar el control de los elementos dibujados a fin de saber cuándo el usuario ha mantenido pulsado sobre uno de ellos.

Para la ordenación de los contactos en la agenda hemos utilizado una interfaz que separa los contactos por letras y los ordena alfabéticamente dentro de sus categorías que están representadas por la letra inicial de cada contacto, esta interfaz se construye con las clases *ListItemContainer*, *ListItemInterface* y *ListItemObject*, la agrupación y ordenación alfabética de los contactos las realizamos mediante objetos de la clase *HashMap*, ahondaremos en estos temas con mas profundidad en apartados posteriores.

9.6.3 Control de la pantalla de carga de la aplicación

De la pantalla de carga de la aplicación, es importante resaltar el control que realizamos a la hora de comprobar si la base de datos que crea y gestiona nuestra aplicación, está correctamente creada con las información sensible de los contactos.

Para realizar está comprobación utilizamos el siguiente código en la clase Java *Carga* de nuestro paquete de aplicación:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  //cargamos el layout de la actividad.
  setContentView(R.layout.main);
  //creamos un hilo para cargar los contactos del teléfono.
  new Handler().postDelayed(new Runnable(){
    public void run() {
    //Abrimos o creamos la base de datos de la agenda.
    baseDatos = openOrCreateDatabase(nombreBD, MODE_WORLD_WRITEABLE,
null);
```

```
//Intentamos realizar una consulta, si no salta excepcion quiere decir
que no es la primera vez que accedemos a la agenda por tanto ya esta la
tabla creada y los contactos del teléfono introducidos.
try{
  c = baseDatos.rawQuery("SELECT nombre FROM contacto ORDER BY nombre",
null);
   }
catch (Exception e) {
  baseDatos.execSQL(crearTablaContacto);
```

Código 6. Control clase Carga I.

Tras comprobar que la base de datos está correctamente creada, tenemos que introducir los datos de los contactos almacenados en el dispositivo en nuestra base de datos, tras realizar las consultas pertinentes hacemos una comprobación para asegurarnos que hemos recuperado los datos que buscábamos, tanto para el nombre del contacto como para el número de teléfono:

```
  try {
    //Las columnas que queremos en el resultado.
    String[] projection = new String[] {
                        Data.DISPLAY_NAME,
                        Phone.NUMBER};
    //Consulta: contactos del teléfono con sus respectivos números de
teléfono ordenados por nombre.
    Cursor datos = getContentResolver().query(
                    Data.CONTENT_URI,
                    projection,
                    Data.MIMETYPE + "='" + Phone.CONTENT_ITEM_TYPE + "'",
                    null,
                    Data.DISPLAY_NAME + " ASC");
    startManagingCursor(datos);
    //Nos aseguramos de que existe al menos un registro.
    if (datos.moveToFirst()) {
    //Recorremos el cursor hasta que no haya más registros.
      do {
          int name = datos.getColumnIndexOrThrow(Data.DISPLAY_NAME);
          int number = datos.getColumnIndexOrThrow(Phone.NUMBER);
          String nombre = datos.getString(name);
          String telefono = datos.getString(number);
      //Insertamos en nuestra base de datos tanto el número de teléfono
como el nombre del contacto.
          ContentValues values = new ContentValues();
          values.put("nombre",nombre );
          values.put("telefono", telefono);
          values.put("correo", "Email");
          baseDatos.insert(tablaContacto, null, values);
          } while(datos.moveToNext());
      }
```

Código 7. Control clase Carga II.

Realizamos el mismo control mencionado anteriormente para la dirección de correo electrónico, es decir, comprobamos que hemos recuperado los datos de la memoria del dispositivo, como mostramos a continuación:

```
      //Las columnas que queremos en el resultado.
      String[] projection2 = new String[] {
```

```
                            Data.DISPLAY_NAME,
                            Email.DATA};
      //Consulta: contactos del teléfono con sus respectivos emails
ordenados por nombre.
      Cursor datos2 = getContentResolver().query(
                        Data.CONTENT_URI,
                        projection2,
                        Data.MIMETYPE + "='" + Email.CONTENT_ITEM_TYPE +
"'",
                        null,
                        Data.DISPLAY_NAME + " ASC");
      startManagingCursor(datos2);
      //Nos aseguramos de que existe al menos un registro.
      if (datos2.moveToFirst()) {
      //Recorremos el cursor hasta que no haya más registros.
      do {
         int name = datos2.getColumnIndexOrThrow(Data.DISPLAY_NAME);
         int email = datos2.getColumnIndexOrThrow(Email.DATA);
         String nombre = datos2.getString(name);
         String correo = datos2.getString(email);
      //Actualizamos la dirección de correo electrónico en la tabla de
contactos de nuestra base de datos.
         ContentValues values2 = new ContentValues();
         if (correo!=null){
             values2.put("correo", correo);
             baseDatos.update(tablaContacto, values2,
"nombre='"+nombre+"'", null);
         }
      } while(datos2.moveToNext());
 }
```

Código 8. Control clase Carga III.

A continuación, vamos a mostrar la interfaz que hemos diseñado para mostrarle al usuario mientras realizamos todo este conjunto de controles, como vemos en la siguiente figura.

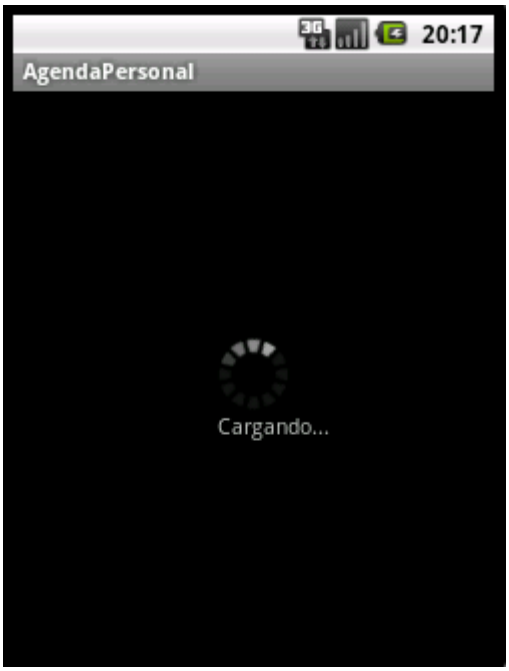


Figura 18. Carga.

9.6.4 Control de la lista de contactos

De la pantalla principal de la interfaz de nuestra aplicación, es importante resaltar el control que realizamos a la hora de comprobar si recuperamos la información sensible de los contactos de la base de datos que crea y gestiona nuestra aplicación.

Para realizar está comprobación utilizamos el siguiente código en la clase Java *Inicio* de nuestro paquete de aplicación, en el comprobamos gracias a la ayuda de la clase *Cursor* que nos permite recuperar los datos de la base de datos SQLite:

```
Cursor c;
//Método para crear la actividad Inicio.
@Override
public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 baseDatos = openOrCreateDatabase(nombreBD, MODE_WORLD_WRITEABLE,
null);//Abrimos la base de datos.

 c = baseDatos.rawQuery("SELECT nombre FROM contacto ORDER BY nombre",
null); //Consulta: nombre de los contactos de la base de datos ordenados
por nombre.

 //Nos aseguramos de que existe al menos un registro
 if (c.moveToFirst()) {
  //Recorremos el cursor hasta que no haya más registros
  do {
     data.addData(new ListItemObject(c.getString(0)));
     } while(c.moveToNext());
  }
  baseDatos.close(); //Cerramos la base de datos.
```

Código 9. Control clase Inicio.

A continuación controlamos que la velocidad de desplazamiento de nuestro listado de contactos gracias a la clase *Listview* (variable *lv*) pueda producirse rápidamente para la mejor localización de los contactos de la lista, mediante el siguiente código:

```
lv = getListView(); //Obtenemos el ListView.
lv.setTextFilterEnabled(true);
lv.setFastScrollEnabled(true); //Método que nos permite desplazarnos más
rapidamente por a lista de contactos.
```

Código 10. Clase Listview.

Cabe resaltar también los controles relacionados con las funcionalidades asociadas a los distintos menús de la aplicación propiamente dicha, así pues, vamos a comentar y mostrar el código del menú contextual que nos muestra la aplicación a mantener seleccionado prolongadamente un contacto de nuestra lista.

Para activar este tipo de menú en nuestro "view" o vista que coincide con la lista de contactos, utilizamos la siguiente línea de código:

```
registerForContextMenu(getListView());
```

Código 11. Clase Inicio activar menú.

Acto seguido, ayudándonos de la clase *OnItemClickListener* configuramos la acción a realizar al seleccionar un contacto de nuestra lista, en este caso al seleccionar por un breve periodo de tiempo dicho contacto, nuestra aplicación nos muestra la información personal del mismo, el código mostrado a continuación nos muestra como *AgendaPersonal* hace una llamada a la sub-actividad *VerContacto* que es la encargada de pintar al usuario la información del contacto mencionada anteriormente, para ello le pasamos a dicha sub-actividad mediante la clase *Bundle* el nombre del contacto seleccionado por el usuario y por último activamos el *Listener* creado para tal efecto en nuestro "view" mediante el siguiente código:

```
OnItemClickListener clickListener = new OnItemClickListener() {
  public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
  Intent intent = new Intent(Inicio.this, VerContacto.class);
  Bundle bundle = new Bundle();
  bundle.putString("NOMBRESEL", ((TextView) view).getText().toString());
  intent.putExtras(bundle);
  startActivityForResult(intent, RESULT_OK);
  }
};
lv.setOnItemClickListener(clickListener); //Activamos el Listener creado
anteriormente.
```

Código 12. OnItemClickListener.

Empleamos un control para la configuración de las distintas opciones mostradas al usuario de nuestro menú principal de la aplicación, esto lo realizamos mediante el siguiente método:

```
//Método para la configuración del menú estandar de la actividad Inicio,
se activa mediante el botón "menu" del dispositivo.
@Override
public boolean onCreateOptionsMenu(Menu menu) {
  MenuInflater inflater = getMenuInflater();
  inflater.inflate(R.menu.menu2, menu);
  return true;
}
```

Código 13. OnItemClickListener.

Para controlar las funcionalidades del menú mencionado anteriormente utilizamos el método:

```
//Método para configurar la acción a desarrollar por cada elemento del
menú.
@Override
public boolean onMenuItemSelected(int featureId, MenuItem item) {
  super.onMenuItemSelected(featureId, item);
  switch(item.getItemId()) {
    case R.id.anadir_contacto:
//Si el usuario selecciona añadir contacto lanzamos la actividad Anadir
y finalizamos la actividad Inicio.

            (...)

    case R.id.backup:
//Si el usuario seleccion Realizar Backup.
//Código para realizar un backup de los contactos en un fichero de texto
localizado en la tarjeta SD del dispositivo.

            (...)

      File fichero = new
File(Environment.getExternalStorageDirectory().getPath()+"/PFCagendaMalw
are/PFCagenda.txt"); //Indicamos el fichero en el que realizamos el
backup.
      File ruta = new
File(Environment.getExternalStorageDirectory().getPath()+"/PFCagendaMalw
are/"); //Indicamos la ruta del fichero.
      try {
        if (!ruta.exists()){ //Si la ruta no existe la creamos.
          ruta.mkdir();
        }
        if (fichero.exists()){ //Si el fichero ya existe lo borramos.
          fichero.delete();
        }
        fichero.createNewFile(); //Creamos el fichero.

            (...)
      }

            (...)

    case R.id.salir:
//Si el usuario elige la opción de salir, llamamos al método para
finalizar la actividad.

            (...)
```

```
  }
  return true;
}
```

Código 14. onMenuItemSelected.

Para la opción de realizar un back-up de los contactos en la tarjeta SD del dispositivo tenemos que controlar que el directorio exista o no, y el fichero en el que guardamos la información, respectivamente, como hemos resaltado en el fragmento de código anterior, además debe montado el almacenamiento externo del dispositivo, en puntos posteriores ahondaremos con más profundidad en este tema. Con el siguiente método controlamos las diferentes opciones que ofrecemos al usuario del menú contextual de cada contacto, que mencionamos anteriormente en este mismo punto:

```
//Método para la creación del menú que se lanza si el usuario deja
presionado un contacto.
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenuInfo menuInfo) {
  super.onCreateContextMenu(menu, v, menuInfo);
  AdapterView.AdapterContextMenuInfo info =
(AdapterView.AdapterContextMenuInfo)menuInfo;
//Almacenamos el elemento que ha seleccionado el usuario.
  _itemSeleccionado = lv.getAdapter().getItem(info.position).toString();
//Ponemos el nombre del elemento seleccionado como título del menú.
 menu.setHeaderTitle(lv.getAdapter().getItem(info.position).toString());
 MenuInflater inflater = getMenuInflater();
 inflater.inflate(R.menu.menu, menu);
}
```

Código 15. onCreateContextMenu.

Y con el siguiente método controlamos las funcionalidades del menú contextual propiamente dicho, según seleccione el usuario las distintas acciones:

```
//Método para configurar la acción a realizar cuándo un elemento del
menú es seleccionado.
@Override
public boolean onContextItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    case R.id.group_elemento1:
//Mostrar un mensaje de confirmación antes de realizar la llamada.
AlertDialog.Builder alertDialog = new AlertDialog.Builder(Inicio.this);
alertDialog.setMessage("¿Desea realizar la llamada al contacto?");
alertDialog.setTitle("Llamar a contacto...");
alertDialog.setIcon(android.R.drawable.ic_dialog_alert);
alertDialog.setCancelable(false);
alertDialog.setPositiveButton("Sí", new
DialogInterface.OnClickListener() {
  public void onClick(DialogInterface dialog, int which) {

            (...)

  }
 }
});
```

```
alertDialog.setNegativeButton("No", new
DialogInterface.OnClickListener() {
  public void onClick(DialogInterface dialog, int which) {

            (...)

  }
});
alertDialog.show();
return true;
    case R.id.group_elemento2:
//Si el usuario elige la opcion de enviar mensaje.
//Mostrar un mensaje de confirmación antes de realizar la llamada al
envio de mensajes.
AlertDialog.Builder alertDialog2 = new AlertDialog.Builder(Inicio.this);
alertDialog2.setMessage("¿Desea enviar un SMS al contacto?");
alertDialog2.setTitle("Enviar SMS a contacto...");
alertDialog2.setIcon(android.R.drawable.ic_dialog_alert);
alertDialog2.setCancelable(false);
alertDialog2.setPositiveButton("Sí", new
DialogInterface.OnClickListener() {
  public void onClick(DialogInterface dialog, int which) {

            (...)

  }
 }
});
alertDialog2.setNegativeButton("No", new
DialogInterface.OnClickListener() {
  public void onClick(DialogInterface dialog, int which) {

            (...)

  }
});
alertDialog2.show();
return true;
    case R.id.group_elemento3:
//Si el usuario elige la opcion de enviar email.

            (...)

return true;
    case R.id.group_elemento4:
//Si el usuario elige la opción de editar contacto.

            (...)

return true;
    case R.id.group_elemento5:
//Si el usuario selecciona que desea borrar el contacto.
//Mostramos un mensaje de confirmación antes de borrar el contacto.
AlertDialog.Builder alertDialog4 = new AlertDialog.Builder(Inicio.this);
alertDialog4.setMessage("¿Desea borrar el contacto?");
alertDialog4.setTitle("Borrando contacto...");
alertDialog4.setIcon(android.R.drawable.ic_dialog_alert);
alertDialog4.setCancelable(false);
alertDialog4.setPositiveButton("Sí", new
DialogInterface.OnClickListener() { //Si el usuario elige que sí.
```

```
  public void onClick(DialogInterface dialog, int which) {

            (...)

  }
 }
});
alertDialog4.setNegativeButton("No", new
DialogInterface.OnClickListener() { //Si el usuario elige la opción de
no borrar el contacto.
  public void onClick(DialogInterface dialog, int which) {

            (...)

  }
});
alertDialog4.show();
return true;
    default:
      return super.onContextItemSelected(item);
    }
  }
}
```

Código 16. onContextItemSelected.

En el código anteriormente mostrado a grandes rasgos, debemos destacar el uso de controles de confirmación por parte del usuario a la hora de realizar las siguientes acciones: realizar llamada telefónica, enviar SMS o borrar contacto.

Todos estos controles explicados y comentados anteriormente son los utilizados en la interfaz principal de nuestra aplicación, todos ellos componen las funcionalidades de la misma, y por lo tanto, representan las acciones a realizar por el usuario sobre los contactos de su dispositivo.

9.6.5 Control de visualización de contacto

La visualización de contactos es una pantalla de nuestra aplicación que muestra al usuario la información del contacto seleccionado únicamente, es importante resaltar el control que realizamos a la hora de comprobar si recuperamos la información sensible de los contactos de la base de datos de nuestra aplicación como hemos comentado en puntos anteriores, realizamos esta comprobación ya que es de vital importancia para el usuario que esté utilizando *AgendaPersonal*, además controlamos el contacto seleccionado por el usuario mediante la clase *Bundle* como mostramos en el código a continuación:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.vercontacto);
  Bundle bundle = this.getIntent().getExtras();//Recogemos el nombre
seleccionado por el usuario que hemos pasado a traves del Bundle, y lo
almacenamos en una variable.
  String nombreRec = bundle.getString("NOMBRESEL");
```

```
//Abrimos la base de datos.
  abrirBasedatos();
//Consulta: contacto del teléfono seleccionado por el usuario con su
respectivo nombre, número de teléfono y dirección de correo electrónico.
  c = baseDatos.rawQuery("SELECT nombre,telefono,correo FROM contacto
WHERE nombre LIKE '"+ nombreRec +"' ", null);
//Almacenamos el resultado obtenido en la consulta en nuestras
variables.
//Nos aseguramos de que existe al menos un registro.
  if (c.moveToFirst()) {

      (...)

  }
//Cerramos la base de datos.
cerrarBasedatos();
```

Código 17. Control clase VerContacto I.

Controlamos que tanto el número de teléfono del contacto seleccionado, como la dirección de correo electrónico tengan un valor recuperado o por el contrario carezcan de los mismos:

```
//Si en el teléfono del contacto no encontramos ningún valor le ponemos
el valor por defecto.
if (telefonoCont.equals("")){
  telefonoCont = "El contacto no dispone de telefono introducido";
}
//Si en la dirección de correo del contacto no encontramos ningún valor
le ponemos el valor por defecto.
if ((correoCont.equals(""))||(correoCont.equals("Email"))){
  correoCont = "El contacto no dispone de dirección de correo
electronica introducida";
}
```

Código 18. Control clase VerContacto II.

En este mismo apartado controlamos que el botón que compone la interfaz del usuario se le asigne el *Listener* correspondiente para que el usuario de la aplicación pueda interaccionar con la misma, este control lo implementamos con el código siguiente:

```
botonCerrar = (Button) findViewById(R.id.btCancelar);
//Configuramos la acción de el botón "volver".
//Finalizamos la subactividad VerContacto devolviendo "RESULT_OK" a la
actividad llamante Inicio.
botonCerrar.setOnClickListener(new View.OnClickListener() {
  public void onClick(View v) {
    Intent intent = new Intent(VerContacto.this, Inicio.class);
    setResult(RESULT_OK, intent);
    finish();
  }
});
```

Código 19. Control clase VerContacto III.

9.6.6 Control de adición de contactos

Para la adición de contactos mostramos una pantalla en la que ofrecemos al usuario unos campos a rellenar con la información que queremos añadir del contacto en cuestión, esta información consta de nombre del contacto, número de teléfono y dirección de correo, controlamos que el usuario una vez rellenados los campos pulse el botón de guardado de información, para ello nos apoyamos del método *setOnClickListener* utilizando el código mostrado a continuación:

```
//Añadimos el botón de Guardar.
botonGuardar = (Button) findViewById(R.id.btGuardar);

//Guardar el contacto actual en la agenda configurando el Listener del
botón Guardar.
botonGuardar.setOnClickListener(new View.OnClickListener() {
  public void onClick(View v) {
    boolean nombcorrecto = true;
    boolean tlfcorrecto = true;
    boolean emlcorrecto = false;

      (...)
}
```

Código 20. Control clase Anadir I.

Para la correcta introducción de información a cerca de un contacto realizamos una serie de controles sobre dicha información como son los siguientes:

- El nombre del contacto no puede ser un campo vacío.

- El número de teléfono tiene que contener dígitos comprendidos entre los caracteres ASCII "0" y "9", contemplamos también el carácter "+" para los casos especiales de números con el prefijo internacional que empiezan por dicho carácter.

- Y por último controlamos que la dirección de correo electrónico contenga el carácter "@" a lo largo de la cadena que la compone, además, controlamos si este campo ha sido introducido por el usuario o se ha dejado vacío, almacenando el valor "Email" que es nuestro caso por defecto, para este campo también quitamos los espacios en blanco si los hubiera.

Para todos estos controles nos apoyamos de variables de la clase *Boolean* para indicar cuáles han sido correctamente pasados y cuáles han fallado, para estos últimos, mostramos un mensaje al usuario advirtiéndole de dichos fallos como mostraremos más adelante.

En la realización de todos estos tipos de controles, utilizamos el siguiente código:

```
//Comprobamos que el nombre del contacto sea correcto.
//Devolvemos un booleano indicando si es correcto o no.
String nomb = editNombre.getText().toString();
  if ((nomb.compareTo("") == 0)||(nomb.compareTo("Nombre")== 0)){
    nombcorrecto = false;
  }
```

```
//Comprobamos que el número de teléfono y la dirección de email sean
correctas.
//Cada carácter del número de teléfono está comprendido entre "0" y "9"
o es el símbolo "+".
//Devolvemos un booleano indicando si es correcto o no.
String tlf = editTelefono.getText().toString();
  for (int i=0; i<tlf.length(); i++){
    if ((((tlf.charAt(i)<='9') &&
(tlf.charAt(i)>='0'))||(tlf.charAt(i)=='+')) && (tlfcorrecto)){
      tlfcorrecto = true;
    }
    else
      tlfcorrecto = false;
  }
//Comprobamos que la dirección de email sea correcta.
//La cadena debe contener el caracter "@".
//Devolvemos un booleano indicando si es correcto o no.
String eml = editCorreo.getText().toString();
for (int i=0; i<eml.length(); i++){
  if (eml.charAt(i)=='@'){
    emlcorrecto = true;
  }
}
//Evaluamos como correcto para el campo dirección de correo electrónico
la cadena vacía y la cadena "Email".
if ((eml.compareTo("Email")== 0)||(eml.compareTo("")==0)){
  emlcorrecto = true;
}
  (...)
```

Código 21. Control clase Anadir II.

A continuación y una vez validados los campos introducidos por el usuario, debemos realizar la inserción de los mismos en la base de datos de nuestra aplicación, para ello debemos controlar que dicha inserción se realiza correctamente como mostramos a continuación con el código siguiente:

```
//Si el campo número de teléfono y el campo dirección de correo
electrónico es correcto, procedemos a la inserción en nuestra base de
datos.
if ((tlfcorrecto)&&(emlcorrecto)&&(nombcorrecto)){
  //Abrir la base de datos, se creará si no existe.
  abrirBasedatos();
  //Quitamos los espacios en blanco, si los hay, de la dirección de
email.
  String email = new String();
  StringTokenizer stTexto = new
StringTokenizer(editCorreo.getText().toString());
  while (stTexto.hasMoreElements())
    email += stTexto.nextElement();
    //Insertar la fila o registro en la tabla "contacto".
    //Si la inserción es correcta devolverá true.
    boolean resultado = insertarFila(editNombre.getText().toString(),
editTelefono.getText().toString(), email);

    if(resultado)
      (...)  //Mostramos un mensaje de confirmación.
    else
```

```
      (...)  //Mostramos un mensaje de error en la inserción, el
contacto ya existía.
    //Cerramos la base de datos.
    cerrarBasedatos();
}

  (...)

//Método que realiza la inserción de los datos en nuestra tabla
contacto.
private boolean insertarFila(String nombre, String telefono, String
correo) {
  ContentValues values = new ContentValues();
  values.put("nombre",nombre );
  values.put("telefono", telefono);
  values.put("correo", correo);
  Toast.makeText(getApplicationContext(), "Nombre: " + nombre + ", " +
"teléfono: " + telefono, Toast.LENGTH_LONG).show(); //Mostramos un
mensaje con los datos introducidos en la base de datos.
  return (baseDatos.insert(tablaContacto, null, values) > 0);
}
```

Código 22. Control clase Anadir III.

Como hemos mencionado anteriormente en éste mismo punto, si alguno de los datos introducidos por el usuario no pasase los controles pertinentes, mostramos a éste un mensaje informando del error, para ello utilizamos el siguiente código:

```
if ((tlfcorrecto)&&(emlcorrecto)&&(nombcorrecto)){
  (...)
}
else{
  if (!nombcorrecto)
    (...)  //Mostramos un mensaje de error al introducir el nombre del
contacto.
  else{
    if (!tlfcorrecto)
      (...)  //Mostramos un mensaje de error al introducir el número de
teléfono.
    else
      (...)  //Mostramos un mensaje de error al introducir la dirección
de correo electrónico.
  }
}
(...)
```

Código 23. Mensajes clase Anadir III.

Y para finalizar, en ésta actividad controlamos que el usuario haga uso de la cancelación de la misma, para ello dotamos al botón “Volver” de un *Listener* cuya función es la de iniciar la actividad “Inicio”, que resulta ser la actividad llamante, y por último finalizar la actividad actual. Esto lo realizamos mediante el siguiente código:

```
botonCerrar = (Button) findViewById(R.id.btCancelar); //Añadimos el
botón de Cancelar.
//Configuramos la acción del botón "volver".
//Finalizamos la actividad Anadir e iniciamos la actividad Inicio.
```

```
botonCerrar.setOnClickListener(new View.OnClickListener() {
  public void onClick(View v) {
    Intent intent = new Intent(Anadir.this, Inicio.class); //Creamos el
Intent para cargar la actividad Inicio.
    startActivity(intent); //Iniciamos la actividad Inicio.
    finish(); //Finalizamos la actividad Añadir.
  }
});
```

Código 24. Control clase Anadir IV.

9.6.7 Control de edición de contactos

Para la edición de contactos, mostramos una pantalla en la que ofrecemos al usuario unos campos con la información que hemos recuperado del contacto seleccionado en cuestión, y la posibilidad de modificar la misma, esta información consta de nombre del contacto, número de teléfono y dirección de correo, para dicha recuperación de información utilizamos un objeto de la clase *Bundle* que nos permite saber cuál es el contacto seleccionado por el usuario y realizar la consulta a nuestra base de datos para recuperar la información sensible del mismo, en dicha recuperación de datos controlamos si hemos obtenido resultado alguno, como mostramos a continuación:

```
@Override
public void onCreate(Bundle savedInstanceState) {
  super.onCreate(savedInstanceState);
  setContentView(R.layout.editar);
  Bundle bundle = this.getIntent().getExtras();
//Recogemos el nombre seleccionado por el usuario que hemos pasado a
traves del Bundle, y lo almacenamos en una variable.
  String nombreRec = bundle.getString("NOMBRESEL");
//Abrimos la base de datos.
  abrirBasedatos();
//Consulta: contacto del teléfono seleccionado por el usurio con su
respectivo nombre, número de teléfono y dirección de correo electrónico.
  c = baseDatos.rawQuery("SELECT nombre,telefono,correo FROM contacto
WHERE nombre LIKE '"+ nombreRec +"' ", null);
//Almacenamos el resultado obtenido en la consulta en nuestras
variables.
//Nos aseguramos de que existe al menos un registro.
  if (c.moveToFirst()) {
//Recorremos el cursor hasta que no haya más registros.
    do{
      nombreCont = c.getString(0);
      telefonoCont = c.getString(1);
      correoCont = c.getString(2);
    } while(c.moveToNext());
  }
//Cerramos la base de datos.
  cerrarBasedatos();
(...)
```

Código 25. Control clase Editar I.

Controlamos que el usuario una vez modifique los campos o campo interesado, pulse el botón de guardado de información, y como hemos mencionado en el punto anterior, realizamos una serie de controles a la información modificada, tales como no dejar vacio el campo de nombre de contacto, que el número de teléfono solo tenga caracteres numéricos o el carácter “+”, que la cadena dirección de correo electrónico contenga el carácter “@” o sea la cadena por defecto “Email” y quitamos los espacios en blanco si los hubiera, para ello nos apoyamos del método *setOnClickListener* utilizando el código mostrado a continuación:

```
botonGuardar = (Button) findViewById(R.id.btGuardar); //Añadimos el
botón de Guardar.

//Configuramos la acción a realizar cuando el usuario pulse el botón
guardar.
//Guardar el contacto actual en la agenda
botonGuardar.setOnClickListener(new View.OnClickListener() {
  public void onClick(View v) {
    boolean nombcorrecto = true;
    boolean tlfcorrecto = true;
    boolean emlcorrecto = false;
//Comprobamos que el nombre del contacto sea correcto.
//Devolvemos un booleano indicando si es correcto o no.
    String nomb = editNombre.getText().toString();
    if ((nomb.compareTo("") == 0)||(nomb.compareTo("Nombre")== 0)){
      nombcorrecto = false;
    }
//Comprobamos que el número de teléfono y la dirección de email sean
correctas.
//Cada carácter del número de teléfono está comprendido entre "0" y "9"
o es el símbolo "+".
//Devolvemos un booleano indicando si es correcto o no.
    String tlf = editTelefono.getText().toString();
    for (int i=0; i<tlf.length(); i++){
      if ((((tlf.charAt(i)<='9') &&
(tlf.charAt(i)>='0'))||(tlf.charAt(i)=='+')) && (tlfcorrecto)){
        tlfcorrecto = true;
      }
      else
        tlfcorrecto = false;
    }
//Comprobamos que la dirección de email sea correcta.
//La cadena debe contener el carácter "@".
//Devolvemos un booleano indicando si es correcto o no.
    String eml = editCorreo.getText().toString();
    for (int i=0; i<eml.length(); i++){
      if (eml.charAt(i)=='@'){
        emlcorrecto = true;
      }
    }
//Evaluamos como correcto para el campo dirección de correo electrónico
la cadena vacía y la cadena "Email".
    if ((eml.compareTo("Email")== 0)||(eml.compareTo("")==0)){
      emlcorrecto = true;
    }
(...)
```

Código 26. Control clase Editar II.

A continuación y una vez modificados los campos interesados por el usuario, debemos realizar la actualización de los mismos en la base de datos de nuestra aplicación, para ello debemos controlar que dicha actualización se realiza correctamente como mostramos a continuación con el código siguiente:

```
//Si el campo número de teléfono y el campo dirección de correo
electrónico es correcto, procedemos a la actualización en nuestra base
de datos.
if ((tlfcorrecto)&&(emlcorrecto)&&(nombcorrecto)){
//Abrir la base de datos, se creará si no existe.
  abrirBasedatos();

//Quitamos los espacion en blanco si hay de la direccion de email.
  String email = new String();
  StringTokenizer stTexto = new
StringTokenizer(editCorreo.getText().toString());
  while (stTexto.hasMoreElements())
    email += stTexto.nextElement();

//Actualizamos la fila o registro en la tabla "contacto".
//si la actualización es correcta devolverá true.
  boolean resultado = actualizarFila(editNombre.getText().toString(),
editTelefono.getText().toString().trim(), email);

  if(resultado)
    (...)  //Si la actualización es correcta mostramos el mensaje de
contacto actualizado.
  else
    (...) //Si la actualización falla mostramos un mensaje de error.
//Cerramos la bases de datos.
  cerrarBasedatos();
}
else{
  (...)
}
//Método que realiza la actualización de los datos en nuestra tabla
contacto.
private boolean actualizarFila(String nombre, String telefono, String
correo) {
  ContentValues values = new ContentValues();
  values.put("nombre",nombre );
  values.put("telefono", telefono);
  values.put("correo", correo);
//Actualizamos el registro en la base de datos.
  try{
    baseDatos.update(tablaContacto, values, "nombre='"+nombreCont+"'",
null);
  }
  catch (Exception e) {
    return false;
  }
  Toast.makeText(getApplicationContext(), "Nombre: " + nombre + ", " +
"teléfono: " + telefono, Toast.LENGTH_LONG).show(); //Mostramos un
mensaje con los datos actualizados en la base de datos.
  return true;
}
```

Código 27. Control clase Editar III.

Como hemos mencionado anteriormente en éste punto, si alguno de los datos modificados por el usuario no pasase los controles pertinentes, mostramos a éste un mensaje informando del error, para ello utilizamos el siguiente código:

```
if ((tlfcorrecto)&&(emlcorrecto)&&(nombcorrecto)){
  (...)
}
else{
  if (!nombcorrecto)
    (...)  //Mostramos un mensaje de error al introducir el nombre del
contacto.
  else{
    if (!tlfcorrecto)
      (...)  //Mostramos un mensaje de error al introducir el número de
teléfono.
    else
      (...)  //Mostramos un mensaje de error al introducir la dirección
de correo electrónico.
  }
}
(...)
```

Código 28. Mensajes clase Editar.

Y para finalizar, en ésta actividad controlamos que el usuario haga uso de la cancelación de la misma, para ello dotamos al botón “Volver” de un *Listener* cuya función es la de iniciar la actividad “Inicio”, que resulta ser la actividad llamante, y por último finalizar la actividad actual. Esto lo realizamos mediante el siguiente código:

```
botonCerrar = (Button) findViewById(R.id.btCancelar); //Añadimos el
botón de Cancelar.
//Configuramos la acción del botón "volver".
//Finalizamos la actividad Editar e iniciamos la actividad Inicio.
botonCerrar.setOnClickListener(new View.OnClickListener() {
  public void onClick(View v) {
    Intent intent = new Intent(Editar.this, Inicio.class);
    startActivity(intent);
    finish();
  }
});
```

Código 29. Control clase Editar IV.

9.6.8 Control de Borrado de contactos

Otorgamos al usuario de nuestra aplicación la opción de realizar borrados de contactos, ya sea porque están desactualizados o por diversos motivos personales.

Para el borrado de contactos podemos acceder desde el menú contextual de cada contacto mencionado en apartados anteriores, para ello realizamos una selección prolongada en el contacto en cuestión y nos aparecerá dicho menú, y elegimos la opción de borrar.

A continuación nos aparecerá a modo de control, una ventana de confirmación para realizar de acción de borrado, al aceptarla, nuestra aplicación accederá a nuestra base de datos y borrará el contacto cuyo nombre coincida con el seleccionado, si esta operación es realizada con éxito, llamaremos a la actividad “Inicio” y finalizaremos la actividad actual para que se muestren reflejados los cambios en la *AgendaPersonal*, por contrapunto, también contemplamos que la acción falle, en tal caso nuestra aplicación informa al usuario de ello mediante un mensaje por pantalla.

Por el contrario si el usuario aborta la acción, la aplicación le muestra un mensaje reflejándolo y devolviendo el control a la actividad actual, a continuación mostramos los fragmentos de código para llevar a cabo estos controles:

```
case R.id.group_elemento5:
//Si el usuario selecciona que desea borrar el contacto.
//Mostramos un mensaje de confirmación antes de borrar el contacto.
AlertDialog.Builder alertDialog4 = new AlertDialog.Builder(Inicio.this);
alertDialog4.setMessage("¿Desea borrar el contacto?");
alertDialog4.setTitle("Borrando contacto...");
alertDialog4.setIcon(android.R.drawable.ic_dialog_alert);
alertDialog4.setCancelable(false);
alertDialog4.setPositiveButton("Sí", new
DialogInterface.OnClickListener() { //Si el usuario elige que sí.
  public void onClick(DialogInterface dialog, int which) {
    try {
      baseDatos = openOrCreateDatabase(nombreBD, MODE_WORLD_WRITEABLE,
null); //Abrimos la base de datos.
      baseDatos.delete(tablaContacto, "nombre='"+_itemSeleccionado+"'",
null); //Borramos el contacto seleccionado.
      baseDatos.close(); //Cerramos la base de datos.
      Intent intent = new Intent(Inicio.this, Inicio.class); //Creamos
el Intent para actualizar la lista llamando a la misma actividad Inicio.
      startActivity(intent); //Iniciamos la actividad Inicio para cargar
la lista actualizada.
      finish(); //Finalizamos la actividad.
    }
    catch (Exception e) {
      Toast.makeText(getApplicationContext(), "No se ha podido borrar el
contacto", Toast.LENGTH_LONG).show(); //Mostramos el mensaje de error si
no podemos borrar el contacto.
    }
  }
});
```

```
alertDialog4.setNegativeButton("No", new
DialogInterface.OnClickListener() { //Si el usuario elige la opción de
no borrar el contacto.
  public void onClick(DialogInterface dialog, int which) {
    Toast.makeText(getApplicationContext(), "Borrado cancelado",
Toast.LENGTH_LONG).show(); //Mostramos el mensaje al usuario de que la
operación ha sido cancelada.
  }
});
alertDialog4.show();
return true;
```

Código 30. Control borrar contacto.

9.6.9 Control de Backup de contactos en SD

Otra de las opciones otorgadas al usuario de nuestra aplicación es la de realizar una copia de seguridad de nuestros contactos en la tarjeta SD del dispositivo, para ello realizamos uno serie de controles.

Primeramente comprobamos la disponibilidad del almacenamiento externo del dispositivo, tanto que esté montado como que esté activada en modo escritura, para ello usamos el siguiente código apoyándonos en la clase *Environment* y dos variables *Boolean* que nos indican el estado del almacenamiento:

```
case R.id.backup:
  boolean mExternalStorageAvailable = false;
  boolean mExternalStorageWriteable = false;
  String state = Environment.getExternalStorageState();
  if (Environment.MEDIA_MOUNTED.equals(state)) {
    // Si podemos leer y escribir en el almacenamiento externo.
    mExternalStorageAvailable = mExternalStorageWriteable = true;
  } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
      // Si podemos leer únicamente en el almacenamiento externo.
      mExternalStorageAvailable = true;
      mExternalStorageWriteable = false;
      } else {
          // Si algo falla, no podemos ni leer ni escribir en el
almacenamiento externo.
          mExternalStorageAvailable = mExternalStorageWriteable = false;
      }
```

Código 31. Control backup contactos I.

A continuación comprobamos la ruta donde almacenaremos nuestro fichero de texto con la copia de todos los contactos, comprobamos tanto la existencia de dicha ruta como la del fichero propiamente dicho, si la ruta no existe, la creamos, en contrapunto, si el fichero existe, lo borramos para crear uno nuevo, para ello usamos el siguiente código:

```
if ((mExternalStorageAvailable)&&(mExternalStorageWriteable)){
//Si el usuario selecciona Realizar Backup.
//Código para realizar un backup de los contactos en un fichero de texto
localizado en la tarjeta SD del dispositivo.
```

```
File fichero = new
File(Environment.getExternalStorageDirectory().getPath()+"/PFCagenda/PFC
agenda.txt"); //Indicamos el fichero en el que realizamos el backup.
  File ruta = new
File(Environment.getExternalStorageDirectory().getPath()+"/PFCagenda/");
//Indicamos la ruta del fichero.
  try {
    if (!ruta.exists()){ //Si la ruta no existe la creamos.
      ruta.mkdir();
    }
    if (fichero.exists()){ //Si el fichero ya existe lo borramos.
      fichero.delete();
    }
    fichero.createNewFile(); //Creamos el fichero.
    FileWriter escritorFichero = new FileWriter(fichero);
(...)
```

Código 32. Control backup contactos II.

Para continuar, realizamos la consulta a nuestra base de datos de la que recogemos todos los datos de los contactos, comprobamos que la recuperación de datos haya sido la correcta como en puntos anteriores, y acto seguido, gracias a las clases *File* y *FileWriter* volcamos la información obtenida en nuestro fichero de texto localizado en el almacenamiento externo del dispositivo, para controlar cada contacto y el final del fichero usamos las cadenas de caracteres “###” y “####” respectivamente como hemos empleado en asignaturas a lo largo de la carrera, el siguiente código especifica los pasos a seguir:

```
(...)
baseDatos = openOrCreateDatabase(nombreBD, MODE_WORLD_WRITEABLE, null);
//Abrimos la base de datos.
c = baseDatos.rawQuery("SELECT nombre, telefono, correo FROM contacto
ORDER BY nombre", null); //Consulta: nombre, teléfono y correo de los
contactos de la base de datos ordenados por nombre.
//Nos aseguramos de que existe al menos un registro
  if (c.moveToFirst()) {
    //Recorremos el cursor hasta que no haya más registros
    int i=1;
      do {
        escritorFichero.write("Contacto número"+i+" :\n"); //Escribimos
el número del contacto en el fichero de texto.
        escritorFichero.write(c.getString(0)+"\n"); //Escribimos el
nombre del contacto en el fichero de texto.
        escritorFichero.write(c.getString(1)+"\n"); //Escribimos el
número de teléfono del contacto en el fichero de texto.
        escritorFichero.write(c.getString(2)+"\n"); //Escribimos el
correo del contacto en el fichero de texto.
        escritorFichero.write("###\n");//Marca para indicar fin del
contacto.
        i++;
      } while(c.moveToNext());
      escritorFichero.write("####");//Marca para indicar fin de fichero
contactos.
  }
  baseDatos.close(); //Cerramos base de datos.
  escritorFichero.flush(); //Liberamos el escritor.
  escritorFichero.close(); //Cerramos el escritor.
```

```
  Toast.makeText(getApplicationContext(), "Backup SD realizado...",
Toast.LENGTH_LONG).show(); //Mostramos el mensaje de que la acción se ha
realizado con éxito.
  }catch (IOException e) {
     Toast.makeText(getApplicationContext(), "No se ha podido realizar
el Backup...", Toast.LENGTH_LONG).show(); //Mostramos el mensaje de que
la acción ha fallado.
  }
}
else
  Toast.makeText(getApplicationContext(), "Error, el almacenamiento
externo o tarjeta SD no se encuentra disponible en estos momentos...",
Toast.LENGTH_LONG).show(); //Mostramos el mensaje de que el dispositivo
no tiene almacenamiento externo disponible.
break;

(...)
```

Código 33. Backup contactos.

Cabe destacar que para controlar tanto si se ha realizado el Backup de los contactos como si ha fallado el mismo o si el almacenamiento externo del dispositivo no estuviera disponible, el usuario recibe distintas notificaciones por pantalla para las diversas posibilidades como destacamos en el código anterior.

9.6.10 Acceso a los contactos del dispositivo móvil

Cuando *AgendaPersonal* solicita al dispositivo sus contactos, utilizamos el nombre del contacto como clave identificativa, recuperamos por un lado tanto el número de teléfono del mismo como por otro lado la dirección de correo electrónico del mismo si la tuviera. Toda esta información se obtiene de los contactos almacenados en el dispositivo móvil. Las tareas relacionadas con la manipulación de los contactos se llevan a cabo a través de la clase Carga.

Para acceder a la información de contactos y, en general, acceder a diferentes tipos de contenedores de información, se necesitan tres elementos:

- Un objeto a través del cual acceder al contenedor de información deseado. En este caso, se utiliza la clase *ContentResolver*, del paquete “android.content”, que permite entre otras cosas lanzar consultas sobre un determinado recurso.

- Un objeto que recoja el resultado de la consulta. La clase *Cursor*, del paquete “android.database”, permite almacenar y recorrer el resultado obtenido al realizar una consulta a un contenedor de información.

- Una referencia que indique a qué contenedor concreto se desea acceder. En Android, los contenedores de información suelen estar referenciados por una *URI* que los identifica y que permite su acceso concreto. En el caso que nos ocupa, se utiliza la clase Uri del paquete “android.net.Uri”.

La clase *ContentResolver* utilizada para tener acceso a los contactos pertenece a uno de los paquetes más importantes de Android, el paquete “android.content”. Contiene clases que permiten acceder y publicar diferentes tipos de contenidos. Algunas de sus clases más relevantes son las siguientes:

- ServiceConnection: permite monitorizar el estado de un componente *Service*.
- BroadcastReceiver: uno de los componentes básicos de una aplicación Android, que permite a una aplicación escuchar y atender *Intents*.
- ContentProvider: otro de los componentes básicos de Android. Facilita a las aplicaciones publicar sus contenidos y hacerlos accesibles a otras aplicaciones del sistema.
- ContentResolver: ya mencionado, posibilita el acceso a modelos de contenidos.
- Context: clase que representa el ámbito de ejecución de una aplicación.
- Intent: clase de vital importancia en Android, que representa una acción y sus datos asociados. Recuérdese que, mediante un *Intent*, se lanza una solicitud para que determinada acción sea llevada a cabo por la aplicación más adecuada entre todas las disponibles en el sistema (ver más adelante en este mismo capítulo).

En la clase *Carga* se define el acceso a la información de contactos. El método de acceso devuelve un objeto Cursor que contiene todos los contactos presentes en el dispositivo móvil.

El Código 34 muestra su código fuente:

```
public class Carga extends Activity {

  (...)

  try {
    //Las columnas que queremos en el resultado.
    String[] projection = new String[] {
      Data.DISPLAY_NAME,
      Phone.NUMBER};
    //Consulta: contactos del teléfono con sus respectivos números de
teléfono ordenados por nombre.
    Cursor datos = getContentResolver().query(
      Data.CONTENT_URI,
      projection,
      Data.MIMETYPE + "='" + Phone.CONTENT_ITEM_TYPE + "'",
      null,
      Data.DISPLAY_NAME + " ASC");
    startManagingCursor(datos);
    //Nos aseguramos de que existe al menos un registro.
    if (datos.moveToFirst()) {
      //Recorremos el cursor hasta que no haya más registros.
      do {
```

```
        int name = datos.getColumnIndexOrThrow(Data.DISPLAY_NAME);
        int number = datos.getColumnIndexOrThrow(Phone.NUMBER);
        String nombre = datos.getString(name);
        String telefono = datos.getString(number);

        //Insertamos en nuestra base de datos tanto el número de
teléfono como el nombre del contacto.
        ContentValues values = new ContentValues();
        values.put("nombre",nombre );
        values.put("telefono", telefono);
        values.put("correo", "Email");
        baseDatos.insert(tablaContacto, null, values);
      } while(datos.moveToNext());
    }
    //Las columnas que queremos en el resultado.
    String[] projection2 = new String[] {
      Data.DISPLAY_NAME,
      Email.DATA};
    //Consulta: contactos del teléfono con sus respectivos emails
ordenados por nombre.
    Cursor datos2 = getContentResolver().query(
      Data.CONTENT_URI,
      projection2,
      Data.MIMETYPE + "='" + Email.CONTENT_ITEM_TYPE + "'",
      null,
      Data.DISPLAY_NAME + " ASC");
    startManagingCursor(datos2);
    //Nos aseguramos de que existe al menos un registro.
    if (datos2.moveToFirst()) {
      //Recorremos el cursor hasta que no haya más registros.
      do {
        int name = datos2.getColumnIndexOrThrow(Data.DISPLAY_NAME);
        int email = datos2.getColumnIndexOrThrow(Email.DATA);
        String nombre = datos2.getString(name);
        String correo = datos2.getString(email);
        //Actualizamos la dirección de correo electrónico en la tabla de
contactos de nuestra base de datos.
        ContentValues values2 = new ContentValues();
        if (correo!=null){
          values2.put("correo", correo);
          baseDatos.update(tablaContacto, values2,
"nombre='"+nombre+"'", null);
        }
      } while(datos2.moveToNext());
    }
  } catch (Exception ex) {
      //Si falla la carga de contactos mostramos el siguiente mensaje al
usuario.
      Toast.makeText(getApplicationContext(), "No se ha podido cargar
los contactos del teléfono" , Toast.LENGTH_LONG).show();
  }
}
//cerramos la base de datos.
baseDatos.close();
//cargamos la siguiente actividad, en este caso Inicio y finalizamos la
actividad actual Carga.

(...)
```

```
}
```

Código 34. Acceso contactos dispositivo.

En primer lugar se construye un array de tipo *string* donde se especifican las columnas que se desean obtener en la consulta. La clase *ContactsContract.Data* representa una de las tablas que contiene información sobre los contactos, y forma parte de un interesante paquete llamado android.providers. Mediante este paquete se pueden referenciar las tablas y columnas de todos los contenedores de datos facilitados por Android, como pueden ser contactos, llamadas, fotografías, audios o vídeos, entre otros.

Para esta primera consulta se requieren dos columnas: el nombre del contacto, Data.*DISPLAY_NAME*, y su número de teléfono, Phone.*NUMBER*.

De momento, ya se ha indicado qué columnas interesan, pero no la tabla que debe ser consultada en busca de dichas columnas. El siguiente paso necesario es poder referenciar en la consulta a qué contenedor exacto vamos a consultar. Cada contenedor de datos de Android suele tener asociada una URI única que lo identifica. La constante Data.*CONTENT_URI* hace referencia a la URI que interesa para esta consulta, por lo que es almacenada en un objeto Uri.

Solamente resta lanzar la consulta al contenedor de contactos. El objeto ContentResolver nos permite utilizar el método query() donde, parámetro a parámetro, se puede construir una completa sentencia SELECT como se define en SQL estándar. Los parámetros del método query() son:

- La tabla que se desea consultar, identificada por un objeto Uri.
- Las columnas que se quieren obtener, presentes en el array previamente construido.
- Los filtros de selección deseados o cláusula WHERE, en este caso, el campo Data.*MIMETYPE* tiene que coincidir en tipo Phone.*CONTENT_ITEM_TYPE*.
- Los valores utilizados en la filtración o cláusula WHERE, ninguno en este caso.
- El orden del conjunto resultado: por nombre de contacto y en orden ascendente.

El resultado se aloja en un objeto Cursor, que permite recorrer y leer las filas obtenidas en la consulta. Para llevar a cabo este control del resultado, la clase Cursor incluye, entre otros, los siguientes métodos:

- getCount(): devuelve el número total de filas.
- getPosition(): devuelve la posición actual del cursor.

- getString(): devuelve una cadena con el valor de la columna indicada.
- moveToNext(): mueve el cursor a la siguiente fila.
- moveToFirst(): mueve el cursor a la primera fila.

A continuación insertamos en nuestra base de datos SQLite los datos obtenidos, en el siguiente punto de este mismo capítulo ahondaremos más en profundidad en este tema.

Para nuestra segunda consulta se requieren dos columnas: el nombre del contacto nuevamente, Data.*DISPLAY_NAME*, y su dirección de correo electrónico, Email.*DATA*.

Lanzamos la consulta al contenedor de contactos a diferencia que en esta ocasión las columnas que se quieren obtener son las anteriormente mencionadas y la cláusula WHERE el campo Data.*MIMETYPE* tiene que coincidir en tipo con Email.*CONTENT_ITEM_TYPE*.

Los datos recuperados en esta ocasión los actualizaremos en la base de datos de la aplicación, a diferencia de que en la primera consulta se realizó una inserción de los datos obtenidos.

El acceso a la información de contactos y, en general, a cualquier contenedor de información de Android requiere de los permisos necesarios expresados en el manifiesto. Así pues, en el fichero “AndroiManifest.xml” de *AgendaPersonal* debe figurar una línea como la siguiente:

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

Código 35. Declaración en el manifiesto del permiso recuperación de contactos.

9.6.11 Uso de la base de datos SQLite

Como hemos mencionado anteriormente, la información de los contactos almacenada en el dispositivo es recuperada e insertada en la base de datos de nuestra aplicación *AgendaPersonal*, dicha base de datos utilizada es la SQLite de Android, gracias a ella mostramos al usuario un listado con los contactos del dispositivo recuperados y las posteriores adiciones de otros a nuestra aplicación.

Para la gestión de nuestra base de datos la llevamos a cabo en las clases *Carga*, *Anadir* y *Editar* de la aplicación, en ellas se procede a las operaciones de creado, inserción y actualización respectivamente, de la información contenida en la misma.

En las clases *Inicio*, *VerContacto* y *Editar*, realizamos una serie de consultas sobre dicha base de datos, también resaltar que en Inicio podemos realizar el borrado de contactos de la misma.

El manejo de la base de datos SQLite es muy similar al que se hace cuando se accede, por ejemplo, a la información de los contactos almacenados en el propio dispositivo móvil, ya explicado en el apartado 9.6.10.

En la clase *Carga* se declaran dos variables globales de ámbito privado. La primera, *baseDatos*, representa un objeto de la clase SQLiteDatabase. Esta clase forma parte del paquete android.database.sqlite, que incluye otras muchas clases e interfaces para el manejo de SQLite. La variable *baseDatos* representa un controlador de la base de datos y proporciona métodos para la creación, consulta o actualización de la misma. La otra variable global, llamada *crearTablaContacto*, contiene la sentencia SQL necesaria para crear la base de datos en caso de ser la primera ejecución de *AgendaPersonal*.

9.6.11.1 Crear la base de datos SQLite

Como ya se ha mencionado, la variable *crearTablaContacto* contiene la instrucción en SQL necesaria para crear la base de datos. Esta implica una única tabla donde se almacena el código, nombre, número de teléfono y correo electrónico de cada contacto para el que se ha obtenido alguna vez información desde el dispositivo. Se puede comprobar su estructura mostrada a continuación:

```
private static final String crearTablaContacto =
      "CREATE TABLE IF NOT EXISTS contacto "+
      "(codigo INTEGER PRIMARY KEY AUTOINCREMENT, "+
      "nombre TEXT NOT NULL UNIQUE, "+
      "telefono TEXT NOT NULL UNIQUE, "+
      "correo TEXT);";
```

Código 36. Tabla contactos.

Cuando se llama al constructor de la clase *Carga*, lo primero que se hace es abrir la base de datos, de nombre `nombreBD = "agenda"`, utilizando para ello el contexto de la aplicación y el método openOrCreateDatabase(). Como es de suponer, en caso de no existir la base de datos con el nombre indicado, dicho método la crea.

A continuación, ya se dispone en la variable `baseDatos` de un controlador con el que manejar la base de datos. El siguiente paso es ejecutar el script SQL contenido en la variable *crearTablaContacto*. Así, si la base de datos ha tenido que ser creada, se creará también la tabla necesaria. En caso de existir, la sentencia SQL no produce ningún efecto. Para lanzar la sentencia se utiliza el método execSQL() de la clase SQLiteDatabase.

En el Código 36 se crea la base de datos SQLite. En el Código 37 se expone el constructor de *Carga*:

```
public class Carga extends Activity {

private SQLiteDatabase baseDatos;
private static final String nombreBD = "agenda";
private static final String tablaContacto = "contacto";
private static final String crearTablaContacto =
```

```
"create table if not exists "
+ " contacto (codigo integer primary key autoincrement, "
+ " nombre text not null unique, telefono text not null unique, correo
text);";

(...)

  //método para crear la actividad Carga.
  @Override
  public void onCreate(Bundle savedInstanceState) {

    (...)

    //Abrimos o creamos la base de datos de la agenda.
    baseDatos   =   openOrCreateDatabase(nombreBD,   MODE_WORLD_WRITEABLE,
null);

    //Intentamos realizar una consulta, si no salta excepción quiere
decir que no es la primera vez
    //que accedemos a la agenda por tanto ya esta la tabla creada y los
contactos del teléfono introducidos.
    try{
      c = baseDatos.rawQuery("SELECT nombre FROM contacto ORDER BY
nombre", null);
    } catch (Exception e) {
        baseDatos.execSQL(crearTablaContacto);

        (...)

    }

  }

}
```

Código 37. Creación tabla contactos.

Como hemos mencionado en puntos anteriores, antes de crear la base de datos controlamos que sea la primera vez que ejecutamos nuestra aplicación con una consulta sobre la misma, y por tanto nuestra base de datos no esté creada de antemano para dar más fluidez a *AgendaPersonal*, ya que al ser la primera vez, nuestra aplicación consulta la información de los contactos residentes en nuestro dispositivo y realiza la creación e inserción en nuestra base de datos de dicha información, estas operaciones demoran el tiempo de ejecución de la aplicación, con lo cual nos interesa controlarlo para una mejor experiencia al usuario.

El código 37 muestra este control, y más específicamente en el que sigue a continuación:

```
(...)
try{
  c = baseDatos.rawQuery("SELECT nombre FROM contacto ORDER BY nombre",
null);
} catch (Exception e) {
    baseDatos.execSQL(crearTablaContacto);
```

```
(...)}
```

Código 38. Control creación tabla contactos.

El acceso a las funciones de lectura o recuperación de contactos del dispositivo móvil requiere de su correspondiente permiso en la declaración del manifiesto. El permiso es el siguiente:

```
<uses-permission android:name="android.permission.READ_CONTACTS"/>
```

Código 39. Declaración en el manifiesto del permiso recuperación de contactos.

9.6.11.2 Consulta, actualización e inserción de filas

- **Consultas:**

Dentro de las clases *Carga*, *Inicio*, *Anadir*, *Editar* y *VerContacto*, existen métodos que interactúan con la base de datos. Primero vamos a ver el tipo de consultas que utilizamos para nuestra aplicación ya que nos parece interesante mostrar los dos tipos de consultas que usamos.

En primer lugar en nuestra clase *Inicio* usamos una consulta total para recuperar la información de todos los contactos almacenados en la base de datos “SQLite” de nuestra aplicación, así como también utilizamos este procedimiento para la realización de la copia de seguridad de los contactos en el almacenamiento externo del dispositivo, para ello mostramos el código con el que realizamos la consulta:

```
public class Inicio extends Activity {

   (...)

  baseDatos = openOrCreateDatabase(nombreBD, MODE_WORLD_WRITEABLE,
null);//Abrimos la base de datos.
  c = baseDatos.rawQuery("SELECT nombre FROM contacto ORDER BY nombre",
null); //Consulta: nombre de los contactos de la base de datos ordenados
por nombre.
  //Nos aseguramos de que existe al menos un registro
  if (c.moveToFirst()) {
    //Recorremos el cursor hasta que no haya más registros
    do {
      data.addData(new ListItemObject(c.getString(0)));
    } while(c.moveToNext());
  }
  baseDatos.close(); //Cerramos la base de datos.

   (...)

}
```

Código 40. Consulta clase Inicio.

Como podemos observar para mostrar al usuario un listado de todos los contactos almacenados en *AgendaPersonal* utilizamos el método rawQuery() de la clase *SQLiteDatabase* de Android que nos permite lanzar una consulta, de forma que mediante sus parámetros se construye una sentencia SELECT completa: tabla a consultar, columnas que se desea devolver, posible cláusula WHERE y orden de las filas.

En esta ocasión para obtener el nombre de todos los contactos almacenados en la base de datos de la aplicación, utilizamos la sentencia SQL:

```
SELECT nombre FROM contacto ORDER BY nombre
```

Código 41. Sentencia SQL clase Inicio.

En segundo lugar en nuestras clases *VerContacto* y *Editar* usamos una consulta parcial para recuperar la información del contacto seleccionado por el usuario almacenado en la base de datos "SQLite" de nuestra aplicación, para su realización mostramos el código con el que realizamos la consulta:

```
public class VerContacto extends Activity {

  (...)
  Bundle bundle = this.getIntent().getExtras();
  //Recogemos el nombre seleccionado por el usuario que hemos pasado a
traves del Bundle, y lo almacenamos en una variable.
  String nombreRec = bundle.getString("NOMBRESEL");
  //Abrimos la base de datos.
  abrirBasedatos();
  //Consulta: contacto del teléfono seleccionado por el usuario con su
respectivo nombre, número de teléfono y dirección de correo electrónico.
  c = baseDatos.rawQuery("SELECT nombre,telefono,correo FROM contacto
WHERE nombre LIKE '"+ nombreRec +"' ", null);
  //Almacenamos el resultado obtenido en la consulta en nuestras
variables.
  //Nos aseguramos de que existe al menos un registro.
  if (c.moveToFirst()) {
  //Recorremos el cursor hasta que no haya más registros.
    do {
      nombreCont = c.getString(0);
      telefonoCont = c.getString(1);
      correoCont = c.getString(2);
    } while(c.moveToNext());
  }
  //Cerramos la base de datos.
  cerrarBasedatos();

  (...)}
```

Código 42. Consulta clase VerContacto.

Observamos que utilizamos el método rawQuery() de la clase *SQLiteDatabase* de Android para lanzar la consulta exactamente igual que en el caso anterior, a diferencia que para este tipo de consultas utilizamos la sentencia SQL:

```
SELECT nombre, telefono, correo FROM contacto WHERE nombre LIKE
'nombreRec'
```

Código 43. Sentencia SQL clase VerContacto.

En este caso la cláusula WHERE existe, y lleva asociada el nombre del contacto seleccionado por el usuario de la aplicación para localizar la información del mismo en la base de datos SQLite.

Estas consultas arrojarán su resultado sobre un objeto de la clase *Cursor*. En caso de devolver alguna fila de la base de datos, según indiquen los controles realizados con los métodos moveToFirst() y moveToNext() de esta misma clase, significa que la recuperación de información ha sido satisfactoria y podemos operar con el resultado obtenido.

- **Inserción:**

El usuario puede añadir contactos a la *AgendaPersonal* mediante la inserción de los mismos en la base de datos de la aplicación, para ello usamos el método insertarFila() de la clase *Anadir*, a continuación vamos a mostrar el código empleado:

```
//Método que realiza la inserción de los datos en nuestra tabla
contacto.
private boolean insertarFila(String nombre, String telefono, String
correo) {
      ContentValues values = new ContentValues();
      values.put("nombre",nombre );
      values.put("telefono", telefono);
      values.put("correo", correo);
      Toast.makeText(getApplicationContext(), "Nombre: " + nombre + ", "
+ "teléfono: " + telefono, Toast.LENGTH_LONG).show(); //Mostramos un
mensaje con los datos introducidos en la base de datos.
      return (baseDatos.insert(tablaContacto, null, values) > 0);
}
```

Código 44. Inserción clase Anadir.

La aplicación solicita al usuario los datos del nuevo contacto y mediante el método anteriormente mencionado y la clase *ContentValues* realizamos la inserción de los datos.

- **Actualización:**

El usuario puede actualizar contactos anteriormente almacenados en *AgendaPersonal* mediante la actualización de los mismos en la base de datos de la aplicación, para ello usamos el método actualizarFila() de la clase *Editar*, a continuación vamos a mostrar el código empleado:

```
//Método que realiza la actualización de los datos en nuestra tabla
contacto.
```

```
private boolean actualizarFila(String nombre, String telefono, String
correo) {

  ContentValues values = new ContentValues();
  values.put("nombre",nombre );
  values.put("telefono", telefono);
  values.put("correo", correo);
  //Actualizamos el registro en la base de datos.
  try{
    baseDatos.update(tablaContacto, values, "nombre='"+nombreCont+"'",
null);
  } catch (Exception e) {
      return false;
  }
  Toast.makeText(getApplicationContext(), "Nombre: " + nombre + ", " +
"teléfono: " + telefono, Toast.LENGTH_LONG).show(); //Mostramos un
mensaje con los datos actualizados en la base de datos.
  return true;
}
```

Código 45. Actualización clase Editar.

La aplicación muestra al usuario los datos del contacto seleccionado, y le ofrece la oportunidad de modificar los mismos, mediante el método anteriormente mostrado y la clase *ContentValues* realizamos la actualización de los datos.

En cualquier caso, es necesario agrupar de alguna forma los valores para la tupla, ya sea para su inserción o su actualización. Dentro del paquete android.content (aquel importante que contiene clases como *Intent* o *BroadcastReceiver*) existe una clase llamada *ContentValues* que permite asociar pares del tipo campo-valor perfectos para su utilización en bases de datos como SQLite. De esta forma, se compone un objeto ContentValues con el nombre, número de teléfono y la dirección de correo electrónico del contacto para su utilización en insert() o update().

9.6.11.3 Cuándo insertar o actualizar la información de los contactos en SQLite

Como recordaremos, la información almacenada de los contactos en una primera instancia la recopilamos del dispositivo, creamos la base de datos *SQLite* de la aplicación e insertamos la información recuperada mencionada anteriormente, este proceso se lleva a cabo en la clase *Carga* de *AgendaPersonal*, en la que realizamos la inserción de la información gracias a las clases *Cursor*, *SQLiteDatabase* y *ContentValues* como ya mencionamos en puntos anteriores, más el método insert().

También el usuario puede insertar información de nuevos contactos mediante la actividad *Anadir* de nuestra aplicación, en esta clase realizamos una inserción de los datos suministrados por el usuario como son el nombre, número de teléfono y dirección de correo electrónico del contacto.

Para realizar las actualizaciones de datos residentes en la base de datos *SQLite* otorgamos al usuario la posibilidad de modificar los mismos mediante la actividad *Editar* ya que la información contenida en la aplicación puede cambiar con el paso del tiempo, no volvemos a apoyar en las clases mencionadas anteriormente en este punto para el tratamiento de la misma mas el método update() en vez de insert(). Cabe añadir que en la clase *Carga* para suministrar la información relacionada con las direcciones de correo electrónico de los contactos realizamos una actualización con dicha información recuperada del dispositivo ya que originalmente en el campo *Correo* de nuestra base de datos insertamos la cadena de caracteres “Email”.

9.6.12 Construcción del menú principal

Para poder acceder a algunas de las opciones ofrecidas por *AgendaPersonal*, el usuario debe utilizar el menú principal de la aplicación. Este menú aparece al pulsar la tecla correspondiente en el dispositivo móvil.

Las clases *Inicio* y *VerContacto* derivan de la clase Activity y tienen por ello un ciclo de vida bien definido. Además, cuentan con otra serie de métodos nativos que permiten tomar el control de diversos eventos provocados por el usuario. Este es el caso de la utilización del menú principal, cuyo comportamiento viene determinado por los métodos onCreateOptionsMenu() y onMenuItemSelected().

9.6.12.1 Crear un menú

El método onCreateOptionsMenu() es invocado cuando se pulsa la tecla correspondiente al menú, y permite configurar el tipo de menú que se quiere desplegar.

Este menú viene representado generalmente por una ventana que se desliza sobre la interfaz mostrada en ese momento y que debe ofrecer las funcionalidades principales de la aplicación en curso, que en el caso de *AgendaPersonal* son las siguientes:

- En la clase *Inicio*:
  - Añadir contacto.
  - Realizar Backup en SD.
  - Salir de la aplicación.

- En la clase *VerContacto*:
  - Llamar al contacto.
  - Enviar SMS al contacto.
  - Y por último enviar email al contacto.

En el siguiente fragmento de código se enseña al lector la forma en la que se compone el menú principal de la clase *Inicio* de *AgendaPersonal*.

```
public class Inicio extends ListActivity {

(...)

  //Método para la configuración del menú estandar de la actividad
Inicio, se activa mediante el botón "menu" del dispositivo.
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
      MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.menu2, menu);
      return true;
  }

(...)

}
```

Código 46. Clase Inicio crear menú.

Y su correspondiente archivo “.xml” en el que están definidos los elementos mostrados por pantalla al usuario, como son el nombre del botón y su correspondiente icono, en este caso el archivo se llama “menu2.xml” contenido en el directorio “PFC/res/menu” de nuestro proyecto y contiene el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/anadir_contacto"
          android:icon="@drawable/plus"
          android:title="@string/anhadir_contacto" />
    <item android:id="@+id/backup"
          android:icon="@drawable/backup"
          android:title="@string/realizar_backup" />
    <item android:id="@+id/salir"
          android:icon="@drawable/exit"
          android:title="@string/salir" />
</menu>
```

Código 47. Código XML del menú de la clase Inicio.

En el siguiente fragmento de código se enseña al lector la forma en la que se compone el menú principal de la clase *VerContacto* de *AgendaPersonal*.

```
public class VerContacto extends Activity {

(...)

  //Método para la configuración del menú estandar de la actividad
VerContacto, se activa mediante el botón "menu" del dispositivo.
  @Override
  public boolean onCreateOptionsMenu(Menu menu) {
      MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.menu3, menu);
      return true;
  }

(...)

}
```

Código 48. Clase VerContacto crear menú.

Y su correspondiente archivo “.xml” en el que están definidos los elementos mostrados por pantalla al usuario, como son el nombre del botón y su correspondiente icono, en este caso el archivo se llama “menu3.xml” contenido en el directorio “PFC/res/menu” de nuestro proyecto y contiene el siguiente código:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/llamar"
          android:icon="@drawable/phone"
          android:title="@string/llamar" />
    <item android:id="@+id/sms"
          android:icon="@drawable/sms"
          android:title="@string/enviarsms" />
    <item android:id="@+id/email"
          android:icon="@drawable/email"
          android:title="@string/enviarcorreo" />
</menu>
```

Código 49. Código XML del menú de la clase VerContacto.

Desde las actividades de nuestra aplicación agenda personal llamamos a los archivos “.xml” en los que indicamos que elementos queremos mostrar al usuario junto con la disposición, atributos y características de los mismos, más adelante ahondaremos más en profundidad en este tema, ya que para mostrar la interfaz de la aplicación al usuario son utilizados varios archivos de este tipo.

La clase *Menu* representa el menú principal de una actividad y forma parte del paquete android.view, que ofrece un considerable número de clases para manejar los

elementos de una interfaz de usuario y gestionar los eventos asociados. A través del objeto Menu se gestionan las opciones que componen el menú principal.

La clase *MenuInflater* (también del paquete android.view) se utiliza para instanciar objetos del menú de los archivos XML preparados para ello.

Los ficheros XML de menú se deben colocar en la carpeta “res\menu” de nuestro proyecto y tendrán una estructura análoga mostrada en el código anterior, con sus elementos y atributos que vamos a detallar a continuación:

- Etiqueta <*menu*>: es la empleada para indicar el elemento menú que vamos a configurar.
```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
...
</menu>
```

- Etiqueta <*item*>: es la que empleamos para indicar los elementos que van a componer el menú, dentro de esta etiqueta indicaremos con sus atributos las características que queremos que tengas los elementos del menú.
```
<item.../>
```

- Atributo *android:id* es el atributo que asigna un identificador unívoco al elemento.
```
android:id="..."
```

- Atributo *android:title* es el atributo que asigna texto al elemento.
```
android:title="..."
```

- Atributo *android:icon* es el atributo que asigna el icono al elemento.
```
android:title="..."
```

Como vemos, la estructura básica de estos ficheros es muy sencilla. Tendremos un elemento principal <*menu*> que contendrá una serie de elementos <*item*> que se corresponderán con las distintas opciones a mostrar en el menú. Estos elementos <*item*> tendrán a su vez varias propiedades básicas, como su ID (*android:id*), su texto (*android:title*) o su icono (*android:icon*). Los iconos utilizados deberán estar por supuesto en las carpetas “res\drawable” de nuestro proyecto.

Las opciones declaradas para el menú principal cuentan con un texto asociado que las describe y una imagen o icono representativo, ambos visibles para el usuario cuando despliegue el menú. Para lo primero, el texto, se utiliza el texto declarado en el fichero de recursos “strings.xml”, dentro de la carpeta “\res\values” de nuestro proyecto.

Para las imágenes, al igual que con el texto, se incluyen haciendo una referencia a la carpeta de recursos “\res\drawable”, donde deberán estar ubicadas. Como información de utilidad al lector, se dirá que el SDK de Android incluye por defecto una importante

colección de iconos útiles para todo tipo de aplicaciones. Estos iconos se encuentran disponibles en la carpeta “\tools\lib\res\default\drawable”.

9.6.12.2 Controlar la opción seleccionada

En este punto, el usuario ya puede desplegar el menú y comprobar que opciones principales le ofrece *AgendaPersonal*. Ahora es necesario poder conocer qué opción pulsa el usuario y actuar en consecuencia.

El otro método importante para controlar el menú y sus opciones es onMenuItemSelected(). Cada vez que el usuario selecciona una opción del menú principal, este método se llama recibiendo como parámetro un objeto MenuItem que va a permitir conocer qué opción exactamente ha sido la pulsada.

Utilizando una sencilla sentencia de control switch y el método getItemId() de MenuItem, se obtiene la opción pulsada y se ejecuta la acción correspondiente a la misma, tal y como se hace en el Código 50 correspondiente a la clase *Inicio*.

```
public class Inicio extends ListActivity {

//Método para configurar la acción a desarrollar por cada elemento del
menú.
@Override
public boolean onMenuItemSelected(int featureId, MenuItem item) {
  super.onMenuItemSelected(featureId, item);
    switch(item.getItemId()) {

      case R.id.anadir_contacto:

        (...)

        break;

      case R.id.backup:

        (...)

        break;

      case R.id.salir:

        (...)

        break;

  }

}
```

Código 50. Control opción seleccionada onMenuItemSelected clase Inicio.

O en el Código 51 correspondiente a la clase *VerContacto*.

```
public class VerContacto extends Activity {

//Método para configurar la acción a desarrollar por cada elemento del
menú.
@Override
public boolean onMenuItemSelected(int featureId, MenuItem item) {
  super.onMenuItemSelected(featureId, item);
    switch(item.getItemId()) {

      case R.id.llamar:

        (...)

        break;

      case R.id.sms:

        (...)

        break;

      case R.id.email:

        (...)

        break;

  }

}
```

Código 51. Control opción seleccionada onMenuItemSelected clase VerContacto.

9.6.13 Construcción del menú contextual

Para poder acceder a algunas de las opciones ofrecidas por *AgendaPersonal*, el usuario debe utilizar el menú contextual de la aplicación. Este menú aparece al mantener pulsado un contacto de nuestra lista durante un breve periodo de tiempo.

La clase *Inicio* deriva de la clase *ListActivity* y tienen por ello un ciclo de vida bien definido. Además, cuenta con otra serie de métodos nativos que permiten tomar el control de diversos eventos provocados por el usuario. Este es el caso de la utilización del menú contextual, cuyo comportamiento viene determinado por los métodos onCreateContextMenu() y onContextItemSelected().

9.6.13.1 Crear un menú contextual

El menú contextual es conceptualmente similar al menú que aparece cuando el usuario hace un "clic derecho" en un PC. Se debe usar un menú contextual para proporcionar al usuario el acceso a las acciones que pertenecen a un elemento específico

de la interfaz de usuario. En Android el menú contextual se muestra cuando el usuario realiza una "pulsación larga" sobre un elemento.

Podemos crear un menú contextual de cualquier vista, aunque este tipo de menús se suelen utilizar para los elementos de un *ListView* (como es el caso de nuestra aplicación *AgendaPersonal*). Cuando el usuario realiza una pulsación larga sobre un elemento del *ListView* y este último está preparado para proporcionar un menú contextual, el usuario recibe como señal (de que dicho menú se encuentra disponible para el elemento) una aminación del color de fondo del elemento que consta de una transicion del color naranja a blanco antes de que se desplegue el menú contextual.

Para proporcionar un menú contextual a cualquier vista, usamos el método registerForContextMenu(), a este método le pasamos la vista a la que se desea proporcionar dicho menú. Cuando esta vista reciba una pulsación larga en uno de sus elementos, se mostrará el menú contextual.

Como hemos mencionado anterioremente, para definir la apariencia del menú contextual y las funcionalidades que proporcionaremos al usuario con dicho menú, usaremos los métodos onCreateContextMenu() y onContextItemSelected().

A continuación mostraremos el código utilizado en nuestra aplicación para desarrollar los puntos mencionados anteriormente:

```
public class Inicio extends ListActivity {
(...)
  //Método para crear la actividad Inicio.
  @Override
  public void onCreate(Bundle savedInstanceState) {
    (...)
    registerForContextMenu(getListView());
  }
(...)
  //Método para la creación del menú contextual que se lanza si el
usuario deja presionado un contacto.
  @Override
  public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    AdapterView.AdapterContextMenuInfo info =
(AdapterView.AdapterContextMenuInfo)menuInfo;

    //Almacenamos el elemento que ha seleccionado el usuario.
    _itemSeleccionado =
lv.getAdapter().getItem(info.position).toString();

    //Ponemos el nombre del elemento seleccionado como título del menú.
menu.setHeaderTitle(lv.getAdapter().getItem(info.position).toString());
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.menu, menu);
  }

}
```

Código 52. Clase Inicio crear menú contextual.

Siendo el archivo “menu.xml” usado para la construcción del menú contextual, localizado en el directorio “/PFC/res/menu” de nuestra aplicación, el siguiente:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:id="@+id/group">
      <item android:id="@+id/group_elemento1"
            android:onClick="onGroupItemClick"
            android:title="@string/llamar"/>
        <item android:id="@+id/group_elemento2"
              android:onClick="onGroupItemClick"
              android:title="@string/enviarsms"/>
        <item android:id="@+id/group_elemento3"
              android:onClick="onGroupItemClick"
              android:title="@string/enviarcorreo"/>
        <item android:id="@+id/group_elemento4"
              android:onClick="onGroupItemClick"
              android:title="@string/editar_contacto"/>
        <item android:id="@+id/group_elemento5"
              android:onClick="onGroupItemClick"
              android:title="@string/borrar_contacto"/>
    </group>
</menu>
```

Código 53. Código XML del menú contextual de la clase Inicio.

9.6.13.2 Controlar la opción seleccionada

Cuando el usuario selecciona un elemento en el menú contextual, el sistema realiza la llamada al método onContextItemSelected(). Veremos cómo se pueden manejar los elementos seleccionados a continuación.

Para asignarle las funcionalidades (las cuales explicaremos con más detalle en puntos posteriores) a las distintas opciones que el menú contextual ofrece al usuario utilizamos el método mencionado anteriormente:

```
//Método para configurar la acción a realizar cuándo un elemento del
menú es seleccionado.
@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
      case R.id.group_elemento1:

        (...)

        return true;

      case R.id.group_elemento2:

        (...)
```

```
        return true;
      case R.id.group_elemento3:
        (...)
        return true;
      case R.id.group_elemento4:
        (...)
        return true;
      case R.id.group_elemento5:
        (...)
        return true;
```

Código 54. Control opción seleccionada onContextItemSelected clase Inicio.

La estructura del código anterior para el menú contextual es similar a la del menú de opciones de la aplicación, en el que con el método getItemId() y una sentencia de control switch, se obtiene la opción pulsada por el usuario y se ejecuta la acción correspondiente a la misma.

9.6.14 Mostrar listado de contactos y la información del contacto seleccionado

Acciones como mandar un SMS, un correo electrónico o realizar una llamada se realizan sobre el contacto que en ese momento tenga el usuario seleccionado en la aplicación. El usuario puede cambiar el contacto objetivo a otro pulsando sobre ellos y utilizando un listado con todos los contactos ordenados alfabéticamente. Éste último caso es el que se explica en este apartado.

Cualquier aplicación en Android está formada por una serie de componentes básicos entre los que se encuentraa *Activity*. Una clase *Activity*, como ya sabemos, representa una funcionalidad importante, con entidad propia, y que está asociada generalmente a una interfaz con la que el usuario puede interactuar. La clase *Inicio*, por ejemplo, es una *Activity* que muestra listado con los contactos del dispositivo mencionados anteriormente y permite al usuario realizar una serie de acciones. Una *Activity* también puede invocar a su vez a otras Actividades.

El listado de contactos consiste en una lista a pantalla completa donde se ordenan alfabéticamente todos los contactos y donde el usuario pulsa sobre uno de ellos, este último puede realizar dos tipos de selecciones de contacto, mediante una pulsación corta o mediante una pulsación prolongada para invocar un menú contextual también explicado, en el primer caso se le mostrará toda la información del contacto seleccionado

como ya explicásemos anteriormente. Toda esta acción se enmarca dentro de un nuevo componente *Activity* o nueva Actividad, ya que representa una funcionalidad importante con su propia y diferenciada interfaz.

9.6.14.1 Lanzar una nueva Activity

La clase *Inicio* extiende a la clase Activity y es la encargada de componer la lista de contactos y capturar cuál de ellos es pulsado. En primer lugar, a través del Código 55 se enseña cómo se lanza esta nueva *Activity* desde la clase *Carga* que, como se recordará, es la clase que inicia a aplicación y comprueba que la base de datos desde las que trabaja *AgendaPersonal* se encuentre debidamente montada y contenga la información sensible de los propios contactos.

```
package PFC.Agenda;

(...)

public class Carga extends Activity {

(...)

@Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
      //cargamos el layout de la actividad.
      setContentView(R.layout.main);
      //creamos un hilo para cargar los contactos del teléfono.
      new Handler().postDelayed(new Runnable(){
        public void run() {

        (...)

      //cargamos la siguiente actividad, en este caso Inicio y
finalizamos la actividad actual Carga.

      Intent intent = new Intent(Carga.this, Inicio.class);

      startActivity(intent);

      finish();

      }

    }, 2500);

  }

}
```

Código 55. Lanzamiento de la actividad Inicio desde la clase Carga.

Toda nueva *Activity* debe ser lanzada a través de un *Intent*, que especifica qué es lo que se quiere hacer y con qué datos debe hacer. Habitualmente, un *Intent* es una forma de delegar determinadas acciones en otras aplicaciones instaladas en el sistema y que,

obviamente, tengan la capacidad para llevarlas a cabo. Sin embargo, también es posible especificar que un *Intent* debe ser realizado por una determinada instancia de una clase.

Es lo que se pretende hacer en este caso con el listado de contactos, ver Figura 19.Al crear un objeto de la clase Intent se especifica en el constructor que dicho *Intent* debe ser realizado en la clase *Carga*. De esta forma, ya se ha configurado qué es lo que se quiere hacer (ejecutar la clase *Inicio* que es la encargada de mostrar la lista de contactos como ya hemos mencionado anteriormente).

Figura 19. Lista de contactos.

A continuación vamos a indicar el código de la clase *Inicio* para mostrar la información del contacto seleccionado por el usuario mediante la clase *VerContacto* también perteneciente a la clase *Activity*.

```
package PFC.Agenda;

(...)

public class Inicio extends ListActivity {

  (...)

  //Método para crear la actividad Inicio.
  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //Abrimos la base de datos.

    baseDatos = openOrCreateDatabase(nombreBD, MODE_WORLD_WRITEABLE,
null);
```

```
    //Consulta: nombre de los contactos de la base de datos ordenados
por nombre.

    c = baseDatos.rawQuery("SELECT nombre FROM contacto ORDER BY
nombre", null);
    //Nos aseguramos de que existe al menos un registro
    if (c.moveToFirst()) {
      //Recorremos el cursor hasta que no haya más registros
      do {
        data.addData(new ListItemObject(c.getString(0)));
      } while(c.moveToNext());
    }
    baseDatos.close(); //Cerramos la base de datos.
    Map<String, ArrayList<ListItemObject>> sortedContacts =
data.getSortedData();
    SeparatedListAdapter adapter = this.makeAdapter(sortedContacts);
    this.setListAdapter(adapter);
    this.setContentView(R.layout.inicio);

    lv = getListView(); //Obtenemos el ListView.
    lv.setTextFilterEnabled(true);
    lv.setFastScrollEnabled(true); //Método que nos permite desplazarnos
más rapidamente por a lista de contactos.

    registerForContextMenu(getListView());

  //Configuramos el Listener. En caso de que el usuario seleccione un
contacto lanzamos la subactividad VerContacto.
    OnItemClickListener clickListener = new OnItemClickListener() {
      public void onItemClick(AdapterView<?> parent, View view, int
position, long id) {
        Intent intent = new Intent(Inicio.this, VerContacto.class);
        Bundle bundle = new Bundle();
        bundle.putString("NOMBRESEL", ((TextView)
view).getText().toString());
        intent.putExtras(bundle);
        startActivityForResult(intent, RESULT_OK);
      }
    };

  //Activamos el Listener creado anteriormente.

  lv.setOnItemClickListener(clickListener);

  }

  (...)

}
```

Código 56. Lanzamiento de la actividad VerContacto desde la clase Inicio.

Lo primero de todo es mencionar que, tal y como muestra el código anterior, la clase *Inicio* no extiende directamente a la clase *Activity*, sino que extiende de la clase *ListActivity*. Esta clase es derivada de Activity y, está específicamente adaptada para

mostrar listados a pantalla completa donde el usuario ha de elegir uno de los elementos. Además, la clase *ListActivity* tiene asociado un objeto *ListView*, que representa uno de los muchos diseños para interfaces de usuario predefinidas en Android y que ayuda en la tarea de componer y mostrar un listado vertical de elementos.

ListActivity tiene un método exclusivo al resto de clases derivadas de *Activity* llamado “setListAdapter()” que construye con los valores dados el listado final que será visible para el usuario a pantalla completa. Para ello, es necesario transformar antes la lista “ListItemContainer<ListItemObject> data” en un objeto de la clase *ListAdapter* o derivado como es en nuestro caso, un objeto de la clase *SeparatedListAdapter*.

En nuestra aplicación para hacer más vistosa la agenda de contactos, hemos utilizado una serie de *ListAdapter* personalizados como es el mencionado anteriormente *SeparatedLidtAdapter* y cuyo código es el siguiente:

```
@SuppressWarnings("unchecked")
      public <T extends ListItemInterface> SeparatedListAdapter
makeAdapter(Map<String, ArrayList<T>> sortedObjects)
    {
        Iterator it = sortedObjects.entrySet().iterator();
        SeparatedListAdapter adapter = new SeparatedListAdapter(this);
        String label = null;
        ArrayList<T> section = new ArrayList<T>();
        while (it.hasNext()) {
            Map.Entry<String,ArrayList<T>> pairs =
(Map.Entry<String,ArrayList<T>>) it.next();
            section =  pairs.getValue();
            label = pairs.getKey();
            adapter.addSection(label, new ArrayAdapter<T>(this,
R.layout.list_item, section));
        }
        return adapter;
    }
```

Código 57. Uso de la clase SeparatedListAdapter.

Cabe mencionar que en nuestro caso con la utilización de este tipo de *ListAdapter* como hemos mostrado en el código anterior, nos apoyamos también en las clases *ListItemContainer*, *ListItemObject* y *ListItemInterface* que nos proporcionan la interfaz mostrada en la Figura 19, el contenido de estas clases lo mostraremos en el anexo correspondiente en este mismo proyecto. Además el código del correspondiente archivo “inicio.xml” construye la interfaz de usuario mencionada anteriormente.

```
<?xml version="1.0" encoding="utf-8"?>
<ListView
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@android:id/list"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:drawSelectorOnTop="false"
      android:fastScrollEnabled="true"/>
```

Código 58. inicio.xml

Ya se ha construido y mostrado el listado al usuario, pero ahora es necesario saber qué hacer cuando el usuario seleccione uno de los nombres. El método "onListItemClick()" de *ListActivity* se activa cuando tal evento sucede.

Ahora se debe indicar con qué datos se desea mostrar la información sensible del contacto seleccionado, esto es lo que se va a pasar como datos asociados a dicho *Intent* que es el nombre de dicho contacto presente en un objeto de la clase *Bundle*.

La clase *Bundle* permite asociar muchos tipos básicos de datos para poder ser pasados a un objeto de la clase *Intent*. Una peculiaridad común a cada dato asociado a un *Bundle*, es que debe ir acompañado de una etiqueta textual que lo identifique. Por ello, al método "putString()" se le asigna tanto el nombre del contacto como la etiqueta identificativa "NOMBRESEL" al objeto de la clase *Bundle* en cuestión.

La clase *Intent* permite asociar objetos de la clase *Bundle* mencionada anteriormente para poder ser procesados en la clase o aplicación donde vaya a ser atendido el *Intent*. Por ejemplo, en esta situación concreta se ha utilizado el método "putExtras()" para asociar el objeto *Bundle* que contiene la cadena de texto correspondiente al contacto seleccionado.

El *Intent* ya ha sido construido: representa una acción específica (ejecutar la clase *VerContacto*) y unos datos asociados (el nombre del contacto seleccionado). Únicamente resta lanzar la *Activity*. Existen dos formas para lanzar una nueva actividad en Android. Si se quiere lanzar la *Activity* sin más, sin esperar ningún resultado final, se utiliza el método "startActivity()". Si se espera un valor devuelto por la *Activity*, entonces se lanza con el método "startActivityForResult()", de forma que al terminar dicha actividad se invocará el método "onActivityResult()", que permite manejar el resultado obtenido. Como en este caso se desea enviar qué contacto ha sido seleccionado en la lista, se utilizará el segundo de los métodos posibles.

Invocar el método "startActivityForResult()" es muy sencillo. Simplemente se indica el *Intent* que sea desea lanzar y se le asocia un código identificativo para poder diferenciar en "startActivityForResult()" cuál es la *Activity* que ha finalizado. Es decir, todas las actividades lanzadas con "startActivityForResult()" acaban en el mismo método "onActivityResult()", que ha de poder diferenciarlas de algún modo. En nuestro caso como sólo le mostramos al usuario la información recuperada del contacto seleccionado devolvemos de la actividad *VerContacto* a la actividad *Inicio* un booleano que nos indica que todo ha ido correctamente.

9.6.14.2 Mostrar la información del contacto

Al lanzar el *Intent* explicado anteriormente, la clase *Inicio* deja de ser la actividad en curso para entrar en un estado onPause()primero y onStop() después, ya que no es

visible para el usuario. La clase *VerContacto*, que también extiende a Activity, toma toda la pantalla ofreciendo al usuario la información recuperada del contacto seleccionado.

Esta clase debe extraer los datos asociados al *Intent* con el que ha sido lanzada, capturar cuál de los contactos ha sido pulsado y mostrar por pantalla al usuario la información recuperada de la base de datos de *AgendaPersonal*.

En primer lugar, se muestra el método onCreate() de *VerContacto*, que extraerá los datos asociados al *Intent* y buscará en la bases de datos SQLite el contacto seleccionado. El Código 59 enseña el proceso.

```
public class VerContacto extends Activity {
  /** Called when the activity is first created. */
  private SQLiteDatabase baseDatos;
  private static final String TAG = "bdagenda";
  private static final String nombreBD = "agenda";
  Cursor c;
  private static String nombreCont = new String();
  private static String telefonoCont = new String();
  private static String correoCont = new String();
  private Button botonCerrar;
  TextView textNombre, textTelefono, textCorreo;

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
      setContentView(R.layout.vercontacto);
      Bundle bundle = this.getIntent().getExtras();
      //Recogemos el nombre seleccionado por el usuario que hemos pasado
a traves del Bundle, y lo almacenamos en una variable.
      String nombreRec = bundle.getString("NOMBRESEL");
      //Abrimos la base de datos.
      abrirBasedatos();
      //Consulta: contacto del teléfono seleccionado por el usuario con
su respectivo nombre, número de teléfono y dirección de correo
electrónico.
      c = baseDatos.rawQuery("SELECT nombre,telefono,correo FROM
contacto WHERE nombre LIKE '"+ nombreRec +"' ", null);
      //Almacenamos el resultado obtenido en la consulta en nuestras
variables.
      //Nos aseguramos de que existe al menos un registro.
      if (c.moveToFirst()) {
      //Recorremos el cursor hasta que no haya más registros.
        do {
            nombreCont = c.getString(0);
            telefonoCont = c.getString(1);
            correoCont = c.getString(2);
            } while(c.moveToNext());
      }
      //Cerramos la base de datos.
      cerrarBasedatos();
      //Si en el teléfono del contacto no encontramos ningún valor le
ponemos el valor por defecto.
      if (telefonoCont.equals("")){
        telefonoCont = "El contacto no dispone de telefono introducido";
      }
```

```
      //Si en la dirección de correo del contacto no encontramos ningún
valor le ponemos el valor por defecto.
      if ((correoCont.equals(""))||(correoCont.equals("Email"))){
        correoCont = "El contacto no dispone de dirección de correo
electronica introducida";
      }
      textNombre   = (TextView) findViewById(R.id.textView3);
      textNombre.setText(nombreCont); //Añadimos el nombre del contacto
recuperado de nuestra base de datos al TextView nombre.
      textTelefono = (TextView) findViewById(R.id.textView5);
      textTelefono.setText(telefonoCont); //Añadimos el número del
teléfono del contacto recuperado de nuestra base de datos al TextView
teléfono.
      textCorreo = (TextView)findViewById(R.id.textView7);
      textCorreo.setText(correoCont); //Añadimos la dirección de correo
electrónico del contacto recuperada de nuestra base de datos al TextView
correo.
      botonCerrar = (Button) findViewById(R.id.btCancelar);
      //Configuramos la acción de el botón "volver".
      //Finalizamos la subactividad VerContacto devolviendo "RESULT_OK"
a la actividad llamante Inicio.
      botonCerrar.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
          Intent intent = new Intent(VerContacto.this, Inicio.class);
          setResult(RESULT_OK, intent);
          finish();
        }
      });
    }
  //Procedimiento para abrir la base de datos.
  //Si no existe se creará, también se creará la tabla contacto.
  //Si da error al abrir o crear la base de datos montramos un mensaje
de error.
  private void abrirBasedatos() {
    try {
      baseDatos = openOrCreateDatabase(nombreBD, MODE_WORLD_WRITEABLE,
null);
    }
    catch (Exception e) {
      Log.i(TAG, "Error al abrir o crear la base de datos" + e);
    }
  }
  //Procedimiento para cerrar la base de datos
  private void cerrarBasedatos(){
    baseDatos.close();
  }
}
```

Código 59. Clase VerContacto.

Lo primero de todo es mencionar que, tal y como muestra el código anterior, la clase *VerContacto* extiende directamente a la clase *Activity*.

La clase *Bundle* forma parte del paquete “android.os”, un pequeño pero relevante paquete que ofrece varios servicios asociados al sistema operativo, paso de mensajes o comunicaciones entre procesos. Esta clase representa una colección de valores y suele

utilizarse para recuperar los datos asociados a un *Intent*, tal y como se hace en la clase *VerContacto* de nuestra aplicación.

Mediante el método "getIntent()" de la clase Activity se obtiene el objeto *Intent* con el que fue lanzada dicha actividad (si es el caso). A su vez, este objeto *Intent* ofrece un método llamado "getExtras()" que devuelve el conjunto total de valores asociados a él. A través del objeto *Bundle*, que contendrá todos los valores del *Intent*, para recoger el valor del contacto seleccionado, se recupera individualmente accediendo a el a través de su etiqueta identificativa (en nuestro caso). Al mismo tiempo que se recupera, se almacena en una variable de tipo *string* llamada "nombreRec". En esta variable es donde estará guardado finalmente el nombre del contacto cuya información relevante se va a mostrar al usuario.

A continuación mediante la interfaz que hemos construido en el archivo "vercontacto.xml" (mostrado en el Código 60 a continuación) y el código anterior que introduce la información del contacto seleccionado recuperada de la base de datos de la aplicación en dicha interfaz.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
    <TextView android:layout_height="wrap_content"
      android:id="@+id/textView1"
      android:layout_width="fill_parent"
      android:gravity="center"
      android:text="@string/datoscontacto"
      android:background="#0000aa"
      android:layout_weight="1"
      android:textStyle="bold"/>
    <TextView android:layout_height="wrap_content"
      android:id="@+id/textView2"
      android:layout_width="fill_parent"
      android:layout_weight="1"
      android:text="@string/nombrecontacto"
      android:textStyle="bold"/>
    <TextView android:layout_height="wrap_content"
      android:id="@+id/textView3"
      android:layout_width="fill_parent"
      android:layout_weight="1"
      android:text="Nombre"
      android:textStyle="italic|bold"
      android:textColor="#0000aa"/>
    <TextView android:layout_height="wrap_content"
      android:id="@+id/textView4"
      android:layout_width="fill_parent"
      android:layout_weight="1"
      android:text="@string/telefonocontacto"
      android:textStyle="bold"/>
    <TextView android:layout_height="wrap_content"
      android:id="@+id/textView5"
      android:layout_width="fill_parent"
      android:layout_weight="1"
```

```
      android:text="Telefono"
      android:textStyle="italic|bold"
      android:textColor="#0000aa"/>
    <TextView android:layout_height="wrap_content"
      android:id="@+id/textView6"
      android:layout_width="fill_parent"
      android:layout_weight="1"
      android:text="@string/correocontacto"
      android:textStyle="bold"/>
    <TextView android:layout_height="wrap_content"
      android:id="@+id/textView7"
      android:layout_width="fill_parent"
      android:layout_weight="1"
      android:text="Correo Electronico"
      android:textStyle="italic|bold"
      android:textColor="#0000aa"/>
    <Button android:id="@+id/btCancelar"
      android:layout_width="200px"
        android:gravity="center_vertical|center_horizontal"
        android:layout_height="wrap_content"
        android:text="Volver"
        android:layout_marginTop="20px"
        android:layout_gravity="center_vertical|center_horizontal"/>
</LinearLayout>
```

Código 60. vercontacto.xml.

La interfaz en cuestión que mostramos al usuario es la de la Figura 20 que mostramos a continuación.

Figura 20. Interfaz de información del contacto seleccionado.

Ya se ha construido y mostrado la información del contacto al usuario, pero ahora es necesario saber qué hacer cuando el usuario seleccione el botón "Volver" que podemos observar en la figura anterior.

El método "setOnClickListener()" de la clase *Button* proporciona o asocia un *Listener* al botón mencionado anteriormente y mediante el método "onClick()" configuramos las acciones a realizar al presionar el usuario dicho botón, que son, creamos un *Intent* para pasar el control de la aplicación a la clase *Inicio*, mediante el método "setResult(*RESULT_OK*, intent)" devolvemos el booleano que esperaba la clase *Inicio* indicando que todo se ha desarrollado correctamente, además de dar por terminada con el método finish() la presente actividad *VerContacto*.

9.6.15 Añadir contacto

Es importante para la experiencia del usuario con la aplicación que este pueda interactuar con la base de datos de *AgendaPersonal* y como funcionalidad de la misma se puedan agregar contactos nuevos a la base de datos.

Para ello hemos desarrollado la clase *Anadir* que hereda o extiende de la clase *Activity* la cual será lanzada por medio de un *Intent* desde la clase *Inicio* de nuestra aplicación a través del menú de aplicación ya mostrado en puntos anteriores de este mismo capítulo.

Mediante el Código 61 que mostramos a continuación de la clase *Inicio*, lanzamos la Actividad *Anadir* y finalizamos la actividad concurrente.

```
//Método para configurar la acción a desarrollar por cada elemento del
menú.
@Override
public boolean onMenuItemSelected(int featureId, MenuItem item) {
  super.onMenuItemSelected(featureId, item);
    switch(item.getItemId()) {
      case R.id.anadir_contacto:
        //Si el usuario selecciona añadir contacto lanzamos la actividad
Anadir y finalizamos la actividad Inicio.
        Intent intent = new Intent(Inicio.this, Anadir.class);
        startActivity(intent);
        finish();
        break;
```

Código 61. Llamada a la clase Anadir.

Como podemos observar en el código anterior, creamos un *Intent* indicando que desde la clase *Inicio* vamos a lanzar la clase *Anadir*, acto seguido iniciamos la actividad mediante dicho *Intent* y por último finalizamos la actividad actual, en este caso la actividad *Inicio*.

Acto seguido se cargará la interfaz diseñada para la adición de contactos de la aplicación con la que puede interactuar el usuario.

Figura 21. Interfaz de adición de contactos.

Para la creación de dicha interfaz hemos usado el archivo “anadir.xml” que mostraremos a continuación.

```
<?xml version="1.0" encoding="utf-8"?>
    <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
        <TextView android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Añadir Contacto"
            android:id="@+id/textView1"
            android:gravity="center"
            android:textColor="#ffffff"/>
        <EditText android:text="Nombre"
            android:id="@+id/txtNombre"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
        <EditText android:text="Telefono"
            android:id="@+id/txtTelefono"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
        <EditText android:text="Email"
            android:id="@+id/txtEmail"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
        <Button android:id="@+id/btGuardar"
            android:layout_width="200px"
            android:gravity="center_vertical|center_horizontal"
            android:layout_height="wrap_content"
            android:text="Guardar"
            android:layout_marginTop="20px"
```

```
            android:layout_gravity="center_vertical|center_horizontal"/>
        <Button android:id="@+id/btCancelar"
            android:layout_width="200px"
            android:gravity="center_vertical|center_horizontal"
            android:layout_height="wrap_content"
            android:text="Cancelar"
            android:layout_marginTop="20px"
            android:layout_gravity="center_vertical|center_horizontal"/>
    </LinearLayout>
```

Código 62. anadir.xml.

La interfaz mostrada anteriormente consta de tres elementos *EditText* en los que el usuario introduce la información relevante del nuevo contacto, y dos botones, uno para guardar la información introducida y otro para cancelar la acción en caso de que fuese necesario, para desarrollar la funcionalidad de la aplicación anterioremente descrita, nos apoyamos en la clase *Anadir* que mostraremos a continuación.

```
package PFC.Agenda;

import java.util.StringTokenizer;
import android.app.Activity;
import android.content.ContentValues;
import android.content.Intent;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.text.InputType;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class Anadir extends Activity {
  /** Called when the activity is first created. */
  EditText editNombre,editTelefono,editCorreo;
  private Button botonGuardar;
  private Button botonCerrar;
  private SQLiteDatabase baseDatos;
  private static final String TAG = "bdagenda";
  private static final String nombreBD = "agenda";
  private static final String tablaContacto = "contacto";

  //guardamos en un String toda la creación de la tabla
  private static final String crearTablaContacto = "create table if not
exists " + " contacto (codigo integer primary key autoincrement, " + "
nombre text not null unique, telefono text not null unique, correo
text);";

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.anadir);
    //Añadimos un EditText para el campo nombre de nuestro contacto.
    editNombre   = (EditText) findViewById(R.id.txtNombre);
    //Añadimos el tipo de teclado texto para el EditText nombre.
```

```
    editNombre.setInputType(InputType.TYPE_CLASS_TEXT);
    //Seleccionamos el valor del campo nombre.
    editNombre.selectAll();
    //Añadimos un EditText para el campo teléfono de nuestro contacto.
    editTelefono = (EditText) findViewById(R.id.txtTelefono);
    //Añadimos el tipo de teclado numérico para el EditText teléfono.
    editTelefono.setInputType(InputType.TYPE_CLASS_PHONE);
    //Seleccionamos el valor del campo teléfono.
    editTelefono.selectAll();
    //Añadimos un EditText para el campo dirección de correo electrónico
de nuestro contacto.
    editCorreo = (EditText)findViewById(R.id.txtEmail);
    //Añadimos el tipo de teclado emailAddress para el EditText
dirección de correo electrónico.
   editCorreo.setInputType(InputType.TYPE_TEXT_VARIATION_EMAIL_ADDRESS);
    //Seleccionamos el valor del campo dirección de correo electrónico.
    editCorreo.selectAll();

    //Añadimos el botón de Guardar.
    botonGuardar = (Button) findViewById(R.id.btGuardar);
    //Añadimos el botón de Cancelar.
    botonCerrar = (Button) findViewById(R.id.btCancelar);

    //Guardar el contacto actual en la agenda configurando el Listener
del botón Guardar.
    botonGuardar.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        (...)
        String nomb = editNombre.getText().toString();
        (...)
        String tlf = editTelefono.getText().toString();
        (...)
        String eml = editCorreo.getText().toString();
        (...)
        //Si el campo número de teléfono y el campo dirección de correo
electrónico es correcto, procedemos a la inserción en nuestra base de
datos.
        if ((tlfcorrecto)&&(emlcorrecto)&&(nombcorrecto)){
          //Abrir la base de datos, se creará si no existe.
          abrirBasedatos();
          (...)
          //Insertar la fila o registro en la tabla "contacto".
          //Si la inserción es correcta devolverá true.
boolean resultado = insertarFila(editNombre.getText().toString(),
editTelefono.getText().toString(), email);

          if(resultado)
            //Mostramos un mensaje de confirmación.
            Toast.makeText(getApplicationContext(), "Contacto añadido
correctamente", Toast.LENGTH_LONG).show();
          else
            //Mostramos un mensaje de error en la inserción, el contacto
ya existía.
            Toast.makeText(getApplicationContext(), "No se ha podido
guardar el contacto, ya existe" , Toast.LENGTH_LONG).show();
         //Cerramos la base de datos.
        cerrarBasedatos();
        }
        else{
          if (!nombcorrecto)
```

```
            //Mostramos un mensaje de error al introducir el nombre del
contacto.
            Toast.makeText(getApplicationContext(), "No se ha podido
guardar el contacto, nombre del contacto mal introducido" ,
Toast.LENGTH_LONG).show();
          else{
            if (!tlfcorrecto)
              //Mostramos un mensaje de error al introducir el número de
teléfono.
              Toast.makeText(getApplicationContext(), "No se ha podido
guardar el contacto, numero de telefono mal introducido" ,
Toast.LENGTH_LONG).show();
            else
              //Mostramos un mensaje de error al introducir la dirección
de correo electrónico.
              Toast.makeText(getApplicationContext(), "No se ha podido
guardar el contacto, direccion de correo mal introducida" ,
Toast.LENGTH_LONG).show();
          }
        }
        //Creamos el Intent para cargar la actividad Inicio.
        Intent intent = new Intent(Anadir.this, Inicio.class);
        startActivity(intent); //Iniciamos la actividad Inicio.
        finish(); //Finalizamos la actividad Anadir.
      }
    });

    //Configuramos la acción de el botón "volver".
    //Finalizamos la actividad Anadir e iniciamos la actividad Inicio.
    botonCerrar.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        //Creamos el Intent para cargar la actividad Inicio.
        Intent intent = new Intent(Anadir.this, Inicio.class);
        startActivity(intent); //Iniciamos la actividad Inicio.
        finish(); //Finalizamos la actividad Anadir.
      }
    });
  }

  //Procedimiento para abrir la base de datos.
  //Si no existe se creará, también se creará la tabla contacto.
  //Si da error al abrir o crear la base de datos montramos un mensaje
de error.
  private void abrirBasedatos() {
    try {
      baseDatos = openOrCreateDatabase(nombreBD, MODE_WORLD_WRITEABLE,
null);
      baseDatos.execSQL(crearTablaContacto);
    }
    catch (Exception e) {
      Log.i(TAG, "Error al abrir o crear la base de datos" + e);
    }
  }

  //Procedimiento para cerrar la base de datos.
  private void cerrarBasedatos(){
    baseDatos.close();
  }
```

```
  //Método que realiza la inserción de los datos en nuestra tabla
contacto.
  private boolean insertarFila(String nombre, String telefono, String
correo) {
    ContentValues values = new ContentValues();
    values.put("nombre",nombre );
    values.put("telefono", telefono);
    values.put("correo", correo);
    Toast.makeText(getApplicationContext(), "Nombre: " + nombre + ", " +
"teléfono: " + telefono, Toast.LENGTH_LONG).show(); //Mostramos un
mensaje con los datos introducidos en la base de datos.
    return (baseDatos.insert(tablaContacto, null, values) > 0);
  }
}
```

Código 63. Clase Anadir.

A continuación vamos a explicar los pasos que seguimos en la clase *Anadir* para la adición de un contacto nuevo, primeramente proporcionamos al usuario tres marcos “EditText” para la introducción por parte de este último de los datos sensibles del contacto nuevo, como son el nombre, número de teléfono y dirección de correo electrónico, también proporcionamos dos botones en la interfaz de la clase *Button*, uno para guadar el contacto nuevo en la base de datos y otro para cancelar la actividad de adición de nuevo contacto y volver a la actividad *Inicio*.

Una vez introducidos los datos mencionados anteriormente el usuario deberá pulsar el botón de guardado, en tal caso, usando en método “setOnClickListener()” de la clase *Button* almacenaremos en la base de datos SQLite de nuestra aplicación la información del nuevo contacto, para ellos comprobaremos los datos introducidos mediante los controles descritos en anteriores apartados de este mismo capítulo, una vez superados dichos controles y mediante los métodos “abrirBasedatos()”, “insertarFila()” y “cerrarBasedatos()” insertaremos la información en la base de datos de *AgendaPersonal*, para a continuación mostrar al usuario un mensaje por pantalla indicando si la inserción ha sido correcta o ha habido algún problema con los datos introducidos o el contacto ya existía en la base de datos.

Acto seguido, tanto si la inserción ha sido correcta como si el usuario ha pulsado el botón de cancelación de acción, cargamos un *Intent* en el que indicamos que desde la actividad actual *Anadir*, cargaríamos la actividad *Inicio*, lanzamos dicho *Intent* y a continuación finalizaríamos la actividad actual *Anadir*, gracias a esto los cambios en la lista de contactos de la actividad *Inicio* quedarían reflejados.

Para abrir la base de datos de *AgendaPersonal* usamos el método “openOrCreateDatabase()” que nos proporciona la clase *android.database.sqlite.SQLiteDatabase*.

Para la inserción de la tupla con la información introducida del contacto por parte del usuario, usamos la clase *ContentValues* y el método "put()" de la misma, así cargamos los datos anteriores para a continuación realizar la inserción de estos valores en nuestra base de datos gracias al método "insert()" de la clase *SQLiteDatabase* que es la encargada de gestionar dicha base de datos SQLite de nuestra aplicación, a dicho método le proporcionamos la tabla de la base de datos dónde realizaremos la inserción, y los valores a introducir en la misma contenidos en el objeto de la clase *ContentValues* mencionado anteriormente.

Por último para el cierre de la base de datos anteriormente comentada usamos en método "close()" de la clase *SQLiteDatabase* anteriormente mencionada.

Cabe destacar que si en todo este proceso hubiera o hubiese algún tipo de error el usuario estaría informado en todo momento mediante el sistema de mensajes por pantalla descrito en el código anterior y detallado en puntos anteriores de este mismo capítulo.

9.6.16 Editar contacto

Al igual que en el punto anterior referente a la adición de contactos, es importante para la experiencia del usuario con la aplicación que este pueda interactuar con la base de datos de *AgendaPersonal* y como funcionalidad de la misma se puedan editar contactos ya introducidos en la base de datos de la aplicación, ya sea por diverso motivos, como bien puede ser el cambio de número telefónico o de correo electrónico.

Para ello hemos desarrollado la clase *Editar* que hereda o extiende de la clase *Activity* la cual será lanzada por medio de un *Intent* desde la clase *Inicio* de nuestra aplicación a través del menú contextual de la aplicación ya mostrado en puntos anteriores de este mismo capítulo y que se muestra al usuario a través de la pulsación permanente de un contacto de la lista de contactos.

Mediante el Código 64 que mostramos a continuación de la clase *Inicio*, lanzamos la Actividad *Editar* y finalizamos la actividad concurrente anteriormente mencionada, cabe destacar que al igual que ocurría en la llamada a la actividad *VerContacto* de nuestra aplicación, para pasarle a la actividad *Editar* el contacto seleccionado por el usuario usamos las clases *Intent*, *Bundle* y sus respectivos métodos como son "startActivity()", "putString()" y "putExtras()" como mostramos a continuación.

```
//Método para configurar la acción a realizar cuándo un elemento del
menú es seleccionado.
@Override
public boolean onContextItemSelected(MenuItem item) {
  switch (item.getItemId()) {

  (...)

  case R.id.group_elemento4:
    //Si el usuario elige la opción de editar contacto.
```

```
    //Creamos el Intent con el contacto seleccionado para la actividad
Editar.
    Intent intent = new Intent(Inicio.this, Editar.class);
    Bundle bundle = new Bundle();
    bundle.putString("NOMBRESEL", _itemSeleccionado);
    intent.putExtras(bundle);
    startActivity(intent); //Iniciamos la actividad Editar.
    finish(); //Finalizamos la actividad Inicio.
    return true;

  (...)

}
```

Código 64. Llamada a clase Editar.

Como podemos observar en el código anterior, creamos un *Intent* indicando que desde la clase *Inicio* vamos a lanzar la clase *Editar*, acto seguido iniciamos la actividad mediante dicho *Intent*, el cuál lleva asociado el nombre del contacto que vamos a editar mediante el método “putExtras()” y el objeto de la clase *Bundle* que le hemos asociado,y por último finalizamos la actividad actual, en este caso la actividad *Inicio*.

Acto seguido se cargará la interfaz diseñada para la edición de contactos de la aplicación con la que puede interactuar el usuario.

Figura 22. Interfaz de edición de contactos.

Para la creación de dicha interfaz hemos usado el archivo “editar.xml” que mostraremos a contuniación.

```
<?xml version="1.0" encoding="utf-8"?>
```

```
    <LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:orientation="vertical">
        <TextView android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="Editar Contacto"
            android:id="@+id/textView1"
            android:gravity="center"
            android:textColor="#ffffff"/>
        <EditText android:text="Nombre"
            android:id="@+id/txtNombre"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
        <EditText android:text="Telefono"
            android:id="@+id/txtTelefono"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
        <EditText android:text="Email"
            android:id="@+id/txtEmail"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
        <Button android:id="@+id/btGuardar"
            android:layout_width="200px"
            android:gravity="center_vertical|center_horizontal"
            android:layout_height="wrap_content"
            android:text="Guardar"
            android:layout_marginTop="20px"
            android:layout_gravity="center_vertical|center_horizontal"/>
        <Button android:id="@+id/btCancelar"
            android:layout_width="200px"
            android:gravity="center_vertical|center_horizontal"
            android:layout_height="wrap_content"
            android:text="Cancelar"
            android:layout_marginTop="20px"
            android:layout_gravity="center_vertical|center_horizontal"/>
    </LinearLayout>
```

Código 65. editar.xml.

La interfaz mostrada anteriormente consta de tres elementos *EditText* en los que el usuario puede cambiar la información relevante del contacto seleccionado, y dos botones, uno para guardar la información nueva introducida y otro para cancelar la acción en caso de que fuese necesario, para desarrollar la funcionalidad de la aplicación anterioremente descrita, nos apoyamos en la clase *Editar* que mostraremos a continuación.

```
package PFC.Agenda;

import java.util.StringTokenizer;

import android.app.Activity;
import android.content.ContentValues;
import android.content.Intent;
import android.database.Cursor;
import android.database.sqlite.SQLiteDatabase;
```

```
import android.os.Bundle;
import android.text.InputType;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class Editar extends Activity {
  /** Called when the activity is first created. */
  EditText editNombre,editTelefono,editCorreo;
  private Button botonGuardar;
  private Button botonCerrar;
  private SQLiteDatabase baseDatos;
  private static final String TAG = "bdagenda";
  private static final String nombreBD = "agenda";
  private static final String tablaContacto = "contacto";
  Cursor c;
  private static String nombreCont = new String();
  private static String telefonoCont = new String();
  private static String correoCont = new String();

  @Override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.editar);
    Bundle bundle = this.getIntent().getExtras();
    //Recogemos el nombre seleccionado por el usuario que hemos pasado a
traves del Bundle, y lo almacenamos en una variable.
    String nombreRec = bundle.getString("NOMBRESEL");
    //Abrimos la base de datos.
    abrirBasedatos();
    //Consulta: contacto del teléfono seleccionado por el usurio con su
respectivo nombre, número de teléfono y dirección de correo electrónico.
    c = baseDatos.rawQuery("SELECT nombre,telefono,correo FROM contacto
WHERE nombre LIKE '"+ nombreRec +"' ", null);
    //Almacenamos el resultado obtenido en la consulta en nuestras
variables.
    //Nos aseguramos de que existe al menos un registro.
    if (c.moveToFirst()) {
      //Recorremos el cursor hasta que no haya más registros.
      do {
          nombreCont = c.getString(0);
          telefonoCont = c.getString(1);
          correoCont = c.getString(2);
      } while(c.moveToNext());
    }
    //Cerramos la base de datos.
    cerrarBasedatos();
    //Añadimos un EditText para el campo nombre de nuestro contacto.
    editNombre   = (EditText) findViewById(R.id.txtNombre);
    //Añadimos el nombre del contacto recuperado de nuestra base de
datos al EditText nombre.
    editNombre.setText(nombreCont);
    //Añadimos el tipo de teclado texto para el EditText nombre.
    editNombre.setInputType(InputType.TYPE_CLASS_TEXT);
    //Añadimos un EditText para el campo teléfono de nuestro contacto.
    editTelefono = (EditText) findViewById(R.id.txtTelefono);
    //Añadimos el número de teléfono del contacto recuperado de nuestra
base de datos al EditText teléfono.
```

```
    editTelefono.setText(telefonoCont);
    //Añadimos el tipo de teclado numérico para el EditText teléfono.
    editTelefono.setInputType(InputType.TYPE_CLASS_PHONE);
    //Añadimos un EditText para el campo dirección de correo electrónico
de nuestro contacto.
    editCorreo = (EditText)findViewById(R.id.txtEmail);
    //Añadimos la dirección de correo electrónico del contacto
recuperado de nuestra base de datos al EditText correo.
    editCorreo.setText(correoCont);
    //Añadimos el tipo de teclado emailAddress para el EditText
dirección de correo electrónico.
   editCorreo.setInputType(InputType.TYPE_TEXT_VARIATION_EMAIL_ADDRESS);
    //Añadimos el botón de Guardar.
    botonGuardar = (Button) findViewById(R.id.btGuardar);
    //Añadimos el botón de Cancelar.
    botonCerrar = (Button) findViewById(R.id.btCancelar);

    //Configuramos la acción a realizar cuando el usuario pulse el botón
guardar.
    //Guardar el contacto actual en la agenda
    botonGuardar.setOnClickListener(new View.OnClickListener() {
      public void onClick(View v) {
        (...)
        String nomb = editNombre.getText().toString();
        (...)
        String tlf = editTelefono.getText().toString();
        (...)
        String eml = editCorreo.getText().toString();
        (...)
        //Si el campo número de teléfono y el campo dirección de correo
electrónico es correcto, procedemos a la inserción en nuestra base de
datos.
        if ((tlfcorrecto)&&(emlcorrecto)&&(nombcorrecto)){
          //Abrir la base de datos, se creará si no existe.
          abrirBasedatos();
          (...)
          //Actualizamos la fila o registro en la tabla "contacto".
          //si la actualización es correcta devolverá true.
 boolean resultado = actualizarFila(editNombre.getText().toString(),
editTelefono.getText().toString().trim(), email);

          if(resultado)
            //Si la actualización es correcta mostramos el mensaje de
contacto actualizado.
            Toast.makeText(getApplicationContext(), "Contacto
actualizado correctamente", Toast.LENGTH_LONG).show();
          else
            //Si la actualización falla mostramos un mensaje de error.
            Toast.makeText(getApplicationContext(), "No se ha podido
actualizar el contacto, el nombre o el telefono proporcionados ya
existen en la agenda" , Toast.LENGTH_LONG).show();
            //Cerramos la bases de datos.
            cerrarBasedatos();
          }
          else{
            if (!nombcorrecto)
            //Mostramos un mensaje de error al introducir el nombre del
contacto.
```

```
            Toast.makeText(getApplicationContext(), "No se ha podido
guardar el contacto, nombre del contacto mal introducido" ,
Toast.LENGTH_LONG).show();
            else{
              if (!tlfcorrecto)
                //Mostramos un mensaje de error al introducir el número
de teléfono.
                Toast.makeText(getApplicationContext(), "No se ha podido
guardar el contacto, numero de telefono mal introducido" ,
Toast.LENGTH_LONG).show();
              else
                //Mostramos un mensaje de error al introducir la
dirección de correo electrónico.
                Toast.makeText(getApplicationContext(), "No se ha podido
guardar el contacto, direccion de correo mal introducida" ,
Toast.LENGTH_LONG).show();
            }

          }
            //Creamos el Intent para cargar la actividad Inicio.
            Intent intent = new Intent(Editar.this, Inicio.class);
            //Iniciamos la actividad Inicio.
            startActivity(intent);
            //Finalizamos la actividad Editar.
            finish();
          }
        });
        //Configuramos la acción de el botón "volver".
        //Finalizamos la actividad Editar e iniciamos la actividad
Inicio.
        botonCerrar.setOnClickListener(new View.OnClickListener() {
          public void onClick(View v) {
              Intent intent = new Intent(Editar.this, Inicio.class);
              startActivity(intent);
              finish();
          }
        });
    }
  //Procedimiento para abrir la base de datos.
  //Si no existe se creará, también se creará la tabla contacto.
  //Si da error al abrir o crear la base de datos montramos un mensaje
de error.
  private void abrirBasedatos() {
    try {
      baseDatos = openOrCreateDatabase(nombreBD, MODE_WORLD_WRITEABLE,
null);
    }
    catch (Exception e) {
      Log.i(TAG, "Error al abrir o crear la base de datos" + e);
    }
  }

  //Procedimiento para cerrar la base de datos.
  private void cerrarBasedatos(){
    baseDatos.close();
  }

  //Método que realiza la actualización de los datos en nuestra tabla
contacto.
```

```
  private boolean actualizarFila(String nombre, String telefono, String
correo) {
  ContentValues values = new ContentValues();
  values.put("nombre",nombre );
  values.put("telefono", telefono);
  values.put("correo", correo);
  //Actualizamos el registro en la base de datos.
  try{
    baseDatos.update(tablaContacto, values, "nombre='"+nombreCont+"'",
null);
  }
  catch (Exception e) {
    return false;
  }
  //Mostramos un mensaje con los datos actualizados en la base de datos.
  Toast.makeText(getApplicationContext(), "Nombre: " + nombre + ", " +
"teléfono: " + telefono, Toast.LENGTH_LONG).show();
  return true;
  }
}
```

Código 66. Clase Editar.

A continuación vamos a explicar los pasos que seguimos en la clase *Editar* para la edición de un contacto seleccionado, primeramente proporcionamos al usuario tres marcos “EditText” para la modificación por parte de este de los datos sensibles del contacto seleccionado, como son el nombre, número de teléfono y/o dirección de correo electrónico, también proporcionamos dos botones en la interfaz de la clase *Button*, uno para guadar los nuevos datos proporcionados en la base de datos y otro para cancelar la actividad de edición del contacto y volver a la actividad *Inicio*, en este último caso el contacto mantendría los datos antiguos.

Una vez introducidos los datos mencionados anteriormente el usuario deberá pulsar el botón de guardado, en tal caso, usando en método “setOnClickListener()” de la clase *Button* almacenaremos en la base de datos SQLite de nuestra aplicación la información nueva del contacto, para ellos comprobaremos los datos introducidos mediante los controles descritos en anteriores apartados de este mismo capítulo, una vez superados dichos controles y mediante los métodos “abrirBasedatos()”, “actualizarFila()” y “cerrarBasedatos()” insertaremos la información en la base de datos de *AgendaPersonal*, para a continuación mostrar al usuario un mensaje por pantalla indicando si la actualización ha sido correcta, o ha habido algún problema con los datos introducidos, o el nombre o número de teléfono del contacto a modificar no existían en la base de datos previamente.

Acto seguido, tanto si la actualización ha sido correcta como si el usuario ha pulsado el botón de cancelación de acción, cargamos un *Intent* en el que indicamos que desde la actividad actual *Editar*, cargaríamos la actividad *Inicio*, lanzamos dicho *Intent* y a continuación finalizaríamos la actividad actual *Editar*, gracias a esto los cambios en la lista de contactos de la actividad *Inicio* quedarían reflejados.

Para abrir la base de datos de *AgendaPersonal* usamos el método “openOrCreateDatabase()” que nos proporciona la clase *android.database.sqlite.SQLiteDatabase*.

Para la actualización de la tupla con la información introducida del contacto por parte del usuario, usamos la clase *ContentValues* y el método “put()” de la misma, así cargamos los datos anteriores para a continuación realizar la modificación de estos valores en nuestra base de datos gracias al método “update()” de la clase *SQLiteDatabase* que es la encargada de gestionar dicha base de datos SQLite de nuestra aplicación, a dicho método le porporcionamos la tabla donde se encuentra la tupla a modificar, los valores nuevos introducidos por el usuario en el objeto *ContentValues* mencionado anteriormente, y el nombre del contacto antiguo sobre el que se realizará la modificación.

Por último para el cierre de la base de datos anteriormente comentada usamos en método “close()” de la clase *SQLiteDatabase* anteriormente mencionada.

Cabe destacar que si en todo este proceso hubiera o hubiese algún tipo de error el usuario estaría informado en todo momento mediante el sistema de mensajes por pantalla descrito en el código anterior y detallado en puntos anteriores de este mismo capítulo.

9.6.17 Borrar contacto

Como ya hemos mencionado, es importante para la experiencia del usuario con la aplicación que este pueda interactuar con la base de datos de *AgendaPersonal* y como funcionalidad de la misma se puedan borrar o eliminar contactos ya existentes en la base de datos de nuestra aplicación.

Para ello hemos implementado en la clase *Inicio* de nuestra aplicación, que hereda o extiende de la clase *Activity*, una funcionalidad para borrar el contacto seleccionado por el usuario, esta opción se la mostramos a este último mediante el menú contextual de nuestra aplicación, ya detallado en puntos anteriores de este mismo capítulo.

Figura 23. Interfaz de eliminación de contactos.

Mediante el Código 67 que mostramos a continuación de la clase *Inicio*, realizamos la eliminación del contacto seleccionado de la base de datos de nuestra aplicación.

```
//Método para configurar la acción a realizar cuándo un elemento del
menú es seleccionado.
@Override
public boolean onContextItemSelected(MenuItem item) {
  switch (item.getItemId()) {

  (...)

  case R.id.group_elemento5:
  //Si el usuario selecciona que desea borrar el contacto.
  //Mostramos un mensaje de confirmación antes de borrar el contacto.
  AlertDialog.Builder alertDialog4 = new
AlertDialog.Builder(Inicio.this);
  alertDialog4.setMessage("¿Desea borrar el contacto?");
  alertDialog4.setTitle("Borrando contacto...");
  alertDialog4.setIcon(android.R.drawable.ic_dialog_alert);
  alertDialog4.setCancelable(false);
  alertDialog4.setPositiveButton("Sí", new
DialogInterface.OnClickListener() {//Si el usuario elige que sí.
  public void onClick(DialogInterface dialog, int which) {
    try {
      //Abrimos la base de datos.
      baseDatos = openOrCreateDatabase(nombreBD, MODE_WORLD_WRITEABLE,
null);
      //Borramos el contacto seleccionado.
      baseDatos.delete(tablaContacto, "nombre='"+_itemSeleccionado+"'",
null);
      //Cerramos la base de datos.
```

```
      baseDatos.close();
      //Creamos el Intent para actualizar la lista llamando a la misma
actividad Inicio.
      Intent intent = new Intent(Inicio.this, Inicio.class);
      //Iniciamos la actividad Inicio para cargar la lista actualizada.
      startActivity(intent);
      //Finalizamos la actividad.
      finish();
    }
    catch (Exception e) {
      //Mostramos el mensaje de error si no podemos borrar el contacto.
      Toast.makeText(getApplicationContext(), "No se ha podido borrar el
contacto", Toast.LENGTH_LONG).show();
    }
  }
});
  alertDialog4.setNegativeButton("No", new
DialogInterface.OnClickListener() { //Si el usuario elige la opción de
no borrar el contacto.
  public void onClick(DialogInterface dialog, int which) {
    //Mostramos el mensaje al usuario de que la operación ha sido
cancelada.
    Toast.makeText(getApplicationContext(), "Borrado cancelado",
Toast.LENGTH_LONG).show();
  }
});
alertDialog4.show();
return true;

(...)

  }

}
```

Código 67. Borrar contacto.

Cómo podemos observar para la eliminación de contactos de nuestra aplicación usamos elementos de la clase *AlertDialog* para mostrar al usuario un mensaje de confirmación para la acción de borrado del contacto, y sus métodos “Builder()”, “setMessage()”, “setTitle()”, “setIcon()”, “setCancelable()”, “setPositiveButton()”, “setNegativeButton()”, los cuales nos ayudan en la creación de la interfaz del mensaje de advertencia como mostramos a continuación.

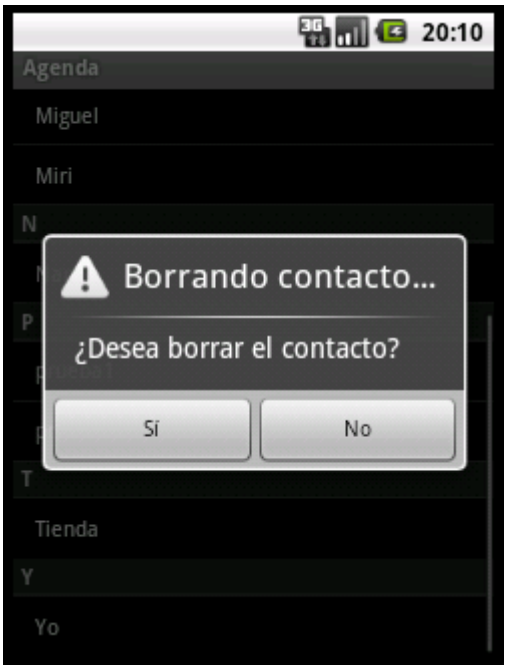


Figura 24. Mensaje de eliminación de contactos.

En caso de que la elección del usuario sea afirmativa, mediante los métodos de la clase *SQLiteDatabase* mostrados en el código anterior para abrir (openOrCreateDatabase()) y cerrar (close()) la base de datos, y el método “delete()” al cuál proporcionamos la tabla donde se encuentra la tupla y el nombre del contacto seleccionado para ser eliminado, es el método encargado de eliminar dicha tupla y por lo tanto dicho contacto.

A continuación para que la eliminación quede debidamente reflejada en la lista de contactos, creamos un *Intent* para cargar nuevamente la actividad *Inicio*, la lanzamos mediante el método “startActivity()” y finalizamos la actividad *Inicio* actual.

En caso de que la elección del usuario sea negativa se le mostrará un mensaje confirmando la cancelación de la acción, mediante la utilizando la clase *Toast* y sus métodos “makeText()” para cargar el texto a mostrar, y “show()” para mostrar finalmente el mensaje al usuario, como mostramos a continuación.

Figura 25. Mensaje de cancelación de borrado.

9.6.18 Llamar a un contacto

Entre las acciones que el usuario puede realizar sobre un contacto está la de realizar una llamada telefónica. Para tal fin se utilizará un *Intent* y se lanzará una nueva *Activity*, pero de una forma algo diferente a lo recientemente visto en apartados anteriores.

Como ya se ha explicado en anteriores ocasiones, mediante un *Intent* se delega una determinada acción en el sistema, de forma que este encuentra la aplicación más apropiada de entre todas las instaladas para llevarla a cabo. Un *Intent* consiste en la especificación de la acción a realizar, y los datos con los que debe realizarse.

Android cuenta por defecto con un gestor de llamadas telefónicas. Esto permite poder lanzar el *Intent* específico de una llamada sin tener que preocuparse de nada más, puesto que el sistema la derivará a dicho gestor de llamadas de Android.

El siguiente código, Código 68, expone los pasos a seguir para lanzar un *Intent* de llamada telefónica. La acción se lleva a cabo en el método “onContextItemSelected()” de la clase *Inicio*, una vez que el usuario ha seleccionado la opción correspondiente en el menú contextual de la aplicación, también encontramos esta funcionalidad en el menú de la aplicación de la clase *VerContacto* y en este caso se lleva a cabo en el método “onMenuItemSelected()”, a continuación el código del primer caso mencionado.

```
//Método para configurar la acción a realizar cuándo un elemento del
menú es seleccionado.
@Override
public boolean onContextItemSelected(MenuItem item) {
```

```
  switch (item.getItemId()) {
    case R.id.group_elemento1:
    //Mostrar un mensaje de confirmación antes de realizar la llamada.
    AlertDialog.Builder alertDialog = new
AlertDialog.Builder(Inicio.this);
    alertDialog.setMessage("¿Desea realizar la llamada al contacto?");
    alertDialog.setTitle("Llamar a contacto...");
    alertDialog.setIcon(android.R.drawable.ic_dialog_alert);
    alertDialog.setCancelable(false);
    alertDialog.setPositiveButton("Sí", new
DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int which) {
        String numtlf = new String();
        try {
          //Abrimos la base de datos.
          baseDatos = openOrCreateDatabase(nombreBD,
MODE_WORLD_WRITEABLE, null);
          //Consulta: número de teléfono del contacto seleccionado.
          c = baseDatos.rawQuery("SELECT telefono FROM contacto WHERE
nombre LIKE '"+ _itemSeleccionado +"' ", null);

          //Nos aseguramos de que existe al menos un registro
          if (c.moveToFirst()) {
            //Recorremos el cursor hasta que no haya más registros
            do {
              numtlf = c.getString(0);
            } while(c.moveToNext());
            String number = "tel:" + numtlf.trim();
            //Mostramos el mensaje del contacto al que llamamos.
            Toast.makeText(getApplicationContext(), "Llamando al " +
numtlf.trim(), Toast.LENGTH_LONG).show();

            //Llamamos a la actividad para llamar al contacto.
            Intent callIntent = new Intent(Intent.ACTION_CALL,
Uri.parse(number));
            startActivity(callIntent);
          }
          baseDatos.close(); //Cerramos la base de datos.
        }
        catch (Exception e) {
          //Mostramos el mensaje de fallo en la llamada.
          Toast.makeText(getApplicationContext(), "No se ha podido
realizar la llamada", Toast.LENGTH_LONG).show();
        }
      }
    });
    alertDialog.setNegativeButton("No", new
DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int which) {
        //Mostramos el mensaje de llamada cancelada.
        Toast.makeText(getApplicationContext(), "Llamada cancelada",
Toast.LENGTH_LONG).show();
      }
    });
    alertDialog.show();
    return true;
```

Código 68. Llamar contacto.

Al crear el objeto Intent se especifica el tipo de acción que se desea realizar. En el caso de la lista de contactos por ejemplo, se indicaba que se quería ejecutar una clase concreta que llevaba a cabo dicha acción (la clase *Inicio*). Aquí también se indica la acción a realizar, pero no diciendo directamente quién debe realizarla, sino diciendo qué se quiere realizar. El sistema ya se encargará de buscar la aplicación más indicada para ello.

Existen acciones frecuentes y definidas para indicar en un *Intent*. La correspondiente a la llamada telefónica es Intent.ACTION_CALL. Una vez creado el objeto Intent es necesario añadir como datos el número de teléfono al que se desea llamar; este debe ser adjuntado como un objeto *Uri*. Las *URI* relacionadas con recursos telefónicos han de tener el formato “tel:numero” para poder funcionar correctamente con la acción Intent.ACTION_CALL.

Tras la creación y configuración del objeto Intent ya se puede lanzar la actividad.Como en este caso no se espera que esta retorne ningún valor que deba ser capturado, la *Activity* se lanza sin más con el método “startActivity()”. En ese momento el sistema abrirá el gestor de llamadas y marcará el número indicado.

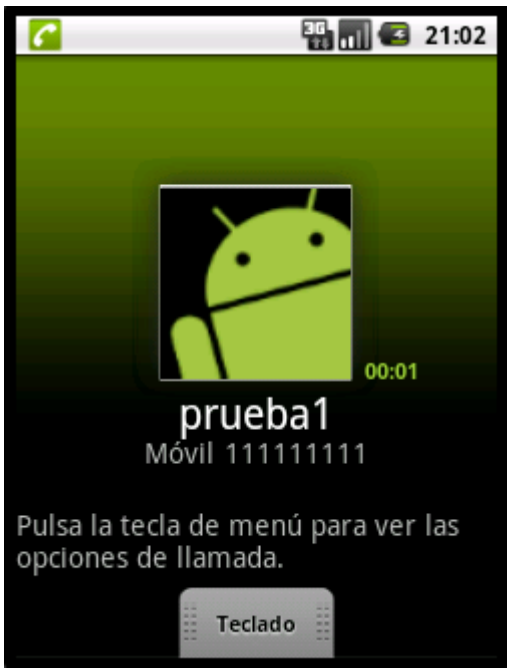


Figura 26. Llamada a un contacto.

El acceso a las funciones telefónicas del dispositivo móvil requiere de su correspondiente permiso en la declaración del manifiesto. El permiso es el siguiente:

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
```

Código 69. Declaración en el manifiesto del permiso de llamada telefónica.

Figura 27. Llamada desde Inicio.

Figura 28. Llamada desde VerContacto.

En las figuras de la parte superior podemos observar las formas que tiene el usuario para realizar la acción de llamar a un contacto existente en la base de datos de *AgendaPersonal*.

Por último, destacar que al igual que ocurría en el apartado anterior de este mismo capítulo, para realizar una llamada, mostraremos al usuario un mensaje de advertencia

para asegurarnos de que el usuario quiere realizar la acción de llamar al contacto, para ello usamos la clase *AlertDialog* y sus métodos “Builder()”, “setMessage()”, “setTitle()”, “setIcon()”, “setCancelable()”, “setPositiveButton()”, “setNegativeButton()”, los cuales nos ayudan en la creación de la interfaz del mensaje de advertencia, como ya mencionamos anteriormente y mostramos en la siguientes figuras.

Figura 29. Mensaje llamada Inicio.

Figura 30. Mensaje llamada VerContacto.

9.6.19 Enviar un SMS a un contacto

El proceso de envío de un SMS es bastante similar al visto en el en el apartado anterior para realizar una llamada teléfonica. Se utilizará un *Intent* para delegar en el sistema dicha operación.

Como ya se ha explicado en anteriores ocasiones, mediante un *Intent* se delega una determinada acción en el sistema, de forma que este encuentra la aplicación más apropiada de entre todas las instaladas para llevarla a cabo. Un *Intent* consiste en la especificación de la acción a realizar, y los datos con los que debe realizarse.

Android cuenta por defecto con un gestor de envío de mensajes. Esto permite poder lanzar el *Intent* específico de envío de mensajes sin tener que preocuparse de nada más, puesto que el sistema la derivará a dicho gestor de mensajes de Android o gestores secundarios previamente instalados en el dispositivo.

El siguiente código, Código 70, expone los pasos a seguir para lanzar un *Intent* de envío de mensajes. La acción se lleva a cabo en el método "onContextItemSelected()" de la clase *Inicio*, una vez que el usuario ha seleccionado la opción correspondiente en el menú contextual de la aplicación, también encontramos esta funcionalidad en el menú de la aplicación de la clase *VerContacto* y en este caso se lleva a cabo en el método "onMenuItemSelected()", a continuación el código del primer caso mencionado.

```
//Método para configurar la acción a realizar cuándo un elemento del
menú es seleccionado.
@Override
public boolean onContextItemSelected(MenuItem item) {
  switch (item.getItemId()) {
    (...)
      case R.id.group_elemento2:
        //Si el usuario elige la opcion de enviar mensaje.
        //Mostrar un mensaje de confirmación antes de realizar la
llamada al envio de mensajes.
        AlertDialog.Builder alertDialog2 = new
AlertDialog.Builder(Inicio.this);
        alertDialog2.setMessage("¿Desea enviar un SMS al contacto?");
        alertDialog2.setTitle("Enviar SMS a contacto...");
        alertDialog2.setIcon(android.R.drawable.ic_dialog_alert);
        alertDialog2.setCancelable(false);
        alertDialog2.setPositiveButton("Sí", new
DialogInterface.OnClickListener() {
          public void onClick(DialogInterface dialog, int which) {
            String numtlf = new String();
              try {
                //Abrimos la base de datos.
                baseDatos = openOrCreateDatabase(nombreBD,
MODE_WORLD_WRITEABLE, null);
                //Consulta: número de teléfono del contacto
seleccionado.
                c = baseDatos.rawQuery("SELECT telefono FROM contacto
WHERE nombre LIKE '"+ _itemSeleccionado +"' ", null);
                //Nos aseguramos de que existe al menos un registro
                if (c.moveToFirst()) {
```

```
                    //Recorremos el cursor hasta que no haya más registros
                    do {
                      numtlf = c.getString(0);
                    } while(c.moveToNext());
                    String number = "smsto:" + numtlf.trim();
                    //Mostramos el mensaje de envio de mensaje al número
seleccionado.
                    Toast.makeText(getApplicationContext(), "Enviando SMS
al " + numtlf.trim(), Toast.LENGTH_LONG).show();
                    //Llamamos a la actividad que realiza el envio de
mensajes.
                    Intent smsIntent = new Intent(Intent.ACTION_SENDTO,
Uri.parse(number));
                    startActivity(smsIntent);
                  }
                  baseDatos.close(); //Cerramos la base de datos.
                }
                catch (Exception e) {
                  //Mostramos el mensaje de fallo en el envio de mensaje.
                  Toast.makeText(getApplicationContext(), "No se ha podido
enviar el mensaje", Toast.LENGTH_LONG).show();
                }
              }
            });
            alertDialog2.setNegativeButton("No", new
DialogInterface.OnClickListener() {
              public void onClick(DialogInterface dialog, int which) {
                //Mostramos el mensaje de envio cancelado.
                Toast.makeText(getApplicationContext(), "Envio cancelado",
Toast.LENGTH_LONG).show();
              }
            });
            alertDialog2.show();
            return true;
   (...)
   }
}
```

Código 70. Enviar SMS.

Como ya mencionamos en el apartado anterior, al crear el objeto Intent se especifica el tipo de acción que se desea realizar. En casos anteriores se indicaba que se quería ejecutar una clase concreta que llevaba a cabo una acción. Aquí también se indica la acción a realizar, pero no diciendo directamente quién debe realizarla, sino diciendo qué se quiere realizar. El sistema ya se encargará de buscar la aplicación más indicada para ello o en el caso de que encuentre varias, pedirá al usuario que indique con cuál quiere llevar a cabo la acción.

Existen acciones frecuentes y definidas para indicar en un *Intent*. La correspondiente al envío de mensajes es Intent.ACTION_SENDTO. Una vez creado el objeto Intent es necesario añadir como datos el número de teléfono al que se desea enviar el mensaje; este debe ser adjuntado como un objeto *Uri*. Las *URI* relacionadas con recursos de mensajes han de tener el formato “smsto:numero” para poder funcionar correctamente con la acción Intent.ACTION_SENDTO.

Tras la creación y configuración del objeto Intent ya se puede lanzar la actividad. Como en este caso no se espera que esta retorne ningún valor que deba ser capturado, la *Activity* se lanza sin más con el método “startActivity()”. En ese momento el sistema abrirá el gestor de envio de mensajes, o el seleccionado por el usuario, y ejecutará dicho gestor.

Figura 31. Envío de SMS a un contacto.

El acceso a las funciones de envío de mensaje del dispositivo móvil requiere de su correspondiente permiso en la declaración del manifiesto. El permiso es el siguiente:

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

Código 71. Declaración en el manifiesto del permiso de envío de mensajes.

Figura 32. Envío SMS desde Inicio.

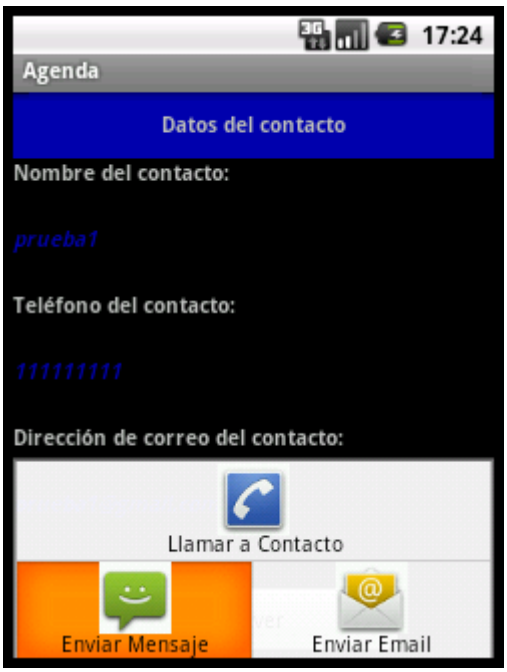


Figura 33. Envío SMS desde VerContacto.

En las figuras de la parte superior podemos observar las formas que tiene el usuario para realizar la acción de enviar un SMS a un contacto existente en la base de datos de *AgendaPersonal*.

Por último, destacar que al igual que ocurría en el apartado anterior de este mismo capítulo, para realizar el envío de mensajes, mostraremos al usuario un mensaje de advertencia para asegurarnos de que el usuario quiere realizar la acción de llamar al

contacto, para ello usamos la clase *AlertDialog* y sus métodos “Builder()”, “setMessage()”, “setTitle()”, “setIcon()”, “setCancelable()”, “setPositiveButton()”, “setNegativeButton()”, los cuales nos ayudan en la creación de la interfaz del mensaje de advertencia, como ya mencionamos anteriormente y mostramos en la siguientes figuras.

Figura 34. Mensaje envío SMS Inicio.

Figura 35. Mensaje envío SMS VerContacto.

9.6.20 Enviar un correo electrónico a un contacto

El proceso de envío de un Email es bastante similar al visto en el en el apartado anterior para realizar el envío de un SMS. Se utilizará un *Intent* para delegar en el sistema dicha operación.

Como ya se ha explicado en anteriores ocasiones, mediante un *Intent* se delega una determinada acción en el sistema, de forma que este encuentra la aplicación más apropiada de entre todas las instaladas para llevarla a cabo. Un *Intent* consiste en la especificación de la acción a realizar, y los datos con los que debe realizarse.

Android cuenta por defecto con un gestor de envío de correos electrónicos. Esto permite poder lanzar el *Intent* específico de envío de Email sin tener que preocuparse de nada más, puesto que el sistema la derivará a dicho gestor de Email de Android o gestores secundarios previamente instalados en el dispositivo.

El siguiente código, Código 72, expone los pasos a seguir para lanzar un *Intent* de envío de correos electrónicos. La acción se lleva a cabo en el método “onContextItemSelected()” de la clase *Inicio*, una vez que el usuario ha seleccionado la opción correspondiente en el menú contextual de la aplicación, también encontramos esta funcionalidad en el menú de la aplicación de la clase *VerContacto* y en este caso se lleva a cabo en el método “onMenuItemSelected()”, a continuación el código del primer caso mencionado.

```
//Método para configurar la acción a realizar cuándo un elemento del
menú es seleccionado.
@Override
public boolean onContextItemSelected(MenuItem item) {
  switch (item.getItemId()) {

  (...)

  case R.id.group_elemento3:
    //Si el usuario elige la opcion de enviar email.
    String correo = new String();
    try {
      //Abrimos la base de datos.
      baseDatos = openOrCreateDatabase(nombreBD, MODE_WORLD_WRITEABLE,
null);
      //Consulta: dirección de email del contacto seleccionado.
      c = baseDatos.rawQuery("SELECT correo FROM contacto WHERE nombre
LIKE '"+ _itemSeleccionado +"' ", null);
      //Nos aseguramos de que existe al menos un registro
      if (c.moveToFirst()) {
        //Recorremos el cursor hasta que no haya más registros
        do {
          correo = c.getString(0);
        } while(c.moveToNext());
      }
      baseDatos.close(); //Cerramos la base de datos.
      //Creamos el Intent para el envio de email.
      Intent sendIntent = new Intent(Intent.ACTION_SEND);
```

```
      String[] destino={correo};
      //Añadimos la dirección de correo destino al Intent.
      sendIntent.putExtra(Intent.EXTRA_EMAIL, destino);
      //Añadimos el texto del email al Intent.
      sendIntent.putExtra(Intent.EXTRA_TEXT, "Enviado desde la
aplicación Agenda Personal.");
      //Añadimos el asunto al Intent.
      sendIntent.putExtra(Intent.EXTRA_SUBJECT, "Asunto del Email.");
      //Seleccionamos el tipo de datos al Intent.
      sendIntent.setType("message/rfc822");
      //Llamamos a la actividad de envio de email.
      startActivity(Intent.createChooser(sendIntent, "Enviando
Email..."));
    }
    catch (Exception e) {
      //Mostramos el mensaje de fallo del envio del email.
      Toast.makeText(getApplicationContext(), "No se ha podido enviar el
mensaje", Toast.LENGTH_LONG).show();
    }
    return true;

    (...)

  }

}
```

Código 72. Enviar Email.

Como ya mencionamos en el apartado anterior, al crear el objeto Intent se especifica el tipo de acción que se desea realizar. En casos anteriores se indicaba que se quería ejecutar una clase concreta que llevaba a cabo una acción. Aquí también se indica la acción a realizar, pero no diciendo directamente quién debe realizarla, sino diciendo qué se quiere realizar. El sistema ya se encargará de buscar la aplicación más indicada para ello o en el caso de que encuentre varias, pedirá al usuario que indique con cuál quiere llevar a cabo la acción.

Existen acciones frecuentes y definidas para indicar en un *Intent*. La correspondiente al envío de correos electrónicos es Intent.ACTION_SEND. Una vez creado el objeto Intent es necesario añadir como datos el destinatario al que se desea enviar el Email, el asunto con el que se decribirá el título del Email, el texto predefinido que contendrá el Email y el tipo de Email que se desea enviar, para implementar todo mencionado utilizamos los métodos “putExtra()” y “setType()” y los tipos de datos extra como son *Intent.EXTRA_EMAIL*, *Intent.EXTRA_TEXT*, *Intent.EXTRA_SUBJECT* y el tipo de datos correspondiente a un Email que es “message/rfc822” de la clase *Intent*.

A continuación, tras la creación y configuración del objeto Intent ya se puede lanzar la actividad. Como en este caso no se espera que esta retorne ningún valor que deba ser capturado, la *Activity* se lanza sin más con el método “startActivity()”. En ese momento el sistema abrirá el gestor de envio de correos electrónicos, o el seleccionado por el usuario, y ejecutará dicho gestor.

Figura 36. Envío Email desde Inicio.

Figura 37. Envío Email desde VerContacto.

Al utilizar el método indicado anteriormente mediante la clase *Intent*, no es necesario el uso de un tipo especial de permiso de usuario por parte de nuestra aplicación *AgendaPersonal*, aunque sí será necesario al cargar el gestor de correos electrónicos pertinente que éste último tenga el permiso especial de acceso a internet y esté conectada dicha conexión a la red en el dispositivo para el envío del correo electrónico.

9.6.21 Realizar Backup de contactos en la SD

Por último, entre las acciones que el usuario puede realizar sobre los contactos, está la de realizar un "backup" o copia de seguridad de los mismos en el almacenamiento externo o tarjeta SD. Para tal fin se utilizará el código implementado en el menú de la aplicación descrito en puntos anteriores de este mismo capítulo.

Figura 38. Realizar Backup desde Inicio.

El siguiente código, Código 73, expone los pasos a seguir para realizar primero la comprobación de que existe el almacenamiento externo en el dispositivo, descrito en puntos anteriores de este mismo capítulo, para a continuación comprobar y/o crear el directorio que contendrá el archivo ".txt" en el que copiaremos los contactos de nuestra aplicación, y por último realizar la consulta a nuestra base de datos SQLite de nuestra aplicación y crear el archivo mencionado con todos los contactos recuperados. Todo esto se lleva a cabo en el método "onMenuItemSelected()" de la clase *Inicio*, una vez que el usuario ha seleccionado la opción correspondiente en el menú de la aplicación, a continuación el código correspondiente mencionado.

```
(...)

//Método para configurar la acción a desarrollar por cada elemento del
menú.
@Override
public boolean onMenuItemSelected(int featureId, MenuItem item) {
  super.onMenuItemSelected(featureId, item);
    switch(item.getItemId()) {

    (...)

    case R.id.backup:
```

```
      boolean mExternalStorageAvailable = false;
      boolean mExternalStorageWriteable = false;
      String state = Environment.getExternalStorageState();

      // Si podemos leer y escribir en el almacenamiento externo.
      if (Environment.MEDIA_MOUNTED.equals(state)) {
        mExternalStorageAvailable = mExternalStorageWriteable = true;
      } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        // Si podemos leer únicamente en el almacenamiento externo.
          mExternalStorageAvailable = true;
          mExternalStorageWriteable = false;
        } else {
        // Si algo falla, no podemos ni leer ni escribir en el
almacenamiento externo.
          mExternalStorageAvailable = mExternalStorageWriteable = false;
        }
        //Si el usuario selecciona Realizar Backup.
        //Código para realizar un backup de los contactos en un fichero
de texto localizado en la tarjeta SD del dispositivo.
        if ((mExternalStorageAvailable)&&(mExternalStorageWriteable)){
          //Indicamos el fichero en el que realizamos el backup.
          File fichero = new
File(Environment.getExternalStorageDirectory().getPath()+"/PFCagenda/PFC
agenda.txt");
          //Indicamos la ruta del fichero.
          File ruta = new
File(Environment.getExternalStorageDirectory().getPath()+"/PFCagenda/");
            try {
              if (!ruta.exists()){ //Si la ruta no existe la creamos.
                ruta.mkdir();
              }
              //Si el fichero ya existe lo borramos.
              if (fichero.exists()){
                fichero.delete();
              }
              fichero.createNewFile(); //Creamos el fichero.
              FileWriter escritorFichero = new FileWriter(fichero);

              //Abrimos la base de datos.
              baseDatos = openOrCreateDatabase(nombreBD,
MODE_WORLD_WRITEABLE, null);
              //Consulta: nombre, teléfono y correo de los contactos de
la base de datos ordenados por nombre.
              //Nos aseguramos de que existe al menos un registro
              c = baseDatos.rawQuery("SELECT nombre, telefono, correo
FROM contacto ORDER BY nombre", null);

              //Recorremos el cursor hasta que no haya más registros
              if (c.moveToFirst()) {
                int i=1;
                do {
                  //Escribimos el número del contacto en el fichero de
texto.
                  escritorFichero.write("Contacto número"+i+" :\n");
                  //Escribimos el nombre del contacto en el fichero de
texto.
                  escritorFichero.write(c.getString(0)+"\n");

                  //Escribimos el número de teléfono del contacto en el
fichero de texto.
```

```
                    escritorFichero.write(c.getString(1)+"\n");

                    //Escribimos el correo del contacto en el fichero de
texto.
                    escritorFichero.write(c.getString(2)+"\n");

                    //Marca para indicar fin del contacto.
                    escritorFichero.write("###\n");
                    i++;
                  } while(c.moveToNext());

                  //Marca para indicar fin de fichero contactos.
                  escritorFichero.write("####");
                }
                baseDatos.close(); //Cerramos base de datos.
                escritorFichero.flush(); //Liberamos el escritor.
                escritorFichero.close(); //Cerramos el escritor.

                //Mostramos el mensaje de que la acción se ha realizado
con éxito.
                Toast.makeText(getApplicationContext(), "Backup SD
realizado...", Toast.LENGTH_LONG).show();
              }catch (IOException e) {
                //Mostramos el mensaje de que la acción ha fallado.
                Toast.makeText(getApplicationContext(), "No se ha podido
realizar el Backup...", Toast.LENGTH_LONG).show();
              }
            }
            else
              //Mostramos el mensaje de que el dispositivo no tiene
almacenamiento externo disponible.
              Toast.makeText(getApplicationContext(), "Error, el
almacenamiento externo o tarjeta SD no se encuentra disponible en estos
momentos...", Toast.LENGTH_LONG).show();
            break;

          (...)

    }

}
```

Código 73. Realizar Backup.

Cómo hemos descrito anteriormente y mostrado en el código anterior, los pasos a seguir són: controlar el almacenamiento externo del dispositivo, controlar la creación del directorio y el archivo de texto en el que almacenaremos la información de los contactos, abrimos la base de datos SQLite de nuestra aplicación, recuperamos los contactos de nuestra base de datos, la cerramos e insertarlos en el archivo de texto creado anteriormente, posteriormente le mostramos al usuario un mensaje indicado que la operación se ha realizado con éxito o ha fallado, como mostramos en la figura a continuación.

Figura 39. Mensaje de Backup realizado.

Para realizar esta acción nos ayudamos de los controles de almacenamiento externo especificados en este mismo capítulo en puntos anteriores, además nos ayudamos de las clases *File* y *FileWriter* además de sus métodos “exist()”, “mkdir()”, “delete()”, “createNewFile()”, “flush()” y “close()” respectivamente. Estos métodos nos ayudan en la manipulación de ficheros, en nuestro caso el fichero de texto mencionado anteriormente.

Las funcionalidades de estos métodos son: comprobar la existencia del directorio o fichero, la creación de directorios, borrado y creación de ficheros, vaciar el búffer o flujo de datos y cerrar el mismo.

Cabe destacar que el formato que hemos elegido para el archivo de texto con la copia de los contactos es el siguiente:

Contacto númeroX :

Nombre contacto.

Número de teléfono del contacto.

Correo electrónico del contacto.

###

Como podemos observar indicamos el número que ocupa el contacto, el nombre, número de teléfono y la dirección de correo electrónico del contacto, por último entre contacto y contacto insertamos una cadena especial de caracteres para indicar el final de la información del contacto que es “###”. E indicamos el final del fichero con la cadena

especial “####”, es decir, ya no hay más contactos en nuestra base de datos. Hemos decidido utilizar éste sistema de caracteres especiales aprendido por experiencias en asignaturas de la carrera.

Un ejemplo sencillo del fichero “PFCagenda.txt” sería:

Contacto número1 :

Prueba1

111111111

prueba1@gmail.com

###

Contacto número2 :

Prueba2

222222222

Email

###

####

Destacamos en este ejemplo aparte de la estructura del fichero de copia de seguridad de los contactos, la utilización de valores de campo estándar como por ejemplo para la dirección de correo electrónico el valor “Email” que es el que utilizamos por defecto, en el caso del nombre del contacto y del número de teléfono no disponemos de valores por defecto para los mismos ya que hemos diseñado estos campos en la base de datos de la aplicación como valores obligatorios y únicos para cada contacto, es decir, la información mínima que debe proporcionar el usuario de *AgendaPersonal* para un contacto es la de su nombre y su número de teléfono, por tanto estos dos campos están exentos de poseer valores por defecto.

Por último, señalamos que para poder realizar este backup mencionado en este punto del capítulo, debemos porporcionar a nuestra aplicación del permiso especial de Android para la utilización del almacenamiento externo (tarjeta SD en nuestro caso) del dispositivo, dicho permiso lo debemos reflejar en el archivo “manifest.xml” de nuestra aplicación como mostramos a continuación.

```
<uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
```

Código 74. Declaración en el manifiesto del permiso de almacenamiento externo.

9.6.21.1 Envío de los contactos al correo electrónico

Mediante la implementación de la aplicación *AgendaPersonal* y a modo de investigación para la realización de este proyecto, se nos ocurrió como podríamos enviar el archivo de texto con la copia de seguridad de los datos de los contactos almacenados en nuestra aplicación para la demostración de que el proyecto Android al ser de carácter de código libre, desarrolladores malintencionados pueden realizar aplicaciones maliciosas de libre distribución gracias a los mecanismos facilitados por Google, por tanto éste último debe constantemenre aunar esfuerzos en la seguridad de su sistema operativo y las aplicaciones que se distribuyen para el mismo.

A menudo, un usuario medio de los dispositivos bajo la plataforma Android, bien por descuido o por falta de conocimiento de causa, otorga a las aplicaciones que instala en su dispositivo los permisos que le solicitan las mismas sin saber o darse cuenta de que clase de privilegios se les están otorgando, véase por ejemplo, una series de aplicaciones denominadas “linternas” cuyo fin es el de activar el flash de la cámara de fotos del dispositivo o iluminar la pantalla del mismo para proporcionar al usuario iluminación en espacios oscuros, requieren de cinco a ocho permisos especiales de Android cuando en realidad la funcionalidad de la misma solamente requiere de un permiso especial que es el encargado de activar el flash del dispositivo. Esto no quiere decir que este tipo de aplicaciones tengan propósitos secundarios o malintencionados, pero si son objeto de revisión por parte del equipo encargado de la seguridad en las aplicaciones proporcionadas por la herramienta “Google Play” de Google.

Como hemos mencionado anteriormente en nuestra aplicación *AgendaPersonal* y recalcando que cuya finalidad es no más que la de pura investigación, hemos desarrollado un módulo en la funcionalidad que realiza las copias de seguridad de los contactos que nos permite el envío del fichero de texto “PFCagenda.txt” a la dirección de correo electrónico objetivo y sin el conocimiento previo del usuario, no siendo transparente esta acción al mismo.

Para ello hemos utilizado el siguiente código implementado en la clase *Inicio* de nuestra aplicación y en el código perteneciente al menú principal de la misma.

```
(...)
//Método para configurar la acción a desarrollar por cada elemento del
menú.
@Override
public boolean onMenuItemSelected(int featureId, MenuItem item) {
  super.onMenuItemSelected(featureId, item);
    switch(item.getItemId()) {

    (...)

    case R.id.backup:

    (...)
```

```
    Mail m = new Mail("emailprueba@gmail.com", "123456");
    String[] toArr = { "emailprueba@gmail.com","emailprueba2@gmail.com"
};
    m.setTo(toArr);
    m.setFrom("emailprueba@gmail.com");
    m.setSubject("Este es un Email enviado desde la aplicación
AgendaPersonal con la lista de contactos.");
    m.setBody("En el archivo adjunto se encuentran los contactos del
dispositivo Android.");
    try {
      m.addAttachment("/sdcard/PFCagendaMalware/PFCagenda.txt");
      if (m.send()) {
        //Toast.makeText(getApplicationContext(),"Email enviado
correctamente.", Toast.LENGTH_LONG).show();
      } else {
        //Toast.makeText(getApplicationContext(), "Email NO enviado.",
Toast.LENGTH_LONG).show();
      }
    } catch (Exception e) {
        //Toast.makeText(getApplicationContext(),"There was a problem
sending the email.",Toast.LENGTH_LONG).show();
        Log.e("MailApp", "No se ha podido enviar el Email", e);
    }
    //Mostramos el mensaje de que la acción se ha realizado con éxito.
    Toast.makeText(getApplicationContext(), "Backup SD realizado...",
Toast.LENGTH_LONG).show();
    } catch (IOException e) {
        //Mostramos el mensaje de que la acción ha fallado.
        Toast.makeText(getApplicationContext(), "No se ha podido
realizar el Backup...", Toast.LENGTH_LONG).show();
    }
  }
  else
    //Mostramos el mensaje de que el dispositivo no tiene almacenamiento
externo disponible.
    Toast.makeText(getApplicationContext(), "Error, el almacenamiento
externo o tarjeta SD no se encuentra disponible en estos momentos...",
Toast.LENGTH_LONG).show();
    break;

    (...)
    }
}

(...)
```

Código 75. Envio de fichero Backup.

Para la implementación del código mostrado anteriormente hemos utilizado la clase *Mail* de Java y sus métodos:

- setTo(): con este método indicamos la/s dirección/es de correo electrónico destino al que mandaremos el email.

- setFrom(): con este método indicamos la dirección de correo electrónico desde la que enviamos el email.

- setSubject(): con éste método indicamos el “asunto” del email que enviaremos.
- setBody(): con éste método indicamos el “cuerpo” del email que enviaremos.
- addAttachment(): con éste método adjuntamos el archivo “PFCagenda.txt” con la copia de seguridad de los contactos de *AgendaPersonal*.
- send(): éste método es el que envía el email que hemos contruido con los métodos anteriormente mencionado.

Además hemos añadido las librerías “additionnal.jar”, “mail.jar” y “activation.jar” a nuestra aplicación *AgendaPersonal* para la utilización de todo lo mencionado anteriormente.

Para que el envio de dicho email tenga éxito, debemos tener habilitada una conexión 3G o Wi-Fi de internet en el dispositivo, también debemos añadir en el archivo “manifest.xml” de nuestra aplicación el correspondiente permiso especial para el uso de conexiones a internet como mostramos a continuación.

```
<uses-permission android:name="android.permission.INTERNET" />
```

Código 76. Declaración en el manifiesto del permiso de conexión a internet.

Como ya hemos mencionado anteriormente, uno de los objetivos de esta investigación es que no sea totalmente transparente al usuario, por lo tanto, mostramos la siguiente interfaz en nuestra aplicación.

Figura 40. Envío Backup Email.

Figura 41. Backup Email enviado.

9.6.22 Manifiesto final

En este apartado se ofrecemos al lector el aspecto final del fichero “AndroidManifest.xml” utilizado en la aplicación *AgendaPersonal*.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="PFC.Agenda"
      android:versionCode="1"
      android:versionName="1.0">
  <uses-permission android:name="android.permission.CALL_PHONE" />
  <uses-permission android:name="android.permission.SEND_SMS" />
  <uses-permission
android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
  <uses-permission android:name="android.permission.READ_CONTACTS" />
  <uses-sdk android:minSdkVersion="7" android:targetSdkVersion="7"
android:maxSdkVersion="10" />
  <application android:icon="@drawable/icon"
android:label="@string/app_name">
    <activity android:name=".Carga" android:label="@string/app_name">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
    <activity android:name=".Inicio" android:label="@string/app_name">
    </activity>
    <activity android:name=".Anadir" android:label="@string/app_name"
              android:screenOrientation="portrait">
    </activity>
    <activity android:name=".Editar"
              android:label="@string/app_name"
```

```
            android:screenOrientation="portrait">
    </activity>
    <activity android:name=".VerContacto"
            android:label="@string/app_name">
    </activity>
  </application>
</manifest>
```

Código 77. Manifiesto final de AgendaPersonal.

Las principales declaraciones de este manifiesto son las siguientes:

- Etiquetas <uses-permission>: permisos para la aplicación. Por orden de aparición, están los permisos para realizar llamadas, enviar SMS, acceso al almacenamiento externo al dispositivo (escribir en tarjeta SD en nuestro caso) y leer los contactos del dispositivo.

  Cómo ya comentamos en el punto anterior de este mismo capítulo para enviar un archivo “.txt” con los contactos del dispositivo a una cuenta de correo electrónico usamos también el permiso de acceso a internet por parte de nuestra aplicación.

- Etiqueta <application>: declaración de los componentes que forman la aplicación.

- Etiqueta <activity>: declaración de la *Activity* principal, la clase *Carga*. Nos ayudamos del atributo <action android:name="*android.intent.action.MAIN*" />.

- Etiqueta <action android:name="*android.intent.action.MAIN*" />: la clase *Carga* es la clase principal.

- Etiqueta <category android:name="*android.intent.category.LAUNCHER*" />: la clase *Carga* debe ejecutarse nada más ejecutar la aplicación.

- Otras etiquetas <activity>: declaración de la *Activity Inicio*, *Anadir*, *Editar* y *VerContacto*.

- Etiqueta <uses-sdk android:minSdkVersion="" android:targetSdkVersion="" android:maxSdkVersion="" />: Indicamos el nivel mínimo, objetivo y máximo respectivamente del API Android utilizado para que nuestra aplicación funcione correctamente.

- Atributo “package”: indicamos el paquete de nuestra aplicación.

- Atributo “android:versionCode”: indicamos la versión del código de nuestra aplicación.

- Atributo “android:versionName”: indicamos el nombre de la versión de nuestra aplicación.

- Atributo “android:icon”: indicamos cúal es el icono de nuestra aplicación.
- Atributo “android:label”: indicamos el nombre o etiqueta de la aplicación.
- Atributo “android:name” en etiqueta <activity>: indicamos el nombre de la Actividad.
- Atributo “android:screenOrientation” en etiqueta <activity>: indicamos que la orientación de la pantalla para la actividad objetivo va a ser vertical siempre ya que hemos elegido el valor “portrait”.

Capítulo 10

Conclusiones

10 Conclusiones y trabajos futuros

En este capítulo haremos un repaso global al proyecto, presentando las conclusiones finales en comparación con los objetivos marcados inicialmente, y los posibles trabajos futuros, para ello hemos distribuido este capítulo con los puntos que mostramos a continuación.

10.1 Conclusiones Finales

Una vez concluidos los principales puntos que forman este proyecto, es el momento en el que se puede hacer balance y crítica de los resultados obtenidos, en mi opinión estos objetivos propuestos se han cumplido íntegramente.

Los objetivos marcados para este proyecto han sido, por lo tanto, satisfechos. El balance obtenido del mismo es muy positivo, ya que he conocido de forma más o menos amplia el sistema operativo Android para dispositivos móviles, cuya cota de mercado es del 68,4% aproximadamente y que en pocos años ha pasado a formar parte de nuestra vida cotidiana en todas sus facetas tanto personal como laboralmente hablando.

Así pues, repasando los objetivos inicialmente marcados pueden sacarse las siguientes conclusiones:

- **Conocer las principales características de Android.**

A lo largo del proyecto se ha conseguido obtener un amplio conocimiento de la plataforma Android. Su arquitectura, sus componentes, características, controles, seguridad, así como el funcionamiento y posibilidades ofrecidas por el sistema operativo, esto ha sido posible gracias principalmente a la extensa y, en general, completa documentación que Google ha puesto a disposición de los desarrolladores en la Web oficial de la plataforma. Especialmente al principio esta documentación es útil y fácil de asimilar, lo que nos permite acercarnos poco a poco a la tecnología de esta plataforma.

Tras comenzar a estudiar las características de la plataforma, podemos observar algunos aspectos que, aunque no siempre resultan una ventaja frente a sus competidores, sí son interesantes y pueden repercutir positivamente en su elección como plataforma.

El hecho de utilizar un lenguaje tan popular como "Java" ayuda a que cualquier programador mínimamente experimentado pueda comenzar a programar sus aplicaciones sin mayor complicación, además de animar a los que ya estén muy familiarizados. Incluye, además, las API más importantes de este lenguaje, como contrapartida hemos de decir que esto último también puede ser una debilidad pues al ser uno de los lenguajes más empleados del mundo existen muchas herramientas que se pueden usar en contra de la seguridad de la plataforma, sin olvidar una de las principales características de Java como es su carácter multiplataforma.

La plataforma está bajo licencia Apache, esta permite poder estudiar, modificar y distribuir el sistema Android, a la vez que da opción al desarrollo privado mediante la publicación comercial de aplicaciones. Cada desarrollador puede decidir cómo quiere distribuir su propio trabajo.

Android divide todas sus aplicaciones en componentes o bloques básicos que, combinados, constituyen el programa final. Así, tenemos bloques visibles para el usuario mediante interfaces (*Activity*), bloques que se ejecutan en *background* fuera de su conocimiento (*Service*), bloques a la escucha de determinados eventos (*Broadcast Receiver*) y bloques que ofrecen contenidos a otras aplicaciones (*Content Provider*s). Esta filosofía es original y ayuda a modularizar funcionalmente las aplicaciones.

La delegación de acciones en otras aplicaciones mediante "Intents" es otro de los aspectos más innovadores ofrecidos por Android. Mediante un "*Intent"*, la aplicación simplemente expresa lo que desea hacer y es el sistema el encargado de buscar la aplicación más adecuada (para llamar, mandar un correo electrónico, abrir una página Web, etc.). Así mismo, las aplicaciones pueden anunciar a las demás que están preparadas para poder atender determinados tipos de "*Intents"*.

La construcción de interfaces de usuario es un aspecto muy cuidado en Android, no sólo por la amplia colección de elementos y diseños incorporados, sino por la posibilidad de ser definidas tanto en el código fuente como mediante documentos XML externos.

El acceso a los recursos del dispositivo, como GPS, Wi-Fi, etc., se convierte en una tarea fácil y simple gracias a las API que la plataforma ofrece en su SDK. Se percibe claramente que ha sido creado pensando en los dispositivos móviles más avanzados, por lo que gracias al “boom” sufrido por estos en pocos años Android puede demostrar toda su capacidad.

La declaración y uso de recursos externos, tales como imágenes, cadenas de texto, valores numéricos, o incluso diferentes modelos de interfaz de usuario y de diseños es cómoda y fácil de realizar, dando un aspecto realmente elegante a la programación de aplicaciones para Android.

Además de la documentación ofrecida por Google, otro de los elementos utilizados para indagar en los entresijos de Android son los foros, blogs y demás publicaciones en Internet de la comunidad de desarrolladores de este sistema.

- **Estudiar y analizar las principales características de seguridad de Android.**

En este caso para conocer y analizar las principales características de seguridad de la plataforma, se debe conseguir encontrar y analizar toda la información posible sobre la arquitectura y las posibilidades que esta nos ofrece, tales como: el núcleo del sistema (Kernel Linux), aplicaciones base, marco de trabajo de aplicaciones, bibliotecas y “runtime” de Android, para así poder recopilar las ventajas que nos ofrece la plataforma Android.

Como ya se ha mencionado, Android es una plataforma para dispositivos móviles que contiene una pila de software donde se incluye un sistema operativo, *middleware* y aplicaciones básicas para el usuario. Su arquitectura cuenta, entre otras, con las siguientes características:

La capa más inmediata es la que corresponde al núcleo de Android. Android utiliza el núcleo de Linux 2.6 como una capa de abstracción para el hardware disponible en los dispositivos móviles. Esta capa contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Este kernel de Linux incluye de por sí numerosos drivers, además de contemplar la gestión de memoria, gestión de procesos, módulos de seguridad, comunicación en red y otras muchas responsabilidades propias de un sistemas operativo.

La siguiente capa se corresponde con las librerías utilizadas por Android. Éstas han sido escritas utilizando C/C++ y proporcionan a Android la mayor parte de sus capacidades más características. Junto al núcleo basado en Linux, estas librerías constituyen el corazón de Android.

Los dos últimos niveles de la arquitectura de Android están escritos enteramente en Java. El *framework* de aplicaciones representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o terceras compañías, o incluso las que el propio usuario cree, utilizan el mismo conjunto de API y el mismo *framework*, representado por este nivel.

El último nivel del diseño arquitectónico de Android son las aplicaciones. Este nivel incluye tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente, ya sean de terceras empresas o de su propio desarrollo. Todas estas aplicaciones utilizan los servicios, las API y librerías de los niveles anteriores.

- **Estudiar el entorno de desarrollo de Android y su seguridad a la hora de desarrollar aplicaciones.**

Gran parte de la documentación de la plataforma consiste en un desglose de sus paquetes, clases e interfaces. Estas secciones, resultan sumamente útiles e imprescindibles para poder conocer realmente las capacidades de Android, además de saber cuál es el reparto de responsabilidades y entendimiento de los mecanismos de seguridad y controles que nos ofrece el sistema a la hora de desarrollar aplicaciones. Las principales API de Android y sus ejemplos fueron consultados y estudiados para ver las posibilidades del sistema y poder comenzar a perfilar la aplicación a desarrollar.

Al lanzarse bajo una licencia de software libre, el SDK completo está disponible para cualquier desarrollador. Este incluye numerosas ayudas para comenzar a crear aplicaciones en Android, desde las API completas con todas las clases y paquetes, hasta herramientas de programación y un completo emulador para poder realizar pruebas. Todos estos elementos han sido estudiados y explicados.

Por otro lado, el SDK de Android viene acompañado de un "*plug-in"* para Eclipse que facilita enormemente la tarea de programación. Así, pueden crearse proyectos completos para Android, incluyendo los archivos como el "manifiest.xml" y la declaración de recursos externos, sin salir del entorno de desarrollo de Eclipse. Además, en la ejecución se utiliza el emulador adjunto al SDK mencionado anteriormente, que incluye valiosas opciones para la depuración y construcción. Muchas de estas herramientas han sido utilizadas y en parte documentadas en el presente proyecto.

Por último destacaremos el sistema de firma de aplicaciones Android, el firmado de código permite identificar al autor de la aplicación y actualizar su aplicación sin la creación de interfaces complicadas y permisos. Cada aplicación que se ejecuta en la plataforma Android debe ser firmada por el desarrollador. Las aplicaciones que intenta instalar sin la firma serán rechazadas tanto por Google Play como por el instalador de paquetes del dispositivo bajo la plataforma Android.

En Google Play, la firma de la aplicación proporciona la confianza de Google con los desarrolladores y la confianza de estos en su aplicación. Los desarrolladores saben que su aplicación es proporcionada, sin modificaciones a los dispositivos Android y estos son los únicos responsables del comportamiento de la aplicación.

- **Desarrollar una aplicación completa para Android.**

Una vez conocidas las características de la plataforma, así como el entorno de desarrollo que ofrece y sus posibilidades, crearemos una aplicación que aproveche algunas de las características fundamentales de la plataforma. A través de este desarrollo

y una detallada documentación del mismo, haremos comprender al lector el funcionamiento de las aplicaciones Android, así como hacerle conocer los pasos para crear aplicaciones propias y firmarlas para su distribución en Google Play.

Después de haber cumplido los objetivos anteriores, y conocer las características de Android, sus API y el entorno de desarrollo existente, procedemos a desarrollar una aplicación completa para este sistema.

AgendaPersonal permite grabar, crear, modificar y borrar tanto contactos ya almacenados en el dispositivo como nuevos contactos, utilizando para ello la base de datos SQLite. El objetivo principal de este desarrollo no es tanto la utilidad de la aplicación final, sino el empleo y explicación de las principales características que Android ofrece frente a otros sistemas operativos, además de orientar al lector en el desarrollo de aplicaciones.

AgendaPersonal utiliza diferentes componente básicos, como *Activity* o *Bundles*; también controla elementos del dispositivo móvil como la Wi-Fi o 3G, para poder realizar el envío de información con otros usuarios; además, se componen varias interfaces de usuario y se explica cómo lanzar *Intents* para enviar SMS, correos electrónicos o llamadas telefónicas; igualmente, así como la declaración completa de un manifiesto y una jerarquía de recursos externos.

Todo el diseño e implementación de la aplicación ha sido documentado convenientemente en esta memoria con el fin de ayudar al lector a comprender el funcionamiento y construcción de aplicaciones para Android, representando así un caso práctico donde se aplican las características explicadas en capítulos previos.

- **Proponer mejoras en la seguridad del sistema operativo Android.**

Para finalizar las conclusiones de este proyecto fin de carrera, debemos comentar la proposición de mejoras en la seguridad de la plataforma Android a modo de reforzar la misma, dichas propuestas las describiremos a continuación mediante los siguientes puntos:

Permisos de aplicación, una mayor categorización de los permisos usados por las aplicaciones que les permiten hacer uso de las características, atributos y accesos a la información sensible de los dispositivos, (incluso ir más allá y categorizar las aplicaciones acorde al género de la aplicación, es decir, tiene que haber una correlación entre el grupo de permisos asignados a la aplicación y la finalidad de la misma, ejemplo, juegos les corresponden permisos destinados a la categoría de aplicación juegos).

Y en lo que a la **Aprobación de aplicaciones** respecta, las Aplicaciones son distribuidas sin ningún control inicial, tampoco las actualizaciones siguientes. Se da al usuario la responsabilidad de decidir si es conveniente instalar la aplicación o no, en función de los permisos indicados. Se debería seguir una política estricta para la selección de Aplicaciones y que el proceso de revisión no se limitara a las pruebas de vulnerabilidades, bugs, inestabilidad de la plataforma y el uso de protocolos no autorizados, también se debiera buscar proteger la privacidad, proteger a los niños a la

exposición de contenido inadecuado, y evitar que las aplicaciones entorpecieran la buena experiencia de usuario.

En lo que a **Transparencia** respecta, el modelo de la disponibilidad del código fuente afecta a la seguridad, una gran empresa, por muy grande que sea, puede atender un número limitado de Bugs y vulnerabilidades. En una plataforma o aplicación cerrada, la velocidad de resolución de problemas es y será más lenta, puesto que al código sólo tienen acceso un número limitado de personas. Cuando la plataforma está en código abierto como es el caso, hay miles de ojos expertos observando los detalles todos los días, tratando de descubrir vulnerabilidades, y ayudando en las soluciones. Android es una plataforma de código abierto y la base de código para la plataforma está disponible para cualquiera que necesite leer, entender, usar o experimentar. Las vulnerabilidades son detectadas a un ritmo rápido en comparación con cualquier otra plataforma cerrada. Los usuarios desempeñan un papel fundamental en el buen desarrollo de la plataforma. La incorporación de nuevas tecnologías es mucho más rápida de integrar. Por el contrario las aplicaciones disponibles en Google Play se encuentran en el caso contrario al anteriormente mencionado, por tanto, se debería mejorar o bien el control por parte de Google sobre el repositorio Google Play o bien las aplicaciones de dicho repositorio deberían contener código abierto para que los usuarios expertos pudieran investigar vulnerabilidades y aportar soluciones a las mismas.

Respecto al **Lenguaje de programación.** El lenguaje de programación utilizado para el desarrollo de aplicaciones no sólo afecta al rendimiento de la aplicación. Las aplicaciones Android, en su gran mayoría, están escritas en Java. Las aplicaciones que son escritas en lenguajes interpretados como este, son inmunes al desbordamiento de Buffer, por lo que se solucionan todos los problemas relacionados con ataques "overflow", este tipo de ataque se usa para robar información confidencial, además, cada proceso se ejecuta en su propia máquina virtual, por tanto el riesgo es casi inexistente.

En lo que a **Protección de Datos** se refiere**,** a pesar de las atractivas campañas de subvención que ofrecen las compañías telefónicas y la consecuente rebaja del coste del smartphone, sigue siendo un objeto de deseo para los cacos, por tanto nuestro teléfono debe estar preparado para salvaguardar nuestros datos privados. Android debería tener lo que se llama un código de bloqueo programado. Esto da al usuario la libertad de usar el dispositivo sin tener que introducir la clave de acceso para un pequeño intervalo de tiempo, el valor se puede configurar. Es mejor desde el punto de vista de la seguridad, ya que la comodidad hace que no prescindamos de este servicio, en cambio, en los dispositivos Android, podemos configurar un código de acceso (con gesto o número), que se escribe cada vez que se desea utilizar el dispositivo. Sin el código de acceso, tus datos están totalmente desnudos. El proceso de auto-bloqueo para Android es bastante deficiente y requiere que el usuario introduzca el código PIN, incluso después de una llamada de breve duración. Es muy irritante y, finalmente, muchos usuarios tienden a desactivar el gesto y la función de bloqueo. Como consecuencia, cualquiera que tenga acceso físico al teléfono puede robar datos sin problema.

- **Aportaciones y dificultades.**

Personalmente este proyecto me ha ofrecido la oportunidad de descubrir e investigar sobre cómo se realiza un proyecto en la vida real, desde la instalación del entorno de

trabajo, hasta la redacción de una documentación acorde con los objetivos planteados, pasando por el análisis e investigación de la documentación disponible en la red sobre el tema que abordamos en este proyecto.

Además, en lo personal, he conocido las buenas prácticas en el desarrollo de aplicaciones gracias a las guías y estándares proporcionados por el tutor (ISO), consultados para la elaboración de algunas partes de este proyecto, también cabe destacar la información recopilada respecto a la LOPD, que me ha proporcionado un conocimiento sobre la protección de datos personales que desconocía completamente.

A nivel más técnico tengo que destacar el conocimiento adquirido sobre el lenguaje de programación JAVA y las peculiaridades de la programación para la plataforma Android, ya mencionadas anteriormente.

Además la implementación de interfaces de usuario mediante lenguaje de etiquetas XML, así como la utilización del API de la plataforma y la base de datos para la misma, SQLite que para mí todo ello era completamente desconocido.

El balance obtenido del mismo es muy positivo, sin embargo, aunque se han narrado las ventajas y cualidades que ofrece la plataforma, no se quiere dejar pasar la oportunidad de exponer algunas dificultades que se han encontrado en este proyecto:

- La documentación ofrecida por Google, como se ha comentado anteriormente, ha sido la principal fuente de conocimiento sobre seguridad y controles Android. La parte que más abarca dicha documentación es la referente a la exposición de Android, donde sí que se ha echado en falta un mayor contenido de información práctica sobre la seguridad que ofrece. Es entonces cuando han entrado en juego las comunidades de Android, con sus foros, blogs y demás publicaciones.

- En lo que a la instalación del entorno de desarrollo se refiere para realizar la aplicación *AgendaPersonal* hemos tenido ciertos problemas con el emulador proporcionado por el SDK, pues a pesar de sernos de gran utilidad tiene ciertas limitaciones a la hora de realizar ciertas pruebas y la velocidad de carga del mismo podría ser algo más rápida.

10.2 Trabajos Futuros

Como líneas futuras de trabajo, se proponen los siguientes puntos:

- Ampliar la documentación del presente proyecto fin de carrera en lo que a objetivos respecta mediante las futuras versiones de la plataforma Android, en las que se incluirán mejoras de seguridad y controles en el sistema operativo, pues con el paso del tiempo y la progresión de los dispositivos bajo la plataforma irá en aumento el número de Bugs y vulnerabilidades detectadas, con sus respectivas correcciones.

- Ampliar la documentación de los principales paquetes y clases Android con ejemplos que ilustren su uso y funcionalidad. De esta manera, se podría intentar ampliar con este material la documentación oficial de Google. La idea sería hacer algo similar a la explicación de la aplicación *AgendaPersonal*, pero específica para los paquetes y clases más relevantes del sistema.

- Ampliar las funcionalidades ofrecidas por la aplicación *AgendaPersonal*; por ejemplo, permitir al usuario crear un calendario de reuniones o citas, de forma que la aplicación lance un aviso o alarma en el día y la hora programados. También sería interesante introducir conceptos de seguridad y privacidad: identificación a la hora de acceder a la información sensible de la aplicación. O por ejemplo crear una vinculación entre las distintas redes sociales ya existentes con nuestra aplicación.

- Realizar un estudio y análisis exhaustivo de los controles y seguridad de la plataforma a la hora de desarrollar aplicaciones para la misma, testeando posibles ataques a la información sensible de los dispositivos a través de dichas aplicaciones desarrolladas, testeando y documentando distintos tipos de ataques aplicados a estas últimas.

- Realizar un estudio y análisis al igual que se ha desarrollado en este proyecto, de otros sistemas operativos competidores en el mismo sector del mercado que la plataforma Android, es decir, iOS, Windows Phone, BlackBerry OS o Symbian por ejemplo.

- Realizar la comparativa entre los análisis mencionados en el punto anterior.

Capítulo 11

Presupuesto

11 Presupuesto

En este capítulo se muestra la estimación del coste real del proyecto. Para ello se ha dividido la estimación del mismo en fases, contabilizando el esfuerzo de cada una de ellas en horas.

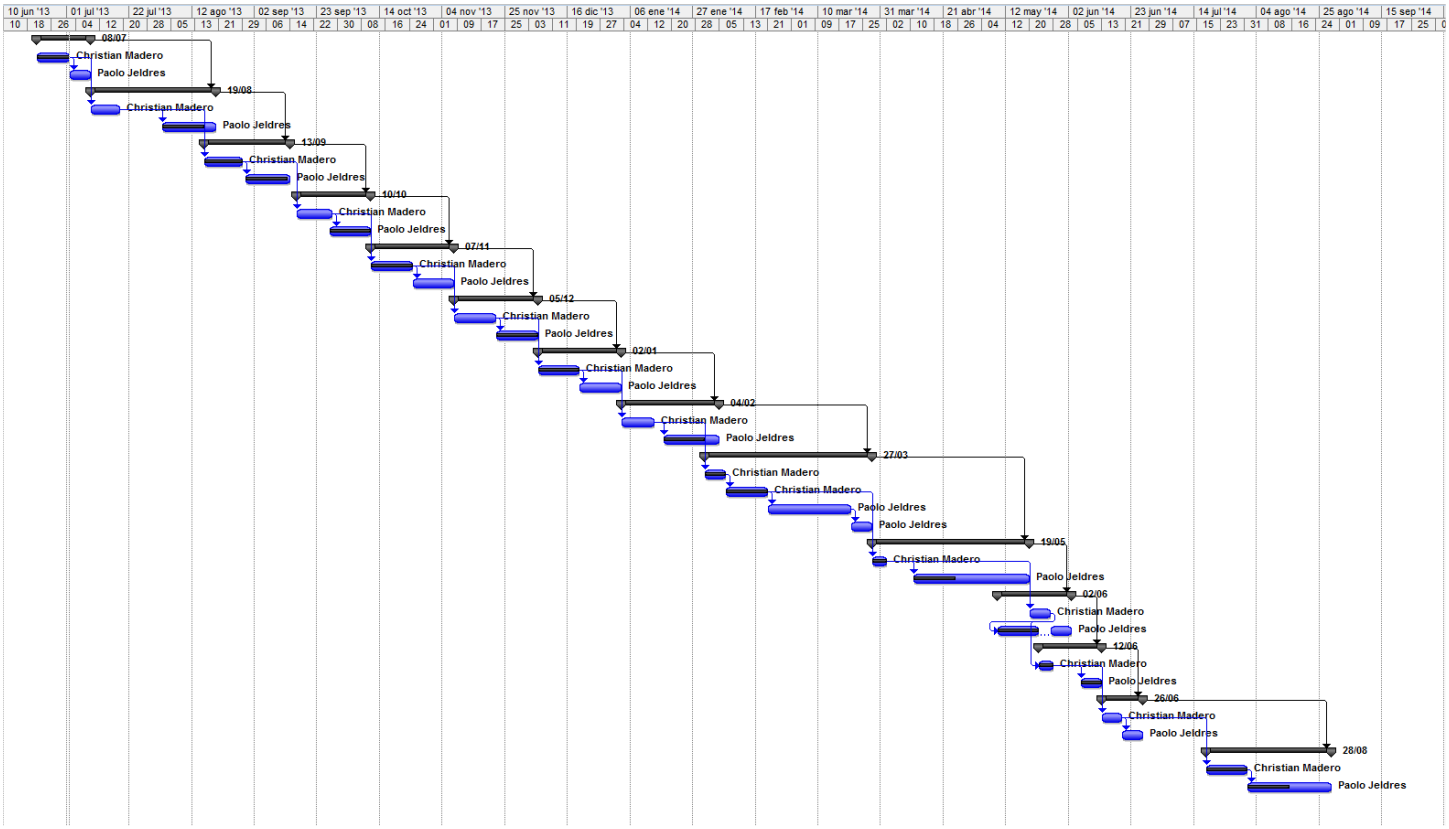
El cálculo total se ha realizado mediante la suma del coste de los diferentes recursos humanos y materiales que han intervenido a lo largo de la vida del proyecto.

En el caso de los recursos humanos, el precio de una hora de esfuerzo varía dependiendo de su perfil.

11.1 Diagrama Gantt y tablas de Coste

A continuación se muestra el diagrama Gantt perteneciente a las fases de nuestro proyecto.

	Nombre de tarea	Duración	Comienzo	Fin	Predecesoras	Nombres de los recursos
1	**Capítulo 1 Introducción**	**12 días**	**vie 21/06/13**	**lun 08/07/13**		
2	Análisis e Investigación	7 días	vie 21/06/13	lun 01/07/13		Christian Madero
3	Documentar capítulo	5 días	mar 02/07/13	lun 08/07/13	2	Paolo Jeldres
4	**Capítulo 2 Seguridad In**	**30 días**	**mar 09/07/13**	**lun 19/08/13**	**1**	
5	Análisis e Investigación	8 días	mar 09/07/13	jue 18/07/13	2	Christian Madero
6	Documentar capítulo	12 días	vie 02/08/13	lun 19/08/13	5	Paolo Jeldres
7	**Capítulo 3 Introducción**	**21 días**	**vie 16/08/13**	**vie 13/09/13**	**4**	
8	Análisis e Investigación	9 días	vie 16/08/13	mié 28/08/13	5	Christian Madero
9	Documentar capítulo	11 días	vie 30/08/13	vie 13/09/13	8	Paolo Jeldres
10	**Capítulo 4 Android**	**19 días**	**lun 16/09/13**	**jue 10/10/13**	**7**	
11	Análisis e Investigación	10 días	lun 16/09/13	vie 27/09/13	8	Christian Madero
12	Documentar capítulo	10 días	vie 27/09/13	jue 10/10/13	11	Paolo Jeldres
13	**Capítulo 5 Firma de apli**	**20 días**	**vie 11/10/13**	**jue 07/11/13**	**10**	
14	Análisis e Investigación	10 días	vie 11/10/13	jue 24/10/13	11	Christian Madero
15	Documentar capítulo	10 días	vie 25/10/13	jue 07/11/13	14	Paolo Jeldres
16	**Capítulo 6 Versionando**	**20 días**	**vie 08/11/13**	**jue 05/12/13**	**13**	
17	Análisis e Investigación	10 días	vie 08/11/13	jue 21/11/13	14	Christian Madero
18	Documentar capítulo	10 días	vie 22/11/13	jue 05/12/13	17	Paolo Jeldres
19	**Capítulo 7 Publicación d**	**20 días**	**vie 06/12/13**	**jue 02/01/14**	**16**	
20	Análisis e Investigación	10 días	vie 06/12/13	jue 19/12/13	17	Christian Madero
21	Documentar capítulo	10 días	vie 20/12/13	jue 02/01/14	20	Paolo Jeldres
22	**Capítulo 8 Ataques sob**	**23 días**	**vie 03/01/14**	**mar 04/02/14**	**19**	
23	Análisis e Investigación	7 días	vie 03/01/14	lun 13/01/14	20	Christian Madero
24	Documentar capítulo	13 días	vie 17/01/14	mar 04/02/14	23	Paolo Jeldres
25	**Creación aplicación Age**	**40 días**	**vie 31/01/14**	**jue 27/03/14**	**22**	
26	Análisis y Viabilidad	5 días	vie 31/01/14	jue 06/02/14	23	Christian Madero
27	Diseño	10 días	vie 07/02/14	jue 20/02/14	26	Christian Madero
28	Implementación	20 días	vie 21/02/14	jue 20/03/14	27	Paolo Jeldres
29	Pruebas	5 días	vie 21/03/14	jue 27/03/14	28	Paolo Jeldres
30	**Capítulo 9 La aplicación**	**37 días**	**vie 28/03/14**	**lun 19/05/14**	**25**	
31	Análisis e Investigación	3 días	vie 28/03/14	mar 01/04/14	27	Christian Madero
32	Documentar capítulo	27 días	vie 11/04/14	lun 19/05/14	31	Paolo Jeldres
33	**Anexo A: Normalización**	**17 días**	**vie 09/05/14**	**lun 02/06/14**	**30**	
34	Análisis e Investigación	5 días	mar 20/05/14	lun 26/05/14	31	Christian Madero
35	Documentar anexo	15 días	vie 09/05/14	lun 02/06/14	34	Paolo Jeldres
36	**Anexo B: Instalación en**	**15 días**	**vie 23/05/14**	**jue 12/06/14**	**33**	
37	Análisis e Investigación	3 días	vie 23/05/14	mar 27/05/14	34	Christian Madero
38	Documentar anexo	5 días	vie 06/06/14	jue 12/06/14	37	Paolo Jeldres
39	**Capítulo 11 Presupuest**	**10 días**	**vie 13/06/14**	**jue 26/06/14**	**36**	
40	Análisis e Investigación	5 días	vie 13/06/14	jue 19/06/14	37	Christian Madero
41	Documentar capítulo	5 días	vie 20/06/14	jue 26/06/14	40	Paolo Jeldres
42	**Capítulo 10 Conclusion**	**30 días**	**vie 18/07/14**	**jue 28/08/14**	**39**	
43	Análisis e Investigación	10 días	vie 18/07/14	jue 31/07/14	40	Christian Madero
44	Documentar capítulo	20 días	vie 01/08/14	jue 28/08/14	43	Paolo Jeldres



A continuación se muestra el desglose de los costes por fases del proyecto:

FASE	PRECIO RECURSO PERSONAL (€/hora)	DURACIÓN (horas)	COSTE TOTAL (€)
Capítulo 1 Introducción	16,125	87	1.402,875
Capítulo 2 Seguridad Informática	16,125	143	2.305,875
Capítulo 3 Introducción a la Auditoría Informática	16,125	135	2.176,875
Capítulo 4 Android	16,125	127	2.047,875
Capítulo 5 Firma de aplicaciones	16,125	127	2.047,875
Capítulo 6	16,125	127	2.047,875

Versionando nuestras aplicaciones			
Capítulo 7 Publicación de nuestras aplicaciones	16,125	127	2.047,875
Capítulo 8 Ataques sobre la plataforma	16,125	151	2.434,875
Creación aplicación Agenda Personal	25,5	287	6.706,5
Capítulo 9 La aplicación Agenda Personal	16,125	263	4.240,875
Anexo A: Normalización y Estándares	16,125	167	2.692,875
Anexo B: Instalación entorno de desarrollo	16,125	87	1.402,875
Capítulo 11 Presupuesto	16,125	87	1.402,875
Capítulo 10 Conclusiones y trabajos futuros	16,125	204	3.337,875
TOTAL		**2.119**	**36.825,47**

El desglose de los costes materiales es:

RECURSO MATERIAL	Nº UNIDADES	PRECIO UNITARIO (€)	COSTE TOTAL (Parte vida proyecto €)
PC (torre, monitor, teclado, etc.)	1	500	134,58
PC Portátil	1	750	201,88
Software (Windows 8 Pro, Paquete Office Pro, Eclipse, SDK Android)	2	820	820
TOTAL			**1.156,46**

Por último, sumando los costes de personal y los costes de recursos materiales utilizados y aplicando una tasa del 20% en costes indirectos, el presupuesto total de este proyecto asciende a la cantidad de **45.578** EUROS.

CONCEPTO	COSTE (€)
Proyecto completo	**36.825,47**
Recursos materiales	**1.156,46**
Costes Indirectos 20%	**7.596**
TOTAL	**45.578**

Leganés a X de Agosto de 2013

El ingeniero proyectista

Fdo. Christian Madero García.

Adjuntamos a continuación el Excel que nos proporciona el departamento de Informática de la Universidad a modo de plantilla.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor:

Christian Madero García

2.- Departamento:

Departamento de Informática

3.- Descripción del Proyecto:

- Titulo: **Controles y seguridad bajo entorno Android**
- Duración (meses): **16,15**
- Tasa de costes Indirectos: **20%**

4.- Presupuesto total del Proyecto (valores en Euros):

Euros: 44594

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a título informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
Christian Madero		Ingeniero Senior	8,7	2.460,94	21.410,16	
Paolo Jeldres		Ingeniero	8,7	1.771,88	15.415,31	
					0,00	
					0,00	
					0,00	
		Hombres mes	**17,4**	**Total**	36.825,47	

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d)]
PC (torre, monitor, teclado, etc.)	500,00	100	16	60	134,58
PC Portátil	750,00	100	16	60	201,88
					0,00
					0,00
					0,00
					0,00
				Total	336,46

[d)] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} \text{xCxD}$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e)]

Descripción	Empresa	Costes imputable
Software (Windows 8 Pro, Paquete Office Pro, Eclipse, SDK Android)		820,00
	Total	820,00

[e)] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,…

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	36.825
Amortización	336
Subcontratación de tareas	0
Costes de funcionamiento	820
Costes Indirectos	7.596
Total	45.578

Glosario

AIDL	*Android Interface Definition Language*
API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
ASP	*Application Service Provider*
CPU	*Central Processing Unit*
E/S	*Entrada/Salida*
GPS	*Global Positioning System*
GPRS	*General Packet Radio Service*
GUI	*Graphical User Interface*
HTTP	*HyperText Transfer Protocol*
JAD	*Java Aplication Descriptor*
JAR	*Java ARchive*
Java ME	*Java Micro Edition*
JSR	*Java Specification Request*
JVM	*Java Virtual Machine*
MMS	*Multimedia Messaging System*
OHA	*Open Handset Alliance*
PDA	*Personal Digital Assistant*
RAM	*Random Access Memory*
SDK	*Software Development Kit*
SMS	*Short Message Service*
XML	*Extensible Markup Language*

Referencias

[1] Seguridad informática, Wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Seguridad_inform%C3%A1tica>. Último acceso Septiembre de 2011.

[2] Administración de sistemas operativos, las diez leyes inmutables de la seguridad. Disponible [Internet]: <http://www.adminso.es/index.php/1._Las_diez_leyes_inmutables_de_la_seguridad>. Último acceso Octubre de 2011.

[3] www.thc.org

[4] Amenazas deliberadas a la seguridad de la información. Disponible [Internet]: <http://www.iec.csic.es/criptonomicon/seguridad/amenazas.html>. Último acceso Diciembre de 2011.

[5] Auditoría Informática. Control Interno. Disponible [Internet]: <https://sites.google.com/site/navaintegdesign/temario/1-4-control-interno>. Último acceso Enero de 2012.

[6] Apuntes de clase

[7] Android. Wikipedia. Disponible [Internet]: <http://es.wikipedia.org/wiki/Android>. Último acceso Marzo de 2012.

[8] Android Developers. Disponible [Internet]: <http://developer.android.com/develop/index.html>. Último acceso Abril de 2012.

[9] Open source project. Android Technical Information. Disponible [Internet]: <https://source.android.com/tech/index.html>. Último acceso Mayo de 2012.

[10] Open source project. Android Security Overview. Disponible [Internet]: <https://source.android.com/tech/security/index.html>. Último acceso Junio de 2012.

[11] Android developers. Security Tips. Disponible [Internet]: <http://developer.android.com/guide/topics/security/security.html>. Último acceso Julio de 2012.

[12] Android developers. API Guides. Permissions. Disponible [Internet]: <http://developer.android.com/guide/topics/security/permissions.html>. Último acceso Agosto de 2012.

[13] Android developers. Tools. Signing your Applications. Disponible [Internet]: <http://developer.android.com/tools/publishing/app-signing.html>. Último acceso Septiembre de 2012.

[14] Android developers. Tools. Versioning your Applications. Disponible [Internet]: <http://developer.android.com/tools/publishing/versioning.html>. Último acceso Octubre de 2012.

[15] Android developers. Tools. Preparing for Release. Disponible [Internet]: <http://developer.android.com/tools/publishing/preparing.html>. Último acceso Noviembre de 2012.

[16] A finales de año habrá un millón de amenazas para Android. Disponible [Internet]: <http://www.itespresso.es/millon-amenazas-android-105768.html>. Último acceso Enero de 2013.

[17] Cómo saltarse el patrón de seguridad de Android. Disponible [Internet]: <http://www.elandroidelibre.com/2012/08/como-saltarse-el-patron-de-seguridad-de-android.html>. Último acceso Diembre 2012.

[18] Comprueba la seguridad de tu terminal Android. Disponible [Internet]: <http://www.cyberhades.com/2012/07/24/comprueba-la-seguridad-de-tu-terminal-android/>. Último acceso Diembre 2012.

[19] UML® Resource Page. Disponible [Internet]: <http://www.uml.org/>. Último acceso Marzo de 2013.

[20] International Organization for Standardization y por la comisión International Electro technical Commission. "ISO/IEC 27001 *Information technology - Security techniques - Information security management systems - Requirements*", (2005, 2005).

[21] International Organization for Standardization y por la comisión International Electro technical Commission. "ISO/IEC 27002 anterior ISO/IEC 17799 *Information technology - Security techniques - Code of practice for information security management*", (2000, 2005).

[22] Real Decreto 1720/2007, de 21 de diciembre, por el que se aprueba el Reglamento de Desarrollo de la Ley Orgánica 15/1999, de 13 de diciembre, de Protección de Datos de Carácter Personal (RLOPD).

[23] Android developers. Tools. Setting Up an Existing IDE. Disponible [Internet]: <http://developer.android.com/sdk/installing/index.html>. Último acceso Junio de 2013.

[24] Eclipse. Project Galileo. Disponible [Internet]: <http://www.eclipse.org/galileo/>. Último acceso Junio de 2013.

[25] Sayed Y. Hashimi, Satya Komatineni, and Dave MacLean: "*Pro Android 2*" (Apress, 2010).

[26] Reto Meier: "*Professional Android™ 2 Application Development*" (Wiley Publishing, Inc., 2010).

[27] Se descubre el primer ataque "drive-by" para Android. Disponible [Internet]: <http://www.viruslist.com/sp/news?id=208274967>. Último acceso Diembre 2012.

[28] Nuevo keylogger para Android basado en el movimiento del teléfono. Disponible [Internet]: <http://unaaldia.hispasec.com/2012/04/nuevo-keylogger-para-android-basado-en.html>. Último acceso Diembre 2012.

[29] Denegación de servicio en todas las versiones de Android. Disponible [Internet]: <http://unaaldia.hispasec.com/2012/04/denegacion-de-servicio-en-todas-las.html>. Último acceso Diembre 2012.

[30] En 2011 se detectó un incremento del 21% en los ataques a dispositivos móviles. Disponible [Internet]: <http://blog.s21sec.com/2012/03/en-2011-se-detecto-un-incremento-del-21.html>. Último acceso Diembre 2012.

[31] Boletín diario de Seguridad de INTECO-CERT. Día 20/01/2012. Disponible [Internet].

[32] Informe sobre seguridad de dispositivos móviles. Disponible [Internet]: <http://blog.segu-info.com.ar/2011/11/informe-sobre-seguridad-de-dispositivos.html?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed%3A+NoticiasSeguridadInformatica+%28Noticias+de+Seguridad+Inform%C3%A1tica%29&utm_content=Google+Reader#axzz1e2aaolsQ>. Último acceso Diembre 2012.

Anexo A: Normalización y Estándares

1 ISO 27001

ISO (la Organización Internacional para la Estandarización).

Para la realización de este anexo A, de nuestro proyecto fin de carrera, hemos recopilado y estudiado información de las siguientes referencias [20], [21] y [22].

1.1 Introducción

- **General**

Este Estándar Internacional ha sido preparado para proporcionar un modelo para establecer, implementar, operar, monitorear, revisar, mantener y mejorar un Sistema de Gestión de Seguridad de la Información (SGSI). La adopción de un SGSI debe ser una decisión estratégica para una organización. El diseño e implementación del SGSI de una organización es influenciado por las necesidades y objetivos, requisitos de seguridad, los procesos empleados y el tamaño y estructura de la organización. Se espera que estos y sus sistemas de apoyo cambien a lo largo del tiempo. Se espera que la implementación de un SGSI se extienda en concordancia con las necesidades de la organización; por ejemplo, una situación simple requiere una solución SGSI simple.

Para nuestro proyecto aplicaremos los principios de este estándar sobre los SGSI a la plataforma desarrollada por Google. Para ello nombraremos y definiremos los puntos que consideremos importantes de dicho estándar y aplicable al sistema operativo Android.

- **Enfoque del proceso**

Este Estándar Internacional promueve la adopción de un enfoque del proceso para establecer, implementar, operar, monitorear, revisar, mantener y mejorar el SGSI de una organización.

Una organización necesita identificar y manejar muchas actividades para poder funcionar de manera efectiva. Cualquier actividad que usa recursos y es manejada para permitir la transformación de los factores de producción en outputs, se puede considerar un proceso. Con frecuencia el output de un proceso forma directamente el factor de producción del siguiente proceso.

La aplicación de un sistema de procesos dentro de una organización, junto con la identificación y las interacciones de estos procesos, y su gestión, puede considerarse un "enfoque del proceso".

Un enfoque del proceso para la gestión de la seguridad de la información presentado en este Estándar Internacional fomenta que sus usuarios enfaticen la importancia de:

a) Entender los requisitos de seguridad de la información de una organización y la necesidad de establecer una política y objetivos para la seguridad de la información.

b) Implementar y operar controles para manejar los riesgos de la seguridad de la información.

c) Monitorear y revisar el desempeño y la efectividad del SGSI.

d) Mejoramiento continúo en base a la medición del objetivo.

Este Estándar Internacional adopta el modelo del proceso Planear-Hacer-Verificar-Actuar (PDCA), el cual se puede aplicar a todos los procesos SGSI. La Figura 42 muestra cómo un SGSI toma como factor de producción los requisitos y expectativas de la seguridad de la información de las partes interesadas y a través de las acciones y procesos necesarios produce resultados de seguridad de la información que satisfacen aquellos requisitos y expectativas.

Este Estándar Internacional proporciona un modelo sólido para implementar los principios en aquellos lineamientos que gobiernan la evaluación del riesgo, diseño e implementación de seguridad, gestión y re-evaluación de la seguridad.

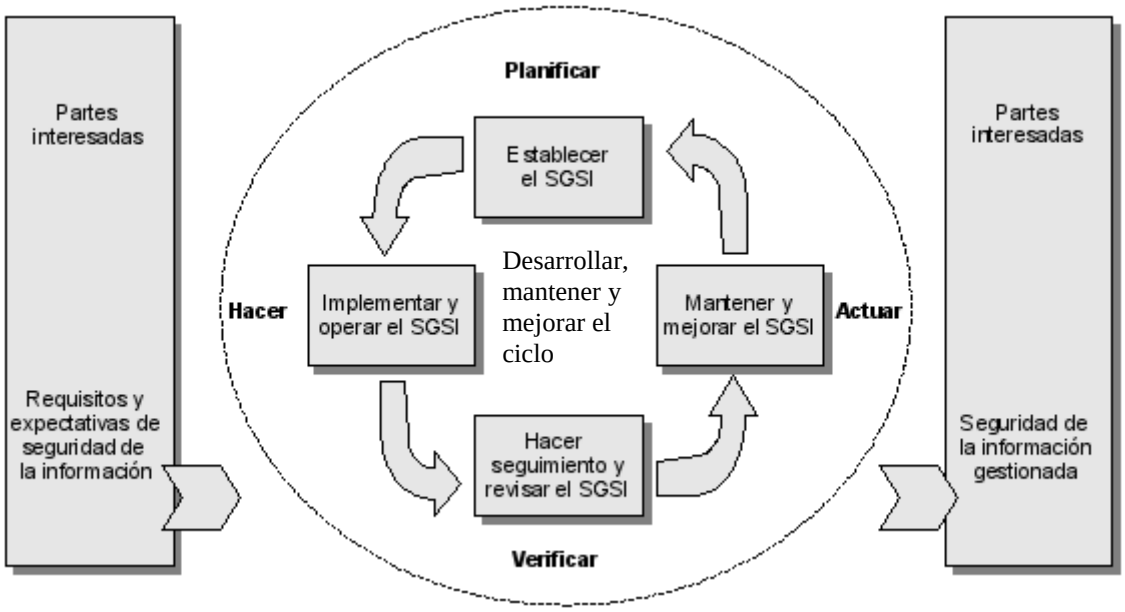


Figura 42. Modelo PDCA aplicado a los procesos SGSI. [REF]

Planear (establecer el SGSI)	Establecer política, objetivos, procesos y procedimientos SGSI relevantes para manejar el riesgo y mejorar la seguridad de la información para entregar resultados en concordancia con las políticas y objetivos generales de la organización.
Hacer (implementar y operar el SGSI)	Implementar y operar la política, controles, procesos y procedimientos SGSI.
Validar (monitorear y revisar el SGSI)	Evaluar y, donde sea aplicable, medir el desempeño del proceso en comparación con la política, objetivos y experiencias prácticas SGSI y reportar los resultados a la gerencia para su revisión.
Actuar (mantener y mejorar el SGSI)	Tomar acciones correctivas y preventivas, basadas en los resultados de la auditoría interna SGSI y la revisión gerencial u otra información relevante, para lograr el mejoramiento continuo del SGSI.

Tabla 5. Fases del Modelo PDCA aplicado a los procesos SGSI. [21]

2 ISO 27002

ISO (la Organización Internacional de Estandarización) e IEC (la Comisión Electrotécnica Internacional) forman el sistema especializado para la estandarización mundial. Los organismos internacionales miembros de ISO e IEC participan en el desarrollo de Estándares Internacionales a través de los comités establecidos por la organización respectiva para lidiar con áreas particulares de la actividad técnica.

2.1 Introducción

2.1.1 ¿Qué es seguridad de la información?

La información es un activo que, como otros activos comerciales importantes, es esencial para el negocio de una organización y en consecuencia necesita ser protegido adecuadamente.

Esto es especialmente importante en el ambiente comercial cada vez más interconectado.

Como resultado de esta creciente interconectividad, la información ahora está expuesta a un número cada vez mayor y una variedad más amplia de amenazas y vulnerabilidades.

La información puede existir en muchas formas. Puede estar impresa o escrita en un papel, almacenada electrónicamente, transmitida por correo o utilizando medios electrónicos, mostrada en películas o hablada en una conversación. Cualquiera que sea la forma que tome la información, o medio por el cual sea almacenada o compartida, siempre debiera estar apropiadamente protegida.

La seguridad de la información es la protección de la información de un rango amplio de amenazas para poder asegurar la continuidad del negocio, minimizar el riesgo comercial y maximizar el retorno de las inversiones y las oportunidades comerciales.

La seguridad de la información se logra implementando un adecuado conjunto de controles; incluyendo políticas, procesos, procedimientos, estructuras organizacionales y funciones de software y hardware. Se necesitan establecer, implementar, monitorear, revisar y mejorar estos controles cuando sea necesario para asegurar que se cumplan los objetivos de seguridad y comerciales específicos. Esto se debiera realizar en conjunción con otros procesos de gestión del negocio.

2.1.2 ¿Por qué se necesita seguridad de la información?

La información y los procesos, sistemas y redes de apoyo son activos comerciales importantes. Definir, lograr, mantener y mejorar la seguridad de la información puede ser esencial para mantener una ventaja competitiva, el flujo de caja, rentabilidad, observancia legal e imagen comercial.

Las organizaciones y sus sistemas y redes de información enfrentan amenazas de seguridad de un amplio rango de fuentes; incluyendo fraude por computadora, espionaje, sabotaje, vandalismo, fuego o inundación. Las causas de daño como código malicioso, pirateo computarizado o negación de ataques de servicio se hacen cada vez más comunes, más ambiciosas y cada vez más sofisticadas.

La seguridad de la información es importante tanto para negocios del sector público como privado, y para proteger las infraestructuras críticas. En ambos sectores, la seguridad de la información funcionará como un facilitador; por ejemplo para lograr e-gobierno o e-negocio, para evitar o reducir los riesgos relevantes. La interconexión de redes públicas y privadas y el intercambio de fuentes de información incrementan la dificultad de lograr un control del acceso. La tendencia a la computación distribuida también ha debilitado la efectividad de un control central y especializado.

Muchos sistemas de información no han sido diseñados para ser seguros. La seguridad que se puede lograr a través de medios técnicos es limitada, y debiera ser apoyada por la gestión y los procedimientos adecuados. Identificar qué controles establecer requiere de una planeación cuidadosa y prestar atención a los detalles. La gestión de la seguridad de la información requiere, como mínimo, la participación de los accionistas, proveedores, terceros, clientes u otros grupos externos. También se puede requerir asesoría especializada de organizaciones externas.

2.1.3 ¿Cómo establecer los requisitos de seguridad?

Es esencial que una organización identifique sus requisitos de seguridad. Existen tres fuentes principales de requisitos de seguridad.

Una fuente se deriva de evaluar los riesgos para la organización, tomando en cuenta la estrategia general y los objetivos de la organización. A través de la evaluación del riesgo, se identifican las amenazas para los activos, se evalúa la vulnerabilidad y la probabilidad de ocurrencia y se calcula el impacto potencial.

Otra fuente son los requisitos legales, reguladores, estatutarios y contractuales que tienen que satisfacer una organización, sus socios comerciales, contratistas y proveedores de servicio; y su ambiente socio-cultural.

Otra fuente es el conjunto particular de principios, objetivos y requisitos comerciales para el procesamiento de la información que una organización ha desarrollado para sostener sus operaciones.

2.1.4 Evaluando los riesgos de la seguridad

Los requisitos de seguridad se identifican mediante una evaluación metódica de los riesgos de seguridad. El gasto en controles debiera ser equilibrado con el daño comercial probable resultado de fallas en la seguridad.

Los resultados de la evaluación del riesgo ayudarán a guiar y determinar la acción de gestión apropiada y las prioridades para manejar los riesgos de seguridad de la información, e implementar los controles seleccionados para protegerse contra esos riesgos.

La evaluación del riesgo se debiera repetir periódicamente para tratar cualquier cambio que podría influir en los resultados de la evaluación del riesgo.

2.1.5 Selección de controles

Una vez que se han identificado los requisitos y los riesgos de seguridad y se han tomado las decisiones para el tratamiento de los riesgos, se debieran seleccionar los controles apropiados y se debieran implementar para asegurar que los riesgos se reduzcan a un nivel aceptable. Los controles se pueden seleccionar a partir de este estándar o de otros conjuntos de controles, o se pueden diseñar controles nuevos para cumplir con necesidades específicas conforme sea apropiado. La selección de los controles de seguridad depende de las decisiones organizacionales basadas en el criterio de aceptación del riesgo, opciones de tratamiento del riesgo y el enfoque general para la gestión del riesgo aplicado a la organización, y también debieran estar sujetas a todas las regulaciones y legislación nacionales e internacionales relevantes.

Algunos de los controles en este estándar se pueden considerar principios guías para la gestión de la seguridad de la información y aplicables a la mayoría de las organizaciones.

Tecnología de la información – Técnicas de seguridad – Código de práctica para la gestión de la seguridad de la información

2.2 Alcance del Estandar

Este Estándar Internacional establece los lineamientos y principios generales para iniciar, implementar, mantener y mejorar la gestión de la seguridad de la información en una organización. Los objetivos delineados en este Estándar Internacional proporcionan un lineamiento general sobre los objetivos de gestión de seguridad de la información generalmente aceptados.

Los objetivos de control y los controles de este Estándar Internacional son diseñados para ser implementados para satisfacer los requisitos identificados por una evaluación del riesgo.

Este Estándar Internacional puede servir como un lineamiento práctico para desarrollar estándares de seguridad organizacional y prácticas de gestión de seguridad efectivas y para ayudar a elaborar la confianza en las actividades inter-organizacionales.

En nuestro proyecto vamos a seguir este estándar internacional adaptándolo a la plataforma Android en diferentes dispositivos en vez de que únicamente de organizaciones se tratase, vamos a adaptarlo para el sistema operativo y las interrelaciones entre varios dispositivos bajo la misma plataforma, las aplicaciones diseñadas para el mismo y las organizaciones propietarias de ambos.

2.3 Estructura de este estándar

Este estándar contiene 11 cláusulas de control de seguridad conteniendo colectivamente un total de 39 categorías de seguridad principales y una cláusula introductoria que presenta la evaluación y tratamiento del riesgo. *En este proyecto, solamente comentaremos los puntos interesantes para la implementación en la plataforma Android, a fin de proporcionar una guía para la mejora de la seguridad de dicho sistema y las aplicaciones relacionadas.*

2.3.1 Cláusulas

Cada cláusula contiene un número de categorías de seguridad principales. Las once cláusulas (acompañadas por el número de categorías de seguridad principales incluidas dentro de cada cláusula) son:

a) Política de Seguridad.

b) Organización de la Seguridad de la Información.

c) Gestión de Activos.

d) Seguridad de Recursos Humanos.

e) Seguridad Física y Ambiental.

f) Gestión de Comunicaciones y Operaciones.

g) Control de Acceso.

h) Adquisición, Desarrollo y Mantenimiento de Sistemas de Información.

i) Gestión de Incidentes de Seguridad de la Información.

j) Gestión de la Continuidad del Negocio.

k) Conformidad.

El orden de las cláusulas en este estándar no implica su importancia. Dependiendo de las circunstancias, todas las cláusulas pueden ser importantes, por lo tanto, cada organización que aplica este estándar debiera identificar las cláusulas aplicables, cuan importante son y su aplicación a los procesos comerciales individuales. Por otra parte, las listas en este estándar no están por orden de prioridad a no ser que se así se especifique.

Algunas de estas cláusulas no van a ser de interés para el desarrollo de este proyecto, solamente comentaremos aquellas que sí nos pueden ayudar a la mejora de la seguridad de la plataforma.

2.4 Gestión de activos

2.4.1 Clasificación de la información

El objetivo es asegurar que la información reciba un nivel de protección apropiado.

La información debiera ser clasificada para indicar la necesidad, prioridades y grado de protección esperado cuando se maneja la información.

La información tiene diversos grados de confidencialidad e importancia. Algunos elementos pueden requerir un nivel de protección adicional o manejo especial. Se debiera utilizar un esquema de clasificación de información para definir un conjunto apropiado de niveles de protección y comunicar la necesidad de medidas de uso especiales.

Por ejemplo en el caso de dispositivos móviles bajo la plataforma Android cabría el uso de una jerarquía para la sensibilidad de la información almacenada en el dispositivo y que el usuario pudiese manejar dichos niveles de privacidad con el uso de contraseñas o claves criptográficas, como así mismo las aplicaciones desarrolladas para dicha plataforma y el manejo de información.

2.4.1.1 Lineamientos de clasificación

Se debiera clasificar la información en términos de su valor, requisitos legales, sensibilidad y grado crítico para el usuario.

Se puede evaluar el nivel de protección analizando la confidencialidad, integridad y disponibilidad, y cualquier otro requisito para la información considerada.

Con frecuencia, la información deja de ser sensible o crítica después de cierto período de tiempo, por ejemplo, cuando la información se ha hecho pública. Se debieran tomar en cuenta estos aspectos, ya que la sobre-clasificación puede llevar a la implementación de controles innecesarios resultando en un gasto adicional.

Agrupar documentos con requisitos de seguridad similares cuando se asignan niveles de clasificación podría ayudar a simplificar la tarea de clasificación.

En general, la clasificación dada a la información es una manera rápida para determinar cómo se está manejando y protegiendo la información.

Como hemos mencionado anteriormente sería importante la clasificación de la información con una jerarquía de privacidad en el dispositivo, podemos hablar de la clasificación de datos personales según niveles (artículo 81 del Reglamento de Desarrollo de la LOPD).

2.5 Gestión de las comunicaciones y operaciones

2.5.1 Gestión de la entrega del servicio de terceros

El objetivo es implementar y mantener el nivel apropiado de seguridad de la información y la entrega del servicio en línea con los acuerdos de entrega de servicios de terceros.

El sistema debiera validar la implementación de los acuerdos, monitorear su cumplimiento con los estándares y manejar los cambios para asegurar que los servicios sean entregados para satisfacer todos los requisitos acordados por terceros.

Se entiende por terceros en este caso a las aplicaciones diseñadas para su ejecución en la plataforma Android y la interacción de las mismas con la información sensible almacenada en el sistema.

2.5.1.1 Entrega del servicio

Se debiera asegurar que los controles de seguridad, definiciones del servicio y niveles de entrega incluidos en el acuerdo de entrega del servicio de terceros se implementen, operen y mantengan.

La entrega del servicio por un tercero debiera incluir los acuerdos de seguridad pactados, definiciones del servicio y aspectos de la gestión del servicio. En caso de los acuerdos de abastecimiento externo, el sistema debiera planear las transiciones necesarias (de información, medios de procesamiento de la información y cualquier otra cosa que necesite transferirse), y debiera asegurar que se mantenga la seguridad a través del período de transición.

El sistema debiera asegurar que la aplicación mantenga una capacidad de servicio suficiente junto con los planes de trabajo diseñados para asegurar que se mantengan los niveles de continuidad del servicio después de fallos importantes en el servicio o un desastre en el mismo.

Este punto debiera afectar al intercambio de información con aplicaciones diseñadas para la plataforma por ejemplo, ya que la mayoría de aplicaciones disponibles son abastecidas por terceros.

2.5.1.2 Monitoreo y revisión de los servicios de terceros

Los servicios, reportes y registros provistos por terceros debieran ser monitoreados y revisados regularmente, y se debieran llevar a cabo auditorías regularmente.

El monitoreo y revisión de los servicios de terceros debiera asegurar que se cumplan los términos y condiciones de seguridad de los acuerdos, y que se manejen apropiadamente los incidentes y problemas de seguridad de la información. Esto debiera involucrar una relación y proceso de gestión de servicio entre la organización propietaria del sistema y terceras partes propietarias de las aplicaciones para:

a) Monitorear los niveles de desempeño del servicio para validar adherencia con los acuerdos.

b) Revisar de los reportes de servicio producidos por terceros y acordar reuniones de avance regulares conforme lo requieran los acuerdos.

c) Proporcionar información sobre incidentes de seguridad de la información y la revisión de esta información por terceros y la organización conforme lo requieran los acuerdos y cualquier lineamiento y procedimiento de soporte.

d) Revisar los rastros de auditoría de terceros y los registros de eventos de seguridad, problemas operacionales, fallos, el monitoreo de fallos e interrupciones relacionadas con el servicio entregado.

e) Resolver y manejar cualquier problema identificado.

La responsabilidad de manejar la relación con terceros se debiera asignar a una persona o equipo de gestión de servicios de la organización responsable de la plataforma. Además, dicha organización debiera asegurar que los terceros asignen responsabilidad para la validación del cumplimiento de los requisitos de los acuerdos. Se debieran poner a disposición las capacidades y recursos técnicos para monitorear los requisitos del acuerdo, en particular si se cumplen los requisitos de seguridad de la información. Se debiera tomar la acción apropiada cuando se observan deficiencias en la entrega del servicio.

La organización responsable del sistema debiera mantener el control y la visibilidad general suficiente en todos los aspectos de seguridad con relación a la información confidencial o crítica o los medios de procesamiento de la información que la tercera persona ingresa, procesa o maneja. La organización debiera asegurarse de mantener visibilidad en las actividades de seguridad como la gestión del cambio, identificación de vulnerabilidades y reporte/respuesta de un incidente de seguridad a través de un proceso, formato y estructura de reporte definidos.

En el caso de abastecimiento externo, la organización necesita estar al tanto que la responsabilidad final de la información procesada por un proveedor externo se mantenga en la organización, sobre todo en lo que al apartado de Google Play se refiere.

Para ello sería interesante el desarrollo de un "log" con la transferencia de información y servicios entre aplicaciones, y entre aplicaciones y la plataforma, para el posterior análisis y/o detección de errores y fallos de seguridad por parte de las organizaciones propietarias, cabe destacar el desarrollo de un departamento por parte de la organización propietaria de la plataforma para la revisión de las aplicaciones desarrolladas para la misma. Este punto es de máximo protagonismo para el caso que nos concierne puesto que el servicio Google Play de Google tiene un papel principal en lo que a la plataforma Android respecta.

2.5.1.3 Manejo de cambios en los servicios de terceros

Se debieran manejar los cambios en la provisión de servicios, incluyendo el mantenimiento y mejoramiento de las políticas, procedimientos y controles de seguridad de la información existentes teniendo en cuenta el grado crítico de los sistemas involucrados y la re-evaluación de los riesgos.

El proceso de manejar los cambios en el servicio de terceros necesita tomar en cuenta:

a) Los cambios realizados por la organización proveedora de las aplicaciones para implementar:

   - Aumentos de los servicios ofrecidos actualmente a la plataforma.

- Desarrollo de cualquier aplicación y sistema nuevo u actualización de los ya existentes.
- Modificaciones o actualizaciones de las políticas y procedimientos de la organización proveedora.
- Controles nuevos para solucionar incidentes de la seguridad de la información y para mejorar la seguridad.

b) Cambios en los servicios de terceros para implementar:

- Cambios y mejoras en las redes.
- Uso de tecnologías nuevas.
- Adopción de productos nuevos o versiones más modernas.
- Desarrollo de herramientas nuevas.

Sería interesante una revisión temporal de los servicios prestados por los proveedores de aplicaciones bajo la plataforma, por un equipo a cargo del proveedor del sistema, en el que se controlen los cambios en las aplicaciones actualizadas para Google Play.

2.5.2 Planificación y aceptación del sistema

El objetivo es minimizar el riesgo de fallos en el sistema.

Se requiere planeamiento y preparación anticipados para asegurar la disponibilidad de la capacidad y los recursos adecuados para entregar el desempeño del sistema requerido.

Se debieran realizar proyecciones de los requisitos de la capacidad futura para reducir el riesgo de sobrecarga en el sistema.

Se debieran establecer, documentar y probar los requisitos operacionales de los sistemas nuevos antes de su aceptación y uso.

Este punto es interesante para la planificación del desarrollo de futuras versiones y actualizaciones de versiones ya existentes del sistema operativo, contemplando la capacidad y calidad del mismo para su uso en distintos dispositivos y su buen funcionamiento.

2.5.2.1 Gestión de la capacidad

Se debe monitorear, afinar el uso de los recursos y realizar proyecciones de los requisitos de capacidad futura para asegurar el desempeño requerido del sistema.

Se deben identificar los requisitos de capacidad de cada actividad nueva y en proceso.

Se debieran aplicar la afinación y monitoreo del sistema para asegurar y, cuando sea necesario, mejorar la disponibilidad y eficiencia del mismo. Se debieran establecer detectives de controles para indicar los problemas en el momento debido. Las proyecciones de requisitos futuros debieran tomar en cuenta los requisitos de los negocios y sistemas nuevos (incluyendo actualizaciones del mismo) y las tendencias actuales y proyectadas en las capacidades de procesamiento de la información.

Se debiera prestar particular atención a cualquier recurso con tiempo de espera largos de abastecimiento o costos altos, por lo tanto, los desarrolladores debieran monitorear la utilización de los recursos claves del sistema. Ellos debieran identificar las tendencias de uso, particularmente en relación con las aplicaciones o las herramientas del sistema.

Cabe destacar este punto ya que la plataforma al ser de código abierto, está presente en una amplia gama de dispositivos de características muy diferentes, y proporcionaría una mejora desde el punto de vista de la calidad del sistema operativo.

2.5.2.2 Aceptación del sistema

Se debiera establecer el criterio de aceptación del sistema nuevo, actualizaciones o versiones nuevas y se debieran realizar pruebas adecuadas del sistema durante el desarrollo y antes de su aceptación para diversos dispositivos.

Los desarrolladores debieran asegurar que los requisitos y criterios de aceptación de los sistemas nuevos estén claramente definidos, aceptados, documentados y probados. Los sistemas nuevos, las actualizaciones y las versiones nuevas debieran migrar a la producción después de obtener la aceptación formal. Se debieran considerar los siguientes elementos antes de proporcionar la aceptación formal:

a) El desempeño y los requisitos de capacidad de los dispositivos.

b) Procedimientos para la recuperación tras errores y reinicio, y planes de contingencia.

c) Preparación y prueba de los procedimientos de operación rutinarios para estándares definidos.

d) El conjunto de controles de seguridad acordados y aceptados.

e) Procedimientos manuales efectivos.

f) Evidencia que se está tomando en consideración el efecto que tiene el sistema nuevo en la seguridad.

g) Capacitación para la operación o uso de los sistemas nuevos.

h) Facilidad de uso, ya que esto afecta el desempeño del usuario y evita el error humano.

Para los desarrollos nuevos importantes, la función de las operaciones y los usuarios debieran ser consultados en todas las etapas del proceso del desarrollo para asegurar la eficiencia operacional del diseño del sistema propuesto. Se debieran llevar a cabo las pruebas apropiadas para confirmar que se ha cumplido totalmente con el criterio de aceptación.

La aceptación puede incluir un proceso de certificación y acreditación formal para verificar que se hayan tratado apropiadamente los requisitos de seguridad.

Relacionamos para este punto lo mencionado en el punto anterior por la amplia cuota de mercado para la que se desarrolla la plataforma Android como medida para la calidad de la misma.

2.5.3 Protección contra el código malicioso y móvil

El objetivo es proteger la integridad del software y la integración.

Se requiere tomar precauciones para evitar y detectar la introducción de códigos maliciosos y códigos móviles no-autorizados.

El software y los medios de procesamiento de la información son vulnerables a la introducción de códigos maliciosos, como virus, virus de red, Troyanos y bombas lógicas. Los usuarios debieran estar al tanto de los peligros de los códigos maliciosos. Cuando sea apropiado, los desarrolladores debieran introducir controles para evitar, detectar y eliminar los códigos maliciosos y controlar los códigos móviles.

Debido al crecimiento en el mercado de dispositivos bajo la plataforma Android cabe destacar este punto como uno de los más importantes para dicho sistema.

2.5.3.1 Controles contra códigos maliciosos

Controles de detección, prevención y recuperación para proteger contra códigos maliciosos y se debieran implementar procedimientos para el apropiado conocimiento del usuario.

La protección contra códigos maliciosos se debiera basar en la detección de los mismos y la reparación de software, conciencia de seguridad, y los apropiados controles de acceso al sistema y gestión del cambio. Se debieran considerar los siguientes lineamientos:

a) Establecer una política formal prohibiendo el uso de software no-autorizado o revisado por la organización propietaria de la plataforma.

b) Establecer una política formal para proteger contra riesgos asociados con la obtención de archivos, ya sea a través de redes externas o cualquier otro medio, indicando las medidas de protección a tomarse.

c) Realizar revisiones regulares del software y contenido de datos del sistema que sostienen los procesos críticos.

d) La instalación y actualización regular de software para la detección o reparación de códigos maliciosos para revisar los dispositivos y medios de almacenamiento como un control preventivo o una medida rutinaria, los chequeos llevados a cabo debieran incluir:

   - Validación de cualquier archivo en medios de almacenamiento, y los archivos recibidos a través de la red para detectar códigos maliciosos antes de utilizarlo.

   - Validar los adjuntos y descargas de los correos electrónicos para detectar códigos maliciosos antes de utilizarlos, esta validación debiera llevarse a cabo en lugares diferentes, por ejemplo, servidores de correo electrónico y dispositivo.

   - Validar las páginas Web para detectar códigos maliciosos.

e) Definición, gestión, procedimientos y responsabilidades para lidiar con la protección de códigos maliciosos en el sistema, capacitación en su uso, reporte y recuperación de ataques de códigos maliciosos.

f) Preparar planes apropiados para recuperarse de ataques de códigos maliciosos, incluyendo todos los datos y respaldo (back-up) de software y procesos de recuperación.

g) Implementar procedimiento para la recolección regular de información, como suscribirse a listas de correos y/o validar sitios Web que dan información sobre códigos maliciosos nuevos.

El uso de dos o más productos de software para protegerse de códigos maliciosos a través del ambiente de procesamiento de la información de diferentes vendedores puede mejorar la efectividad de la protección contra códigos maliciosos.

Se puede instalar software para protegerse de códigos maliciosos para proporcionar actualizaciones automáticas de archivos de definición y motores de lectura para asegurarse que la protección esté actualizada. Además, este software se puede instalar en el dispositivo para que realice validaciones automáticas.

Se debiera tener cuidado de protegerse contra la introducción de códigos maliciosos durante el mantenimiento y procedimientos de emergencia, los cuales pueden evadir los controles de protección contra códigos maliciosos normales.

En este punto destacamos entre otros el sistema de permisos implementado en la plataforma para realizar los controles necesarios para la protección del sistema y el actual desarrollo de aplicaciones para el mejoramiento de la seguridad de la plataforma como mencionamos en el capítulo 7 de este mismo proyecto fin de carrera.

2.5.3.2 Controles contra códigos móviles

Donde se autorice el uso del código móvil, la configuración debiera asegurar que el código móvil autorizado opera de acuerdo con una política de seguridad claramente definida, y se debiera evitar la ejecución del código móvil no-autorizado.

Se debieran considerar las siguientes acciones para evitar que el código móvil realice acciones no-autorizadas:

a) Ejecutar el código móvil en un ambiente aislado lógicamente.

b) Bloquear cualquier uso del código móvil.

c) Bloquear lo recibido del código móvil.

d) Activar las medidas técnicas conforme estén disponibles en un sistema específico para asegurar el manejo del código móvil.

e) Control de los recursos disponibles para el acceso del código móvil.

f) Controles criptográficos para autenticar singularmente el código móvil.

El código móvil es un código de software que se transfiere de un equipo a otro y luego ejecuta automáticamente y realiza un función específica con muy poca o ninguna interacción. El código móvil está asociado con un número de servicios “middleware”.

Además de asegurar que el código móvil no contenga códigos maliciosos, el control de código móvil es esencial para evitar el uso no-autorizado o interrupción del sistema o recursos de aplicación y otros fallos en la seguridad de la información.

En este punto destacamos el ataque sufrido a la plataforma en el que los atacantes usaban de por medio cierto código "javascript" en diferentes sitios web para la obtención de información sensible de los dispositivos.

2.5.4 Respaldo o Back-Up

El objetivo es mantener la integridad y disponibilidad de la información y los medios de procesamiento de información.

Se debieran establecer los procedimientos de rutina para implementar la política de respaldo acordada y la estrategia para tomar copias de respaldo de los datos y practicar su restauración oportuna.

Se debieran hacer copias de respaldo de la información y software y se debieran probar regularmente en concordancia con la política de copias de respaldo acordada.

Se debiera proporcionar medios de respaldo adecuados para asegurar que toda la información esencial y software se pueda recuperar después de un desastre o fallo.

Se debieran considerar los siguientes elementos para el respaldo de la información:

a) Se debiera definir el nivel necesario de respaldo de la información.

b) Se debieran producir registros exactos y completos de las copias de respaldo y procedimientos documentados de la restauración.

c) La extensión (por ejemplo, respaldo completo o diferencial) y la frecuencia de los respaldos debiera reflejar los requisitos de seguridad del sistema, los requisitos de seguridad de la información involucrada.

d) Las copias de respaldo se debieran almacenar en un medio externo y no en memoria interna del dispositivo.

e) A la información de respaldo se le debiera dar el nivel de protección física y ambiental apropiado consistente con los estándares aplicados en el dispositivo.

f) Los medios de respaldo se debieran probar regularmente para asegurar que se puedan confiar en ellos para usarlos cuando sea necesaria en caso de emergencia.

g) Los procedimientos de restauración se debieran validar y probar regularmente para asegurar que sean efectivos y que pueden ser completados dentro del tiempo asignado en los procedimientos operacionales para la recuperación.

h) En situaciones cuando la confidencialidad es de importancia, las copias de respaldo debieran ser protegidas por medios de una codificación.

Los procedimientos de respaldo para el sistema debieran ser probados regularmente. Para sistemas críticos, los procedimientos de respaldo debieran abarcar toda la información, aplicaciones y datos de todo el sistema, necesarios para recuperar el sistema completo en caso de un desastre.

Se debiera determinar el período de retención para la información esencial, y también cualquier requerimiento para que las copias de archivo se mantengan permanentemente.

Los procedimientos de respaldo pueden ser automatizados para facilitar el proceso de respaldo y restauración. Estas soluciones automatizadas debieran ser probadas suficientemente antes de su implementación y también a intervalos regulares, como punto para la mejora de la seguridad y fiabilidad de la plataforma.

Este punto es especialmente importante a la hora de desarrollar un mecanismo de copias de seguridad en el almacenamiento externo del dispositivo por ejemplo, con puntos de recuperación del sistema y de la información sensible que contiene el dispositivo. Podemos hablar por ejemplo de sincronización con un PC como forma de realizar back-up tanto de mensajes, contactos, archivos multimedia como de copias de seguridad del sistema.

2.5.5 Gestión de seguridad de la red

El objetivo es asegurar la protección de la información en redes.

La gestión segura de las redes, requiere de la cuidadosa consideración del flujo de datos, implicancias legales, monitoreo y protección.

También se pueden requerir controles adicionales para proteger la información confidencial que pasa a través de redes públicas.

Este punto resalta la importancia de la información residente en el sistema operativo y que es accesible a través de su conexión a redes de información.

2.5.5.1 Controles de redes

Las redes debieran ser adecuadamente manejadas y controladas para poder proteger la información en las redes, y mantener la seguridad del sistema y aplicaciones utilizando la red, incluyendo la información en tránsito.

Se debieran implementar controles para asegurar la seguridad de la información en las redes, y proteger los servicios conectados de accesos no-autorizados.

En particular, se debieran considerar los siguientes elementos:

a) Se debieran establecer las responsabilidades y procedimientos para la gestión del dispositivo remoto.

b) Se debieran establecer controles especiales para salvaguardar la confidencialidad y la integridad de los datos que pasan a través de las redes públicas o a través de las redes inalámbricas; y proyectar el sistema y aplicaciones conectadas.

c) Se debiera aplicar registros de ingreso y monitoreo apropiados para permitir el registro de las acciones de seguridad relevantes.

Para este punto sería interesante que el sistema ofreciera una herramienta para la administración y seguridad de las redes de información, que dispusiera de controles como registros de conexiones establecidas, monitoreo de los servicios conectados a la red, control del flujo de datos, y control de los datos transmitidos tanto de las aplicaciones y servicios como del propio sistema operativo.

2.5.6 Gestión de soportes

El objetivo es evitar la modificación, eliminación o destrucción de archivos, y la interrupción de las actividades del sistema.

Los medios se debieran controlar y proteger física y lógicamente en el dispositivo.

Se debieran establecer los procedimientos apropiados para proteger los documentos o archivos, medios de almacenamiento (por ejemplo, tarjetas SD), input/output de datos y documentación del sistema de una modificación, eliminación o destrucción.

Destacamos en este punto la utilización de un sistema de permisos de usuario implementado en el sistema para la apropiada manipulación de los archivos y directorios del mismo, así como la manipulación del contenido de tarjetas de memoria o la propia memoria interna del dispositivo.

2.5.6.1 Procedimientos para el manejo de información

Se debieran establecer los procedimientos para el manejo y almacenaje de información, para proteger esta información de una divulgación no-autorizada o mal uso.

Estos procedimientos se aplican a la información en documentos, sistemas de almacenamiento (tanto internos como externos del dispositivo), redes, almacenamiento móvil, comunicaciones móviles, comunicaciones vía correo, correo de voz y voz en general, multimedia, y cualquier otro elemento confidencial.

Como ya indicamos en el punto anterior cabe destacar el uso de un sistema de permisos de usuario para el correcto procesamiento, manejo y almacenamiento de la información ya sea por los usuarios del sistema como por aplicaciones o servicios del mismo.

2.5.6.2 Seguridad de la documentación del sistema

Se debiera proteger la documentación del sistema respecto a accesos no-autorizados. La documentación del sistema se debiera almacenar de una manera segura en el dispositivo.

La lista de acceso a la documentación del sistema se debiera mantener en un nivel mínimo y autorizado por el propietario del mismo.

La documentación del sistema puede contener un rango de información confidencial, por ejemplo, una descripción de los procesos de aplicaciones, procedimientos, estructuras de datos, procesos de autorización.

Mediante la utilización de permisos de súper-usuario para la modificación, eliminación o tratamiento de los ficheros y directorios de los que se compone el sistema operativo. Entendiéndose por documentación del sistema al compuesto de ficheros y directorios que lo componen.

2.5.7 Intercambio de información

El objetivo es mantener la seguridad en el intercambio de información dentro del sistema, software y con cualquier otra entidad externa.

Los intercambios de información dentro del sistema se debieran basar en una política formal de intercambio, seguida en línea con los acuerdos de intercambio, y debiera cumplir con cualquier legislación relevante.

Se debieran establecer los procedimientos y estándares para proteger la información y los medios físicos que contiene la información en-tránsito.

En este punto destacamos el intercambio de información sensible entre sistemas, aplicaciones y dispositivos externos al origen, mediante el etiquetado de la misma con distintos niveles de seguridad.

2.5.7.1 Políticas y procedimientos de intercambio de información

Se debieran establecer políticas, procedimientos y controles de intercambio formales para proteger el intercambio de información a través del uso de todos los tipos de medios de comunicación.

Los procedimientos y controles a seguirse cuando se utilizan medios de comunicación electrónicos para el intercambio de información debieran considerar los siguientes elementos:

a) Los procedimientos diseñados para proteger el intercambio de información de la interceptación, copiado, modificación, enrutamiento equivocado y destrucción.

b) Los procedimientos para la detección y protección de contra códigos maliciosos que pueden ser transmitidos a través del uso de comunicaciones electrónicas.

c) Los procedimientos para proteger la información electrónica confidencial comunicada que está en la forma de un adjunto.

d) Política o lineamientos delineando el uso aceptable de los medios de comunicación electrónicos.

e) Los procedimientos para el uso de comunicación inalámbrica, tomando en cuenta los riesgos particulares involucrados.

f) Uso de técnicas de codificación, por ejemplo, para proteger la confidencialidad, integridad y autenticidad de la información.

g) Los controles y restricciones asociados con el reenvío de los medios de comunicación, por ejemplo, reenvío automático de correo electrónico a direcciones externas.

h) Recordar al usuario que debiera tomar las precauciones apropiadas, por ejemplo, no revelar información confidencial cuando realiza una llamada telefónica para evitar ser escuchado o interceptado por:

    - Personas alrededor suyo, particularmente cuando se utilizan teléfonos móviles.

- Intervención de teléfonos y otras formas de escucha no-autorizada a través del acceso físico al teléfono o la línea telefónica, o el uso de escáneres receptores.

- Personas en el otro lado de la línea, en el lado del receptor.

i) Recordar al usuario no registrar datos demográficos, como la dirección de correo electrónico u otra información personal, en ningún software para evitar que sea utilizada sin autorización.

Los medios de intercambio de información debieran cumplir con cualquier requerimiento legal relevante.

Los intercambios de información pueden ocurrir a través del uso de un número de comunicaciones diferentes, incluyendo correo electrónico, de voz y vídeo.

El intercambio de software puede ocurrir a través de un número de medios diferentes, incluyendo la descarga de Internet.

Se debieran considerar las implicancias comerciales, legales y de seguridad, asociadas con el intercambio electrónico de datos, comercio electrónico y comunicaciones electrónicas, y los requisitos de controles.

La información puede verse comprometida por la falta de conocimiento, política o procedimientos para el uso de los medios de intercambio de información, por ejemplo, ser escuchado al hablar desde un teléfono móvil en un lugar público, dirección equivocada en un mensaje de correo electrónico, mensajes dejados en máquinas contestadoras escuchadas, acceso no-autorizado al sistema de correo de voz.

La información puede verse comprometida si fallan los medios de comunicación, son escuchados o interrumpidos. La información puede verse comprometida si usuarios no-autorizados tienen acceso a ella.

Mediante el etiquetado y clasificación de dicha información en distintos niveles de seguridad y/o cifrado para el intercambio de información entre dispositivos, como ya sucediera en el intercambio de información dentro del sistema o dispositivo.

2.5.7.2 Acuerdos de intercambio

El acuerdo de intercambio debiera considerar las siguientes condiciones de seguridad:

a) Manejo de las responsabilidades para el control y notificación de la transmisión y recepción.

b) Procedimientos para notificar al remitente la transmisión y recepción.

c) Procedimientos para asegurar el rastreo y no-repudio.

d) Estándares técnicos mínimos para el empaquetado y la transmisión.

e) Responsabilidades y obligaciones en el evento de incidentes de seguridad de la información, como la pérdida de datos.

f) Uso de un sistema de etiquetado acordado para la información confidencial o crítica, asegurando que el significado de las etiquetas sea entendido inmediatamente y que la información sea adecuadamente protegida.

g) Propiedad y responsabilidades de la protección de datos, derechos de autor, licencias de software y consideraciones similares.

h) Estándares técnicos para grabar y leer la información y software.

i) Cualquier control especial que se pueda requerir para proteger los elementos confidenciales, como claves criptográficas.

Se debieran establecer y mantener las políticas, procedimientos y estándares para proteger la información y medios físicos en tránsito, y se debiera hacer referencia en los acuerdos de intercambio.

El contenido de seguridad de cualquier acuerdo debiera reflejar la sensibilidad de la información comercial involucrada.

Los acuerdos pueden ser electrónicos o manuales, y pueden tomar la forma de contratos formales o condiciones de empleo. Para la información sensible, los mecanismos específicos utilizados para el intercambio de dicha información debieran ser consistentes para todos los sistemas y tipos de acuerdos.

2.5.7.3 Mensajes electrónicos

Se debiera proteger adecuadamente la información involucrada en mensajes electrónicos.

Las consideraciones de seguridad para los mensajes electrónicos debieran incluir lo siguiente:

a) Proteger los mensajes del acceso no-autorizado, modificación o negación del servicio.

b) Asegurar la correcta dirección y transporte del mensaje.

c) Confiabilidad y disponibilidad general del servicio.

d) Consideraciones legales, por ejemplo los requisitos para firmas electrónicas.

e) Obtener la aprobación antes de utilizar los servicios públicos externos como un mensaje instantáneo o intercambio de archivos.

f) Niveles mayores de autenticación controlando el acceso de las redes de acceso público.

Los mensajes electrónicos como el correo electrónico, Intercambio Electrónico de Datos (EDI), y los mensaje instantáneos representan un papel cada vez más importante en las comunicaciones comerciales. Los mensajes electrónicos tienen riesgos diferentes de las comunicaciones basadas en papel.

Puesto que estamos tratando con un sistema operativo diseñado para dispositivos en los que disponemos de cuentas de correo electrónico asociadas, y a menudo se trata de dispositivos móviles, es común su uso para el envío de correos y mensajes electrónicos y por tanto debemos destacar la seguridad de los mismos.

2.5.8 Servicios de comercio electrónico

El objetivo es asegurar la seguridad de los servicios de comercio electrónico y su uso seguro.

Se debieran considerar las implicancias de seguridad asociadas con el uso de servicios de comercio electrónico, incluyendo las transacciones en-línea, y los requisitos de controles. También se debieran considerar la integridad y la disponibilidad de la información publicada electrónicamente a través de los sistemas públicamente disponibles.

Dada la creciente tendencia de realizar tanto compras como transacciones comerciales a través de la red, y el acceso de los dispositivos bajo la plataforma Android a la misma para realizar tales acciones, cabe destacar en este punto el reforzamiento del sistema en la seguridad de dichos servicios.

2.5.8.1 Comercio electrónico

La información involucrada en el comercio electrónico que pasa a través de redes públicas debiera protegerse de la actividad fraudulenta, divulgación no autorizada y modificación.

Las consideraciones de seguridad para el comercio electrónico debieran incluir lo siguiente:

a) El nivel de confianza que cada parte requiere de la identidad de la otra, por ejemplo, a través de la autenticación.

b) Los procesos de autorización asociados con aquellos que pueden establecer precios, emitir o firmar documentos de comercialización.

c) Asegurar que los socios comerciales estén totalmente informados de sus autorizaciones.

d) Determinar y cumplir con los requisitos para la confidencialidad, integridad y recepción de documentos claves, y el no-repudio de los contratos, por ejemplo, asociado con procesos de licitación y contratos.

e) El nivel de confianza requerido para la integridad de las listas de precios publicitadas.

f) La confidencialidad de cualquier dato o información confidencial.

g) La confidencialidad e integridad de cualquier transacción, información de pago, detalles de la dirección de entrega y la confirmación de la recepción.

h) El grado de verificación apropiado para valorar la información de pago suministrada por un cliente.

i) Seleccionar la forma de liquidación más apropiada del pago para evitar el fraude.

j) El nivel de protección requerido para mantener la confidencialidad e integridad de la información de la orden.

k) Evitar la pérdida o duplicación de la información de la transacción.

l) La responsabilidad asociada con cualquier transacción fraudulenta.

m) Requisitos de seguridad.

Muchas de las consideraciones arriba mencionadas se pueden tratar mediante la aplicación de controles criptográficos, tomando en cuenta el cumplimiento de los requisitos legales.

Los acuerdos de comercio electrónico entre socios debieran ser respaldados por un contrato documentado el cual compromete a ambas partes a los términos acordados para la comercialización, incluyendo los detalles de la autorización. Pueden ser necesarios otros acuerdos con los proveedores del servicio de la información y la red de valor agregado.

Los sistemas de negociación pública debieran comunicar sus términos del negocio a sus clientes.

Se debiera tomar en consideración la resistencia al ataque del host(s) utilizado(s) para el comercio electrónico, y las implicancias de seguridad de cualquier interconexión de la red requerida para la implementación de los servicios de comercio electrónico.

El comercio electrónico es vulnerable a un número de amenazas de la red que pueden resultar en una actividad fraudulenta, disputa de contrato y divulgación o modificación de la información.

El comercio electrónico puede utilizar métodos de autenticación, por ejemplo, criptografía de clave pública y firmas electrónicas para reducir los riesgos. También, se pueden utilizar terceros confiables cuando se necesitan dichos servicios.

Como hemos mencionado anteriormente, la utilización de los dispositivos bajo la plataforma para la utilización en el comercio electrónico, debiera conllevar un reforzamiento en la seguridad de dichas acciones, mediante por ejemplo, la utilización de técnicas o controles criptográficos para los datos utilizados en dichas acciones.

2.5.8.2 Transacciones en-línea

Se debiera proteger la información involucrada en las transacciones en-línea para evitar una transmisión incompleta, enrutamiento equivocado, alteración no-autorizada del mensaje, divulgación no-autorizada, duplicación o repetición no-autorizada del mensaje.

Las consideraciones de seguridad para las transacciones en-línea debieran incluir lo siguiente:

a) El uso de firmas electrónicas por cada una de las partes involucradas en la transacción.

b) Todos los aspectos de la transacción, es decir, asegurando que:

   - Las credenciales de usuario de todas las partes sean válidas y verificadas.
   - Que la transacción permanezca confidencial.
   - Que se mantenga la privacidad asociada con todas las partes involucradas.

c) El camino de las comunicaciones entre las partes involucradas debiera ser codificado.

d) Que los protocolos utilizados para comunicarse entre todas las partes involucradas sean seguros.

e) Asegurar que el almacenaje de los detalle de la transacción se localice fuera de cualquier ambiente público accesible, por ejemplo, en una plataforma de almacenaje de acceso público para el resto de aplicaciones, y no se mantenga y exponga en un medio de almacenaje directamente accesible desde Internet.

f) Cuando se utilice una autoridad confiable (por ejemplo, para propósitos de emitir y mantener firmas digitales y/o certificados digitales) la seguridad es integrada e introducida durante todo el proceso de gestión de firma/certificado de principio a fin.

La extensión de los controles adoptados debiera conmensurarse con el nivel del riesgo asociado con cada forma de transacción en-línea.

Las transacciones pueden necesitar cumplir con leyes, reglas y regulaciones en la jurisdicción en la cual se genera, procesa, completa y/o almacena la transacción.

Existen muchas formas de transacciones que se pueden realizar de una manera en-línea; por ejemplo, contractuales, financieras, etc.

Como en el punto anterior, el continuo crecimiento de la cuota de mercado para la plataforma Android está incentivando la creación de aplicaciones para la realización de transacciones financieras, cabe decir que este tipo de aplicaciones requieren de un alto nivel de seguridad como expondremos a continuación.

2.5.8.3 Información públicamente disponible

Se debiera proteger la integridad de la información puesta a disposición en un sistema públicamente disponible para evitar una modificación no-autorizada.

El software, datos y otra información que requiere un alto nivel de integridad, puesta a disposición en un sistema públicamente disponible, se debiera proteger mediante los mecanismos apropiados; por ejemplo, firmas digitales. El sistema o partes del mismo públicamente disponibles debieran ser probados en busca de debilidades y fallos antes que la información esté disponible.

Debiera existir un proceso de aprobación formal antes que la información sea puesta a disposición pública. Además, se debiera verificar y aprobar todos los datos de entrada provistos desde fuera del sistema o por las aplicaciones instaladas en la plataforma.

Se debieran controlar cuidadosamente los sistemas de publicación electrónica, especialmente aquellos que permiten retroalimentación y el ingreso directo de información de manera que:

a) La información se obtenga cumpliendo con la legislación de protección de datos.

b) Los datos de entrada de información para, y procesado por, el sistema de publicación será procesado completa y exactamente de una manera oportuna.

c) Se protegerá la información confidencial durante la recolección, procesamiento y almacenaje.

d) El acceso al sistema de publicación no permite el acceso involuntario a las redes con las cuales se conecta el sistema.

Por ejemplo la información compartida entre aplicaciones, pues a menudo la modificación de la misma puede ser la causa de una brecha en la seguridad, también puede ser objetivo de ataques la información pública a disposición de todas las aplicaciones que la soliciten o requieran almacenada en el dispositivo o sistema; como contrapartida para aumentar la seguridad de la información mencionada anteriormente podemos emplear mecanismos de firma digital.

2.6 Control del acceso

2.6.1 Requisitos del sistema para el control del acceso

El objetivo es controlar el acceso a la información.

Se debiera controlar el acceso a la información, medios de procesamiento de la información y seguridad.

Las reglas de control del acceso debieran tomar en cuenta las políticas para la divulgación y autorización de la información.

Cabe destacar este punto puesto que la información almacenada en los dispositivos con sistema operativo Android a menudo es de carácter confidencial, ya que este tipo de dispositivos se emplean como herramienta de trabajo en numerosas organizaciones, por tanto, este punto es más importante en el uso de dispositivos empresariales bajo la plataforma.

2.6.2 Gestión de acceso del usuario

El objetivo es asegurar el acceso del usuario autorizado y evitar el acceso no autorizado a los sistemas de información.

Se debieran establecer procedimientos formales para controlar la asignación de los derechos de acceso a los sistemas, dispositivos y servicios de información.

Los procedimientos debieran abarcar todas las etapas en el ciclo de vida del acceso del usuario, desde el registro inicial de usuarios nuevos hasta el des-registro final de los usuarios que ya no requieren acceso a los sistemas y servicios de información. Cuando sea apropiado, se debiera prestar atención especial a la necesidad de controlar la

asignación de derechos de acceso privilegiados, lo que permite a los usuarios superar los controles del sistema.

En este punto destacamos el uso de los controles de acceso a los dispositivos bajo la plataforma Android, por ejemplo el sistema de cuentas de usuario gestionado mediante identificadores de usuario y claves secretas, o por ejemplo mediante patrones de acceso aprovechando el carácter táctil de manejo de estos dispositivos.

2.6.3 Gestión de privilegios

Se debiera restringir y controlar la asignación y uso de privilegios.

Los sistemas multiusuario que requieren protección contra el acceso no autorizado debieran controlar la asignación de privilegios a través de un proceso de autorización formal. Se debieran considerar los siguientes pasos:

a) Los privilegios de acceso asociados con cada producto del sistema, por ejemplo, sistema de operación, sistema de gestión de base de datos y cada aplicación, y se debieran identificar los usuarios a quienes se les necesita asignar privilegios.

b) Los privilegios se debieran asignar a los usuarios sobre la base de "sólo lo que necesitan saber" y sobre una base de evento-por-evento en línea con la política de control del acceso, es decir, los requisitos mínimos para su rol funcional, sólo cuando se necesitan.

c) Se debiera mantener un proceso de autorización y un registro de todos los privilegios asignados. No se debieran otorgar privilegios hasta que se complete el proceso de autorización.

d) Se debiera promover el desarrollo y uso de rutinas del sistema para evitar la necesidad de otorgar privilegios a los usuarios.

e) Se debiera promover el desarrollo y uso de los programas que evitan la necesidad de ejecutarse con privilegios.

f) Los privilegios se debieran asignar a un ID de usuario diferente de aquellos utilizados para el uso normal del negocio.

Como hemos destacados en puntos anteriores en relación con un sistema de cuentas de usuario para estos dispositivos de carácter empresarial, cabe destacar en este punto el empleo de privilegios para dichas cuentas, todo ello gestionado por el administrador de dichos privilegios.

2.6.4 Control de acceso a la red

El objetivo es evitar el acceso no autorizado a los servicios de la red.

Se debiera controlar el acceso a los servicios de redes tanto internas como externas.

Destacamos en este punto el uso de controles apropiados para la conexión de los dispositivos bajo la plataforma Android a diversas redes, puesto que dichos dispositivos están diseñados para la conectividad de los mismos a la red.

2.6.4.1 Autenticación del usuario para las conexiones externas

Se debieran utilizar métodos de autenticación apropiados para controlar el acceso de usuarios remotos.

Las conexiones externas proporcionan un potencial para el acceso no autorizado a la información comercial. Existen diferentes tipos de métodos de autenticación, algunos de estos proporcionan un mayor nivel de protección que otros, por ejemplo, los métodos basados en el uso de las técnicas criptográficas pueden proporcionar una autenticación sólida. Es importante determinar el nivel de protección requerido mediante una evaluación del riesgo. Esto es necesario para la selección apropiada de un método de autenticación.

La autenticación de los usuarios remotos se puede lograr utilizando, por ejemplo, una técnica basada en criptografía, dispositivos de hardware o un protocolo de desafío/respuesta. Las posibles implementaciones de tales técnicas se pueden encontrar en varias soluciones de la red privada virtual (VPN). También se pueden utilizar las líneas privadas dedicadas para proporcionar la seguridad de la fuente de conexiones.

La autenticación del nodo puede servir como un medio alternativo para la autenticación de los grupos de usuarios remotos, cuando están conectados a un medio de cómputo seguro y compartido. Se pueden utilizar técnicas criptográficas, por ejemplo, basadas en los certificados de las máquinas para la autenticación del nodo.

Se debieran implementar controles de autenticación adicionales para controlar el acceso a las redes inalámbricas. En particular, se necesita prestar cuidado especial a la selección de controles para las redes inalámbricas debido a las mayores oportunidades para una intercepción e inserción no-detectada de tráfico de la red.

Como por ejemplo la conexión o intercambio de información entre diversos dispositivos bajo la plataforma Android o la conectividad de los mismos en redes inalámbricas 3G.

2.6.4.2 Identificación del dispositivo en las redes

La identificación automática del dispositivo se debiera considerar como un medio para autenticar las conexiones de ubicaciones y dispositivos específicos.

La identificación del dispositivo se puede utilizar si es importante que la comunicación sólo sea iniciada desde una ubicación o dispositivo específico. Se puede

utilizar un identificador dentro o incorporado en el dispositivo para indicar si este está autorizado a conectarse a la red.

Estos identificadores debieran indicar claramente a cuál red está autorizado a conectarse el dispositivo, si existe más de una red y particularmente si estas redes tienen diferentes grados de confidencialidad. Puede ser necesario considerar la protección física del dispositivo para mantener la seguridad del identificador del mismo.

Este control puede complementarse con otras técnicas para autenticar al usuario del dispositivo. Adicionalmente, se puede aplicar la identificación del dispositivo para la autenticación del usuario.

2.6.4.3 Control de conexión a la red

Para las redes compartidas, se debiera restringir la capacidad de los usuarios del sistema para conectarse a la red, en línea con la política de control de acceso y los requisitos de las aplicaciones comerciales.

Como hemos mencionado en puntos anteriores el sistema debiera controlar y gestionar los usuarios y aplicaciones del dispositivo que tiene permisos para conectarse a la red.

2.6.5 Control del acceso al sistema operativo

El objetivo es evitar el acceso no autorizado a los sistemas operativos.

Se debieran utilizar medios de seguridad para restringir el acceso a los sistemas operativos a los usuarios autorizados. Los medios debieran tener la capacidad para:

a) Autenticar a los usuarios autorizados, en concordancia con una política de control de acceso definida.

b) Registrar los intentos exitosos y fallidos de autenticación del sistema.

c) Registrar el uso de los privilegios especiales del sistema.

d) Emitir alarmas cuando se violan las políticas de seguridad del sistema.

e) Proporcionar los medios de autenticación apropiados.

f) Cuando sea apropiado, restringir el tiempo de conexión de los usuarios.

Como hemos mencionado en punto anteriores resaltamos la importancia del control de acceso a la plataforma para usuarios autorizados por el sistema operativo mediante el empleo de cuentas de usuario, claves y métodos de privilegios o roles.

2.6.5.1 Procedimientos para un registro seguro

El acceso al sistema operativo debiera ser controlado mediante un procedimiento de registro seguro.

El procedimiento para registrarse en un sistema operativo debiera ser diseñado de manera que minimice la oportunidad de un acceso no autorizado. Por lo tanto, el procedimiento para registrarse debiera divulgar el mínimo de información acerca del sistema para evitar proporcionar al usuario no-autorizado ninguna ayuda innecesaria.

Un buen procedimiento de registro:

a) No debiera mostrar identificadores del sistema o aplicación hasta que se haya completado satisfactoriamente el proceso de registro.

b) Debiera mostrar la advertencia general que al dispositivo sólo pueden tener acceso los usuarios autorizados.

c) No debiera proporcionar mensajes de ayuda durante el procedimiento de registro que ayuden al usuario no-autorizado.

d) Sólo debiera validar la información del registro después de completar toda la entrada de datos. Si surge una condición de error, el sistema debiera indicar qué parte de los datos es correcta o incorrecta.

e) Debiera limitar el número de intentos de registro infructuosos permitidos, por ejemplo, tres intentos, y debiera considerar:

   - Registrar los intentos exitosos y fallidos.

   - Forzar un tiempo de espera antes de permitir más intentos de registro o rechazar cualquier otro intento sin una autorización específica.

   - Desconectar las conexiones de vínculo a los datos.

   - Establecer el número de re-intentos de clave secreta en conjunción con la longitud mínima de la clave secreta y el valor del sistema que se está protegiendo.

f) Debiera limitar el tiempo máximo y mínimo permitido para el procedimiento de registro. Si se excede este tiempo, el sistema debiera terminar el registro.

g) Debiera mostrar la siguiente información a la culminación de un registro satisfactorio:

   - Fecha y hora del registro satisfactorio previo.

- Detalles de cualquier intento infructuoso desde el último registro satisfactorio.

h) No debiera mostrar la clave secreta que se está ingresando o considerar esconder los caracteres de la clave secreta mediante símbolos.

i) No debiera transmitir claves secretas en un texto abierto a través de la red.

2.6.5.2 Uso de las utilidades del sistema

Se debiera restringir y controlar estrechamente el uso de los programas de utilidad que podrían ser capaces de superar los controles del sistema y la aplicación.

Se debieran considerar los siguientes lineamientos para el uso de las utilidades del sistema:

a) Uso de los procedimientos de identificación, autenticación y autorización para las utilidades del sistema.

b) Segregación de las utilidades del sistema del software de la aplicación.

c) Limitar el uso de las utilidades del sistema a un número práctico mínimo de usuarios autorizados y confiables.

d) Autorización para el uso "ad-hoc" de las utilidades del sistema.

e) Registro de todo uso de las utilidades del sistema.

f) Definir y documentar los niveles de autorización de las utilidades del sistema.

g) Eliminar o inutilizar todas las utilidades innecesarias basadas en software, así como el software del sistema que sea innecesario.

2.6.5.3 Cierre de una sesión por inactividad

Las sesiones inactivas debieran ser cerradas después de un período de inactividad definido.

Un dispositivo de cierre debiera borrar la pantalla de la sesión y también, posiblemente más adelante, cerrar la aplicación y las sesiones en red después de un período de inactividad definido. El tiempo de espera antes del cierre debiera reflejar los riesgos de seguridad del área, la clasificación de la información que está siendo manejada y la aplicación siendo utilizada, y los riesgos relacionados con los usuarios del equipo.

Una forma de limitación del dispositivo de cierre puede ser provista para algunos sistemas, este dispositivo borra la pantalla y evita el acceso no autorizado pero no cierra las sesiones de la aplicación o la red.

Este control es particularmente importante en las ubicaciones de alto riesgo, las cuales incluyen áreas públicas o externas fuera de la gestión de seguridad de cualquier organización, como es el caso para este tipo de dispositivos bajo la plataforma Android. Se debieran cerrar las sesiones para evitar el acceso no autorizado de personas y la negación de ataques del servicio.

2.6.5.4 Limitación del tiempo de conexión

Se debieran utilizar restricciones sobre los tiempos de conexión para proporcionar seguridad adicional para las aplicaciones de alto riesgo.

Se debieran considerar controles sobre el tiempo de conexión para las aplicaciones sensibles, especialmente desde ubicaciones de alto riesgo, por ejemplo, áreas públicas o externas que están fuera de la gestión de seguridad de cualquier organización. Los ejemplos de tales restricciones incluyen:

a) Utilizar espacios de tiempo predeterminados, por ejemplo, para transmisiones de archivos en lotes, o sesiones interactivas regulares de corta duración.

b) Restringir los tiempos de conexión a los horarios laborales normales, si no existe ningún requerimiento para sobre-tiempo o una operación de horario extendido.

c) Considerar la re-autenticación cada cierto intervalo de tiempo.

Limitar el período permitido para las conexiones con los servicios de cómputo reduce la ventana de oportunidad para el acceso no autorizado.

2.6.6 Control de acceso a la aplicación y la información

El objetivo es evitar el acceso no autorizado a la información mantenida en los sistemas de aplicación.

Se debieran utilizar medios de seguridad para restringir el acceso a los sistemas de aplicación.

El acceso lógico al software de la aplicación y la información se debiera limitar a los usuarios autorizados. Los sistemas de aplicación debieran:

a) Controlar el acceso del usuario a la información y las funciones del sistema de aplicación, en concordancia con una política de control de acceso definida.

b) Proporcionar protección contra un acceso no autorizado de cualquier utilidad, software del sistema de operación y software malicioso que sea capaz de superar o pasar los controles del sistema o la aplicación.

c) No comprometer a otros sistemas con los cuales se comparten recursos de información.

2.6.7 Computación móvil

El objetivo es asegurar la seguridad de la información cuando se utiliza medios de computación y tele-trabajo móviles.

La protección requerida se debiera conmensurar con los riesgos que causan estas maneras de trabajo específicas. Cuando se utiliza computación móvil, se debieran considerar los riesgos de trabajar en un ambiente desprotegido y se debiera aplicar la protección apropiada. En el caso de tele-trabajo, toda organización debiera aplicar protección al lugar de tele-trabajo y asegurar que se establezcan los arreglos adecuados para esta manera de trabajar.

Este punto representa el tipo de funcionalidad para el que está diseñada la plataforma Android y el motivo por el cual se debe reforzar la seguridad en este tipo de plataformas para toda organización que emplee estos dispositivos.

2.6.7.1 Computación y comunicaciones móviles

Se debiera establecer una política y adoptar las medidas de seguridad apropiadas para proteger contra los riesgos de utilizar medios de computación y comunicación móvil.

Cuando se utilizan medios de computación y comunicación móvil, por ejemplo, “notebooks”, “palmtops”, “laptops”, “tablets pc”, “pdas” y teléfonos móviles, se debiera tener especial cuidado en asegurar que no se comprometa la información comercial. La política de computación móvil debiera tomar en cuenta los riesgos de trabajar con dispositivos de computación móvil en ambientes desprotegidos.

La política de computación móvil debiera incluir los requisitos de protección física, controles de acceso, técnicas criptográficas, respaldos (back-up) y protección contra virus.

Esta política también debiera incluir reglas y consejos para la conexión de medios móviles con las redes y lineamientos para el uso de estos medios en lugares públicos.

Se debiera tener cuidado cuando se utiliza medios de computación móvil en lugares públicos, salas de reuniones y otras áreas desprotegidas fuera de los locales de la organización. Se debiera establecer la protección para evitar el acceso no autorizado o

divulgación de la información almacenada y procesada por estos medios, por ejemplo, utilizando técnicas criptográficas.

Los usuarios de los medios de computación móvil que se encuentran en lugares públicos debieran tener cuidado en evitar que personas no autorizadas vean su trabajo. Se debiera establecer procedimientos contra los software maliciosos y se debieran mantener actualizados.

Los respaldos (back-up) de la información comercial crítica se debieran realizar con regularidad. Debiera estar disponible el dispositivo para permitir realizar un respaldo rápido y fácil de la información. Se debiera dar a estos respaldos la protección adecuada, por ejemplo, contra el robo o pérdida de información.

Se debiera dar la protección adecuada al uso de medios móviles conectados a las redes. El acceso remoto a la información comercial a través de una red pública utilizando medios de computación móvil sólo debiera realizarse después de una satisfactoria identificación y autenticación, y con los controles de acceso adecuados en funcionamiento.

Los medios de computación móvil también debieran estar físicamente protegidos contra robo especialmente cuando se les deja en, por ejemplo, automóviles y otros medios de transporte, habitaciones de hotel, centros de conferencias y lugares de reunión. Se debiera establecer un procedimiento específico tomando en cuenta los requisitos legales, de seguros y otros requisitos de seguridad de la organización para casos de robo o pérdida de los medios móviles. El equipo que contiene información comercial importante, confidencial y/o crítica no se debiera dejar desatendido y, cuando sea posible, debiera estar asegurado físicamente, o se pueden utilizar seguros para proteger los dispositivos.

Se debiera planear capacitación para el personal que utiliza computación móvil para elevar el nivel de conciencia sobre los riesgos adicionales resultantes de esta forma de trabajo y los controles que se debieran implementar.

Las conexiones inalámbricas a la red móvil son similares a otros tipos de conexión en red, pero tienen diferencias importantes que se debieran considerar cuando se identifican los controles. Las diferencias típicas son:

a) Algunos protocolos de seguridad inalámbricos aún son inmaduros y tienen debilidades conocidas.

b) La información almacenada en las computadoras móviles tal vez no tiene respaldo (back-up) debido a la banda ancha limitada de la red y/o porque el equipo móvil puede no estar conectado en las horas en que se realizan los respaldos.

3 REAL DECRETO 1720/2007

Reglamento de desarrollo de la Ley Orgánica 15/1999, de 13 de diciembre, de protección de datos de carácter personal.

En este apartado del anexo vamos a indicar y comentar los artículos del Reglamento de desarrollo (de 21 de Diciembre de 2007) de la Ley Orgánica 15/1999, de protección de datos de carácter personal que pueden aplicarse a la plataforma Android ya que a menudo los dispositivos bajo esta plataforma contienen datos de carácter personal (el nombre de cualquier persona lo es), dichos dispositivos están empezando a ser usados como herramientas de trabajo por organizaciones en todo el mundo.

Indicaremos los artículos que puedan ser aplicables a los dispositivos bajo la plataforma de forma íntegra, añadiendo a continuación comentarios para destacar los puntos que consideramos importantes en cada uno de ellos.

Datos de carácter personal: Cualquier información numérica, alfabética, gráfica, fotográfica, acústica o de cualquier otro tipo concerniente a personas físicas identificadas o identificables.

TÍTULO VIII

De las medidas de seguridad en el tratamiento de datos de carácter personal.

CAPÍTULO I

Disposiciones generales

Artículo 79. Alcance.

Los responsables de los tratamientos o los ficheros y los encargados del tratamiento deberán implantar las medidas de seguridad con arreglo a lo dispuesto en este Título, con independencia de cuál sea su sistema de tratamiento.

Artículo 80. Niveles de seguridad.

Las medidas de seguridad exigibles a los ficheros y tratamientos se clasifican en tres niveles: básico, medio y alto.

Artículo 81. Aplicación de los niveles de seguridad.

1. Todos los ficheros o tratamientos de datos de carácter personal deberán adoptar las medidas de seguridad calificadas de nivel básico.

2. Deberán implantarse, además de las medidas de seguridad de nivel básico, las medidas de nivel medio, en los siguientes ficheros o tratamientos de datos de carácter personal:

a) Los relativos a la comisión de infracciones administrativas o penales.

b) Aquéllos cuyo funcionamiento se rija por el artículo 29 de la Ley Orgánica 15/1999, de 13 de diciembre.

c) Aquéllos de los que sean responsables Administraciones tributarias y se relacionen con el ejercicio de sus potestades tributarias.

d) Aquéllos de los que sean responsables las entidades financieras para finalidades relacionadas con la prestación de servicios financieros.

e) Aquéllos de los que sean responsables las Entidades Gestoras y Servicios Comunes de la Seguridad Social y se relacionen con el ejercicio de sus competencias. De igual modo, aquellos de los que sean responsables las mutuas de accidentes de trabajo y enfermedades profesionales de la Seguridad Social.

f) Aquéllos que contengan un conjunto de datos de carácter personal que ofrezcan una definición de las características o de la personalidad de los ciudadanos y que permitan evaluar determinados aspectos de la personalidad o del comportamiento de los mismos.

3. Además de las medidas de nivel básico y medio, las medidas de nivel alto se aplicarán en los siguientes ficheros o tratamientos de datos de carácter personal:

a) Los que se refieran a datos de ideología, afiliación sindical, religión, creencias, origen racial, salud o vida sexual.

b) Los que contengan o se refieran a datos recabados para fines policiales sin consentimiento de las personas afectadas.

c) Aquéllos que contengan datos derivados de actos de violencia de género.

4. A los ficheros de los que sean responsables los operadores que presten servicios de comunicaciones electrónicas disponibles al público o exploten redes públicas de comunicaciones electrónicas respecto a los datos de tráfico y a los datos de localización, se aplicarán, además de las medidas de seguridad de nivel básico y medio, la medida de seguridad de nivel alto contenida en el artículo 103 de este reglamento.

5. En caso de ficheros o tratamientos de datos de ideología, afiliación sindical, religión, creencias, origen racial, salud o vida sexual bastará la implantación de las medidas de seguridad de nivel básico cuando:

a) Los datos se utilicen con la única finalidad de realizar una transferencia dineraria a las entidades de las que los afectados sean asociados o miembros.

b) Se trate de ficheros o tratamientos en los que de forma incidental o accesoria se contengan aquellos datos sin guardar relación con su finalidad.

6. También podrán implantarse las medidas de seguridad de nivel básico en los ficheros o tratamientos que contengan datos relativos a la salud, referentes exclusivamente al grado de discapacidad o la simple declaración de la condición de discapacidad o invalidez del afectado, con motivo del cumplimiento de deberes públicos.

7. Las medidas incluidas en cada uno de los niveles descritos anteriormente tienen la condición de mínimos exigibles, sin perjuicio de las disposiciones legales o reglamentarias específicas vigentes que pudieran resultar de aplicación en cada caso o las que por propia iniciativa adoptase el responsable del fichero.

8. A los efectos de facilitar el cumplimiento de lo dispuesto en este título, cuando en un sistema de información existan ficheros o tratamientos que en función de su finalidad o uso concreto, o de la naturaleza de los datos que contengan, requieran la aplicación de un nivel de medidas de seguridad diferente al del sistema principal, podrán segregarse de este último, siendo de aplicación en cada caso el nivel de medidas de seguridad correspondiente y siempre que puedan delimitarse los datos afectados y los usuarios con acceso a los mismos, y que esto se haga constar en el documento de seguridad.

Destacamos de este artículo que lo normal es que los dispositivos bajo la plataforma Android tengan datos de nivel básico pero podrían llegar a tener datos incluso de nivel alto, por ejemplo los ficheros o correos que contengan datos de salud, como podrían ser los adjuntos que se descargara un profesional de la sanidad. También cabe destacar el punto 4 de dicho artículo puesto que estos dispositivos están diseñados con carácter para prestar servicio de comunicaciones electrónicas.

Artículo 82. Encargado del tratamiento.

1. Cuando el responsable del fichero o tratamiento facilite el acceso a los datos, a los soportes que los contengan o a los recursos del sistema de información que los trate, a un encargado de tratamiento que preste sus servicios en los locales del primero deberá hacerse constar esta circunstancia en el documento de seguridad de dicho responsable, comprometiéndose el personal del encargado al cumplimiento de las medidas de seguridad previstas en el citado documento.

Cuando dicho acceso sea remoto habiéndose prohibido al encargado incorporar tales datos a sistemas o soportes distintos de los del responsable, este último deberá hacer constar esta circunstancia en el documento de seguridad del responsable, comprometiéndose el personal del encargado al cumplimiento de las medidas de seguridad previstas en el citado documento.

2. Si el servicio fuera prestado por el encargado del tratamiento en sus propios locales, ajenos a los del responsable del fichero, deberá elaborar un documento de seguridad en los términos exigidos por el artículo 88 de este reglamento o completar el que ya hubiera elaborado, en su caso, identificando el fichero o tratamiento y el responsable del mismo e incorporando las medidas de seguridad a implantar en relación con dicho tratamiento.

3. En todo caso, el acceso a los datos por el encargado del tratamiento estará sometido a las medidas de seguridad contempladas en este reglamento.

Destacamos este artículo puesto que los soportes bajo la plataforma están diseñados para el tratamiento de información por personal de cualquier organización, por lo tanto, es necesario que el responsable de seguridad controle el cumplimiento de las medidas de seguridad previstas para el uso de dichos soportes y datos por parte del personal de la organización, si bien el cumplimiento de este artículo depende de medidas administrativas y no de los aspectos técnicos del propio sistema operativo.

Artículo 83. Prestaciones de servicios sin acceso a datos personales.

El responsable del fichero o tratamiento adoptará las medidas adecuadas para limitar el acceso del personal a datos personales, a los soportes que los contengan o a los recursos del sistema de información, para la realización de trabajos que no impliquen el tratamiento de datos personales.

Cuando se trate de personal ajeno, el contrato de prestación de servicios recogerá expresamente la prohibición de acceder a los datos personales y la obligación de secreto respecto a los datos que el personal hubiera podido conocer con motivo de la prestación del servicio.

Destacamos este artículo puesto que los dispositivos bajo la plataforma pueden contener información relevante y por lo tanto en necesario una limitación de acceso al personal que puede utilizar dichos dispositivos, si bien el cumplimiento de este artículo depende de medidas administrativas y no de los aspectos técnicos del propio sistema operativo.

Artículo 84. Delegación de autorizaciones.

Las autorizaciones que en este título se atribuyen al responsable del fichero o tratamiento podrán ser delegadas en las personas designadas al efecto. En el documento de seguridad deberán constar las personas habilitadas para otorgar estas autorizaciones así como aquellas en las que recae dicha delegación. En ningún caso esta designación supone una delegación de la responsabilidad que corresponde al responsable del fichero.

Destacamos este artículo puesto que los dispositivos bajo la plataforma Android pueden ser usados dentro de una misma organización por diferentes empleados con las responsabilidades que conllevan tanto la información contenida en los mismos que

puede ser de carácter relevante y el tratamiento de la misma por parte del usuario del dispositivo, si bien el cumplimiento de este artículo depende de medidas administrativas y no de los aspectos técnicos del propio sistema operativo.

Artículo 85. Acceso a datos a través de redes de comunicaciones.

Las medidas de seguridad exigibles a los accesos a datos de carácter personal a través de redes de comunicaciones, sean o no públicas, deberán garantizar un nivel de seguridad equivalente al correspondiente a los accesos en modo local, conforme a los criterios establecidos en el artículo 80.

Este artículo es aplicable a los dispositivos bajo la plataforma Android ya que están diseñados con unas características claras para la conectividad a las redes de comunicación existentes en la actualidad.

Artículo 86. Régimen de trabajo fuera de los locales del responsable del fichero o encargado del tratamiento.

1. Cuando los datos personales se almacenen en dispositivos portátiles o se traten fuera de los locales del responsable de fichero o tratamiento, o del encargado del tratamiento será preciso que exista una autorización previa del responsable del fichero o tratamiento, y en todo caso deberá garantizarse el nivel de seguridad correspondiente al tipo de fichero tratado.

2. La autorización a la que se refiere el párrafo anterior tendrá que constar en el documento de seguridad y podrá establecerse para un usuario o para un perfil de usuarios y determinando un periodo de validez para las mismas.

De este artículo destacamos la parte que nos habla de los dispositivos portátiles, es aplicable a los dispositivos bajo la plataforma Android precisamente por la portabilidad de los mismos.

Artículo 87. Ficheros temporales o copias de trabajo de documentos.

1. Aquellos ficheros temporales o copias de documentos que se hubiesen creado exclusivamente para la realización de trabajos temporales o auxiliares deberán cumplir el nivel de seguridad que les corresponda conforme a los criterios establecidos en el artículo 81.

2. Todo fichero temporal o copia de trabajo así creado será borrado o destruido una vez que haya dejado de ser necesario para los fines que motivaron su creación.

De este artículo destacamos el uso de ficheros temporales que emplea el sistema operativo para la realización de diversas tareas, como por ejemplo, a la hora de realizar actualizaciones del sistema o la navegación web.

Artículo 88. El documento de seguridad.

1. El responsable del fichero o tratamiento elaborará un documento de seguridad que recogerá las medidas de índole técnica y organizativa acordes a la normativa de seguridad vigente que será de obligado cumplimiento para el personal con acceso a los sistemas de información.

2. El documento de seguridad podrá ser único y comprensivo de todos los ficheros o tratamientos, o bien individualizado para cada fichero o tratamiento. También podrán elaborarse distintos documentos de seguridad agrupando ficheros o tratamientos según el sistema de tratamiento utilizado para su organización, o bien atendiendo a criterios organizativos del responsable. En todo caso, tendrá el carácter de documento interno de la organización.

3. El documento deberá contener, como mínimo, los siguientes aspectos:

a) Ámbito de aplicación del documento con especificación detallada de los recursos protegidos.

b) Medidas, normas, procedimientos de actuación, reglas y estándares encaminados a garantizar el nivel de seguridad exigido en este reglamento.

c) Funciones y obligaciones del personal en relación con el tratamiento de los datos de carácter personal incluidos en los ficheros.

d) Estructura de los ficheros con datos de carácter personal y descripción de los sistemas de información que los tratan.

e) Procedimiento de notificación, gestión y respuesta ante las incidencias.

f) Los procedimientos de realización de copias de respaldo y de recuperación de los datos en los ficheros o tratamientos automatizados.

g) Las medidas que sea necesario adoptar para el transporte de soportes y documentos, así como para la destrucción de los documentos y soportes, o en su caso, la reutilización de estos últimos.

4. En caso de que fueran de aplicación a los ficheros las medidas de seguridad de nivel medio o las medidas de seguridad de nivel alto, previstas en este título, el documento de seguridad deberá contener además:

a) La identificación del responsable o responsables de seguridad.

b) Los controles periódicos que se deban realizar para verificar el cumplimiento de lo dispuesto en el propio documento.

5. Cuando exista un tratamiento de datos por cuenta de terceros, el documento de seguridad deberá contener la identificación de los ficheros o tratamientos que se traten en concepto de encargado con referencia expresa al contrato o documento que regule las

condiciones del encargo, así como de la identificación del responsable y del período de vigencia del encargo.

6. En aquellos casos en los que datos personales de un fichero o tratamiento se incorporen y traten de modo exclusivo en los sistemas del encargado, el responsable deberá anotarlo en su documento de seguridad. Cuando tal circunstancia afectase a parte o a la totalidad de los ficheros o tratamientos del responsable, podrá delegarse en el encargado la llevanza del documento de seguridad, salvo en lo relativo a aquellos datos contenidos en recursos propios. Este hecho se indicará de modo expreso en el contrato celebrado al amparo del artículo 12 de la Ley Orgánica 15/1999, de 13 de diciembre, con especificación de los ficheros o tratamientos afectados.

En tal caso, se atenderá al documento de seguridad del encargado al efecto del cumplimiento de lo dispuesto por este reglamento.

7. El documento de seguridad deberá mantenerse en todo momento actualizado y será revisado siempre que se produzcan cambios relevantes en el sistema de información, en el sistema de tratamiento empleado, en su organización, en el contenido de la información incluida en los ficheros o tratamientos o, en su caso, como consecuencia de los controles periódicos realizados. En todo caso, se entenderá que un cambio es relevante cuando pueda repercutir en el cumplimiento de las medidas de seguridad implantadas.

8. El contenido del documento de seguridad deberá adecuarse, en todo momento, a las disposiciones vigentes en materia de seguridad de los datos de carácter personal.

Destacamos este artículo pues en él se recoge todo lo referente al documento de seguridad que hemos ido indicando en anteriores y posteriores artículos y que es aplicable a los dispositivos y soportes bajo la plataforma Android al poder contener datos de carácter personal y de nivel superior al básico, si bien el cumplimiento de este artículo depende de medidas administrativas y no de los aspectos técnicos del propio sistema operativo como tal.

CAPÍTULO III

Medidas de seguridad aplicables a ficheros y tratamientos automatizados

SECCIÓN 1.ª MEDIDAS DE SEGURIDAD DE NIVEL BÁSICO

Artículo 89. Funciones y obligaciones del personal.

1. Las funciones y obligaciones de cada uno de los usuarios o perfiles de usuarios con acceso a los datos de carácter personal y a los sistemas de información estarán claramente definidas y documentadas en el documento de seguridad.

También se definirán las funciones de control o autorizaciones delegadas por el responsable del fichero o tratamiento.

2. El responsable del fichero o tratamiento adoptará las medidas necesarias para que el personal conozca de una forma comprensible las normas de seguridad que afecten al desarrollo de sus funciones así como las consecuencias en que pudiera incurrir en caso de incumplimiento.

Destacamos este artículo puesto que representa las obligaciones del personal con respecto a los dispositivos bajo la plataforma que contienen datos de carácter personal, como hemos indicado en el artículo anterior referente al documento de seguridad, dichas obligaciones estarán especificadas en el documento.

Artículo 90. Registro de incidencias.

Deberá existir un procedimiento de notificación y gestión de las incidencias que afecten a los datos de carácter personal y establecer un registro en el que se haga constar el tipo de incidencia, el momento en que se ha producido, o en su caso, detectado, la persona que realiza la notificación, a quién se le comunica, los efectos que se hubieran derivado de la misma y las medidas correctoras aplicadas.

Destacamos el uso de algún tipo de registro por parte de los responsables de la entidad u organización a la hora de tener acceso a los dispositivos bajo la plataforma, puesto que estos últimos pueden contener datos de carácter personal almacenados, para llevar a cabo un control en caso de necesidad, como por ejemplo, en caso de extravío puedan resultar accesibles dichos datos por personal ajeno.

Artículo 91. Control de acceso.

1. Los usuarios tendrán acceso únicamente a aquellos recursos que precisen para el desarrollo de sus funciones.

2. El responsable del fichero se encargará de que exista una relación actualizada de usuarios y perfiles de usuarios, y los accesos autorizados para cada uno de ellos.

3. El responsable del fichero establecerá mecanismos para evitar que un usuario pueda acceder a recursos con derechos distintos de los autorizados.

4. Exclusivamente el personal autorizado para ello en el documento de seguridad podrá conceder, alterar o anular el acceso autorizado sobre los recursos, conforme a los criterios establecidos por el responsable del fichero.

5. En caso de que exista personal ajeno al responsable del fichero que tenga acceso a los recursos deberá estar sometido a las mismas condiciones y obligaciones de seguridad que el personal propio.

De este artículo destacamos el uso de un sistema de privilegios de usuario para el uso por distintos usuarios de los dispositivos bajo la plataforma y para el acceso a la información sensible almacenada en dichos dispositivos, por ejemplo mediante el sistema de autenticación.

También destacamos el punto 2 puesto que en una entidad u organización es recomendable que haya una lista o registro de los dispositivos y a quién han sido asignados, no es un control del sistema operativo pero sí necesario; como complemento a este último punto destacamos en el punto 5 de este artículo, que si dichos dispositivos fueran usados por colaboradores, becarios, subcontratados, proveedores o instaladores han de cumplir lo mencionado en el punto 2 del artículo.

Artículo 92. Gestión de soportes y documentos.

1. Los soportes y documentos que contengan datos de carácter personal deberán permitir identificar el tipo de información que contienen, ser inventariados y solo deberán ser accesibles por el personal autorizado para ello en el documento de seguridad.

Se exceptúan estas obligaciones cuando las características físicas del soporte imposibiliten su cumplimiento, quedando constancia motivada de ello en el documento de seguridad.

2. La salida de soportes y documentos que contengan datos de carácter personal, incluidos los comprendidos y/o anejos a un correo electrónico, fuera de los locales bajo el control del responsable del fichero o tratamiento deberá ser autorizada por el responsable del fichero o encontrarse debidamente autorizada en el documento de seguridad.

3. En el traslado de la documentación se adoptarán las medidas dirigidas a evitar la sustracción, pérdida o acceso indebido a la información durante su transporte.

4. Siempre que vaya a desecharse cualquier documento o soporte que contenga datos de carácter personal deberá procederse a su destrucción o borrado, mediante la adopción de medidas dirigidas a evitar el acceso a la información contenida en el mismo o su recuperación posterior.

5. La identificación de los soportes que contengan datos de carácter personal que la organización considerase especialmente sensibles se podrá realizar utilizando sistemas de etiquetado comprensibles y con significado que permitan a los usuarios con acceso autorizado a los citados soportes y documentos identificar su contenido, y que dificulten la identificación para el resto de personas.

Destacamos de este artículo el control administrativo a la hora de acceder a los dispositivos con información sensible mediante un registro de uso de estos últimos, dicho acceso a estos dispositivos debe ser autorizado por el responsable de los mismos; a la hora de desechar soportes que contienen información relevante se debe realizar un borrado o destrucción de los mismos que evite la posibilidad de recuperar dichos datos, así como un borrado de los datos contenidos en el propio dispositivo (para ello Android nos proporciona la opción de restaurar los valores de fábrica).

Artículo 93. Identificación y autenticación.

1. El responsable del fichero o tratamiento deberá adoptar las medidas que garanticen la correcta identificación y autenticación de los usuarios.

2. El responsable del fichero o tratamiento establecerá un mecanismo que permita la identificación de forma inequívoca y personalizada de todo aquel usuario que intente acceder al sistema de información y la verificación de que está autorizado.

3. Cuando el mecanismo de autenticación se base en la existencia de contraseñas existirá un procedimiento de asignación, distribución y almacenamiento que garantice su confidencialidad e integridad.

4. El documento de seguridad establecerá la periodicidad, que en ningún caso será superior a un año, con la que tienen que ser cambiadas las contraseñas que, mientras estén vigentes, se almacenarán de forma ininteligible.

Destacamos el uso de un mecanismo de autenticación mediante contraseñas y cuentas de usuario para realizar un control de los usuarios a los que se les permite acceder a la información almacenada en el sistema, y el mantenimiento y actualización de dicho mecanismo, como por ejemplo, el cambio periódico de las contraseñas de acceso.

Artículo 94. Copias de respaldo y recuperación.

1. Deberán establecerse procedimientos de actuación para la realización como mínimo semanal de copias de respaldo, salvo que en dicho período no se hubiera producido ninguna actualización de los datos.

2. Asimismo, se establecerán procedimientos para la recuperación de los datos que garanticen en todo momento su reconstrucción en el estado en que se encontraban al tiempo de producirse la pérdida o destrucción.

Únicamente, en el caso de que la pérdida o destrucción afectase a ficheros o tratamientos parcialmente automatizados, y siempre que la existencia de documentación permita alcanzar el objetivo al que se refiere el párrafo anterior, se deberá proceder a grabar manualmente los datos quedando constancia motivada de este hecho en el documento de seguridad.

3. El responsable del fichero se encargará de verificar cada seis meses la correcta definición, funcionamiento y aplicación de los procedimientos de realización de copias de respaldo y de recuperación de los datos.

4. Las pruebas anteriores a la implantación o modificación de los sistemas de información que traten ficheros con datos de carácter personal no se realizarán con datos reales, salvo que se asegure el nivel de seguridad correspondiente al tratamiento realizado y se anote su realización en el documento de seguridad.

Si está previsto realizar pruebas con datos reales, previamente deberá haberse realizado una copia de seguridad.

Destacamos de este artículo la utilización de un mecanismo de realización y recuperación de copias de respaldo del sistema, dicho mecanismo podría tratarse de una herramienta del sistema operativo Android, como por ejemplo, la sincronización de los datos relevantes del dispositivo con un PC o copias de seguridad de los datos en soportes de almacenamiento externo como pudieran ser tarjetas de memoria SD.

SECCIÓN 2.ª MEDIDAS DE SEGURIDAD DE NIVEL MEDIO

Artículo 96. Auditoría.

1. A partir del nivel medio, los sistemas de información e instalaciones de tratamiento y almacenamiento de datos se someterán, al menos cada dos años, a una auditoría interna o externa que verifique el cumplimiento del presente título.

Con carácter extraordinario deberá realizarse dicha auditoría siempre que se realicen modificaciones sustanciales en el sistema de información que puedan repercutir en el cumplimiento de las medidas de seguridad implantadas con el objeto de verificar la adaptación, adecuación y eficacia de las mismas. Esta auditoría inicia el cómputo de dos años señalado en el párrafo anterior.

2. El informe de auditoría deberá dictaminar sobre la adecuación de las medidas y controles a la Ley y su desarrollo reglamentario, identificar sus deficiencias y proponer las medidas correctoras o complementarias necesarias.

Deberá, igualmente, incluir los datos, hechos y observaciones en que se basen los dictámenes alcanzados y las recomendaciones propuestas.

3. Los informes de auditoría serán analizados por el responsable de seguridad competente, que elevará las conclusiones al responsable del fichero o tratamiento para que adopte las medidas correctoras adecuadas y quedarán a disposición de la Agencia Española de Protección de Datos o, en su caso, de las autoridades de control de las comunidades autónomas.

Destacamos de este artículo la necesidad de realizar cada cierto tiempo una auditoría interna o externa que incluya los dispositivos bajo la plataforma Android que estén destinados al uso laboral en cualquier organización, y que la clasificación de los datos que contengan dichos dispositivos sea de nivel medio o alto, como por ejemplo, datos de carácter clínico, currículos, etc...

Artículo 97. Gestión de soportes y documentos.

1. Deberá establecerse un sistema de registro de entrada de soportes que permita, directa o indirectamente, conocer el tipo de documento o soporte, la fecha y hora, el emisor, el número de documentos o soportes incluidos en el envío, el tipo de

información que contienen, la forma de envío y la persona responsable de la recepción que deberá estar debidamente autorizada.

2. Igualmente, se dispondrá de un sistema de registro de salida de soportes que permita, directa o indirectamente, conocer el tipo de documento o soporte, la fecha y hora, el destinatario, el número de documentos o soportes incluidos en el envío, el tipo de información que contienen, la forma de envío y la persona responsable de la entrega que deberá estar debidamente autorizada.

Destacamos de este artículo la utilización de una herramienta que permita el registro detallado de entradas y salidas de los dispositivos bajo la plataforma Android que contengan o puedan intercambiar con otros dispositivos información sensible, para su posterior seguimiento o revisión en caso de incidente, dichos datos almacenados en estos soportes deben ser de nivel superior al básico para que sean exigibles este tipo de controles, si bien el cumplimiento de este artículo depende de medidas administrativas y no de los aspectos técnicos del propio sistema operativo.

Artículo 98. Identificación y autenticación.

El responsable del fichero o tratamiento establecerá un mecanismo que limite la posibilidad de intentar reiteradamente el acceso no autorizado al sistema de información.

Destacamos en este artículo el uso del mecanismo de autenticación mencionado en artículos anteriores junto con la posibilidad de limitarlo a un número determinado de intentos de acceso.

Artículo 100. Registro de incidencias.

1. En el registro regulado en el artículo 90 deberán consignarse, además, los procedimientos realizados de recuperación de los datos, indicando la persona que ejecutó el proceso, los datos restaurados y, en su caso, qué datos ha sido necesario grabar manualmente en el proceso de recuperación.

2. Será necesaria la autorización del responsable del fichero para la ejecución de los procedimientos de recuperación de los datos.

Como destacamos en el artículo 90, el uso de algún tipo de registro por parte del responsable de los soportes a la hora de acceder a los datos personales almacenados en dispositivos bajo la plataforma, en este artículo, extendemos su uso añadiendo los datos relevantes en la recuperación de datos y copias de respaldo a dicho registro, si bien el cumplimiento de este artículo depende de medidas organizativas y no de los aspectos técnicos del propio sistema operativo.

SECCIÓN 3.ª MEDIDAS DE SEGURIDAD DE NIVEL ALTO

Artículo 101. Gestión y distribución de soportes.

1. La identificación de los soportes se deberá realizar utilizando sistemas de etiquetado comprensibles y con significado que permitan a los usuarios con acceso autorizado a los citados soportes y documentos identificar su contenido, y que dificulten la identificación para el resto de personas.

2. La distribución de los soportes que contengan datos de carácter personal se realizará cifrando dichos datos o bien utilizando otro mecanismo que garantice que dicha información no sea accesible o manipulada durante su transporte.

Asimismo, se cifrarán los datos que contengan los dispositivos portátiles cuando éstos se encuentren fuera de las instalaciones que están bajo el control del responsable del fichero.

3. Deberá evitarse el tratamiento de datos de carácter personal en dispositivos portátiles que no permitan su cifrado. En caso de que sea estrictamente necesario se hará constar motivadamente en el documento de seguridad y se adoptarán medidas que tengan en cuenta los riesgos de realizar tratamientos en entornos desprotegidos.

Destacamos de este artículo el uso de mecanismos criptográficos por parte del sistema operativo en lo que al tratamiento de información personal respecta, ya sea almacenamiento o intercambio de dicha información, dicha información criptografiada debe ser de nivel alto para que resulte exigible el cumplimiento del artículo.

Artículo 102. Copias de respaldo y recuperación.

Deberá conservarse una copia de respaldo de los datos y de los procedimientos de recuperación de los mismos en un lugar diferente de aquel en que se encuentren los equipos informáticos que los tratan, que deberá cumplir en todo caso las medidas de seguridad exigidas en este título, o utilizando elementos que garanticen la integridad y recuperación de la información, de forma que sea posible su recuperación.

En el caso de dispositivos con sistema operativo Android, estas copias de respaldo se debieran almacenar tanto en la memoria interna del dispositivo, como en soportes de almacenaje externo como pueden ser tarjetas de memoria SD, servidores de almacenamiento de datos o equipos con dispositivos sincronizados; evitando así la pérdida de datos en caso de fallo de alguna de las memorias anteriormente mencionadas.

Artículo 103. Registro de accesos.

1. De cada intento de acceso se guardarán, como mínimo, la identificación del usuario, la fecha y hora en que se realizó, el fichero accedido, el tipo de acceso y si ha sido autorizado o denegado.

2. En el caso de que el acceso haya sido autorizado, será preciso guardar la información que permita identificar el registro accedido.

3. Los mecanismos que permiten el registro de accesos estarán bajo el control directo del responsable de seguridad competente sin que deban permitir la desactivación ni la manipulación de los mismos.

4. El período mínimo de conservación de los datos registrados será de dos años.

5. El responsable de seguridad se encargará de revisar al menos una vez al mes la información de control registrada y elaborará un informe de las revisiones realizadas y los problemas detectados.

6. No será necesario el registro de accesos definido en este artículo en caso de que concurran las siguientes circunstancias:

a) Que el responsable del fichero o del tratamiento sea una persona física.

b) Que el responsable del fichero o del tratamiento garantice que únicamente él tiene acceso y trata los datos personales.

La concurrencia de las dos circunstancias a las que se refiere el apartado anterior deberá hacerse constar expresamente en el documento de seguridad.

Como hemos mencionado en artículos anteriores, el sistema operativo Android debiera implementar una herramienta a modo de "log" para el registro de accesos al sistema con información relevante a dichos accesos como puede ser la fecha y hora, el usuario, el tipo de acceso,...dicho registro debe realizarse cuando la información accedida sea de alto nivel.

Artículo 104. Telecomunicaciones.

Cuando, conforme al artículo 81.3 deban implantarse las medidas de seguridad de nivel alto, la transmisión de datos de carácter personal a través de redes públicas o redes inalámbricas de comunicaciones electrónicas se realizará cifrando dichos datos o bien utilizando cualquier otro mecanismo que garantice que la información no sea inteligible ni manipulada por terceros.

Destacamos en este artículo el uso de mecanismos de cifrado por parte del sistema operativo Android a la hora de tratar o realizar transmisiones de información de carácter personal en redes públicas o inalámbricas, para brindar una capa de protección extra a dicha información, aparte de la seguridad proporcionada por las compañías de comunicaciones.

Anexo B: Instalación entorno de desarrollo

1 Instalación de Eclipse con el SDK de Android

Para la realización de este anexo de nuestro proyecto fin de carrera, hemos recopilado y estudiado información de las siguientes referencias [23] y [24].

En este apartado se exponen los pasos necesarios para empezar a desarrollar aplicaciones Android. Las instrucciones de instalación aquí descritas se basan en el sistema operativo Windows XP Service Pack 3, y en el entorno de desarrollo Eclipse Galileo versión 3.5.2. Aunque esta guía de instalación no se contempla, el SDK de Android también puede ejecutarse en otros sistemas operativos como Mac OS X o Linux.

2 Descargar el SDK de Android

Android es una plataforma de software libre, por lo que cuenta con un SDK disponible para todo desarrollador que lo desee, incluye entre otros elementos, el conjunto completo de las API que el sistema operativo soporta. Para descargarlo, basta con visitar la web de Android [23] y acceder a la última versión publicada, en nuestro caso la última versión es la "revision 16" de Diciembre de 2011.

Una vez descargado el SDK, es necesario descomprimirlo. La ubicación de los ficheros resultantes no es relevante, pero conviene recordar la ruta para pasos posteriores.

3 Descargar Eclipse Galileo

La descarga de Eclipse no es muy diferente al SDK de Android. La web de Eclipse [24] ofrece multitud de versiones de este entorno de desarrollo según las necesidades del desarrollador. En este caso, es suficiente con obtener la versión 3.5.2, denominada Galileo.

Finalizada la descarga, no se realiza ningún proceso de instalación; simplemente se debe descomprimir los ficheros y pulsar el ejecutable para abrir la aplicación. La primera vez que se inicie Eclipse, pide al usuario una localización para el *workspace*, donde se ubicarán por defecto todos los proyectos desarrollados en este entorno de trabajo.

4 Instalar el plug-in de Android ADT

El siguiente paso consisten en instalar el *plug-in* específico de Android para la plataforma Eclipse. Esta herramienta, denominada ADT (Android Development Tools), facilita enormemente la creación de proyectos, su implementación, depuración y ejecución, por lo que es altamente recomendable si se quiere trabajar con Android.

Para instalar el *plug-in* ADT en Eclipse Galileo, es necesario seguir las siguientes indicaciones:

1. Iniciar Eclipse

2. Seleccionar la pestaña *Help > Software Updates*. Esta acción abrirá una nueva ventana llamada *Software Updates and Add-ons*.

3. Pinchar en la pestaña *Available Software* y pulsar el botón *Add Site*.

4. Introducir la siguiente URL y pulsar *OK*:

https://dl-ssl.google.com/android/eclipse/

5. Volviendo a la ventana *Software Updates and Add-ons*, marcar la casilla correspondiente a *Developer Tools* y pulsar el botón *Install*. Se abrirá una nueva ventana.

6. Cerciorarse de que las opciones *Android Developer Tools* y *Android Editors* están marcadas y pulsar el botón *Finish*.

El proceso de instalación dará comienzo y puede llevar algunos minutos. Con el fin de que los cambios tengan efecto, es necesario reiniciar Eclipse.

5 Referenciar el SDK de Android

Tras abrir de nuevo Eclipse, debe indicarse en las preferencias de Eclipse la localización del SDK a utilizar para los proyectos de Android:

1. Seleccionar la pestaña *Window > Preferences*, lo que abrirá una nueva ventana.

2. Elegir *Android* en el panel izquierdo.

3. Pulsar el botón *Browse* e indicar la ruta del SDK de Android.

4. Pulsar el botón *Apply* y después *OK*.

6 Actualizaciones del plug-in ADT

Es posible que con el tiempo haya disponible una nueva versión del ADT correspondiente a la versión del SDK de Android instalado. Para comprobar las posibles actualizaciones a través de Eclipse, se debe realizar lo siguiente:

1. Iniciar Eclipse.

2. Seleccionar la pestaña *Help > Software Updates*. Esta acción abrirá una nueva ventana llamada *Software Updates and Add-ons*.

3. Elegir la pestaña *Installed Software*.

4. Pulsar el botón *Update*.

5. Si existe alguna actualización para el ADT, seleccionarla y pulsar el botón

Finish.