Eludiendo la concesión de permisos de administrador en Android mediante una vulnerabilidad en SuperUser

Patxi Galán-García
DeustoTech Computing
Universidad de Deusto
patxigg@deusto.es

Borja Sanz Urquijo
DeustoTech Computing
Universidad de Deusto
borja.sanz@deusto.es

Carlos Laorden Gómez
DeustoTech Computing
Universidad de Deusto
claorden@deusto.es

Pablo García Bringas
DeustoTech Computing
Universidad de Deusto
pablo.garcia.bringas@deusto.es

***Abstract*— Todo el mundo hoy en día esta conectado de alguna manera a Internet, y cada vez más conexiones se hacen desde los nuevos teléfonos inteligentes o *smartphones*. Estos teléfonos son potentes ordenadores en los que se almacenan contactos, contraseñas, fotos, datos de empresa, cuentas bancarias, perfiles de redes sociales, y demás información sensible. Esta información tiene un alto valor económico y comercial ya que, con ella, se puede enviar publicidad personalizada, robar dinero, chantajear, espiar, etc. También nos permiten personalizarlos a nuestro gusto, incluso nos permiten cambiar el sistema original por otro totalmente diferente. Para llegar a este punto de personalización, hace falta saltarse la seguridad de la que disponen y acceder a un usuario administrador, (hacerse 'root'). Este usuario tiene permisos para hacer todo lo que deseé. El problema radica en que, a la hora de acceder a ese usuario, se debe tener otro sistema de seguridad para controlar los accesos a este usuario. Los sistemas de este tipo que existen actualmente, son efectivos siempre y cuando no se haya permitido previamente a otra aplicación maliciosa acceder a los permisos de administrador. En este documento mostraremos un mecanismo capaz de saltarse ese tipo de seguridad.**

Keywords: **Android, seguridad, permisos, SuperUser, root**

I. Introducción

En los últimos años se ha expandido el uso de *smartphones* o teléfonos inteligentes [1]. Estos dispositivos han tenido una muy buena acogida dada su versatilidad y su potencia. Según un informe de la Comisión del Mercado de las Telecomunicaciones, [2]. el segundo factor que argumentan los usuarios para cambiar de compañía, por detrás del de abaratar su factura, es el cambio de terminal móvil bajo promoción, con un 35,7%, y muy por delante de insatisfacción por la calidad, que motiva al 23,3%.

Las características técnicas como el procesamiento y los sensores para obtener la posición [3] y para comunicarse con elementos cercanos [4] y poder pagar [5] con ellos, combinados con la conectividad permanente a Internet, ha logrado que el número de líneas móviles en España supere en número a las linea de abonado digital asimétrica (ADSL) [2]. Actualmente se activan al día 700.000 dispositivos Android [6], y China ha superado a EEUU [7] en activaciones de este tipo de dispositivos.

Además, el éxito y la naturaleza libre de Android, está propiciando la creación de numerosas comunidades para desarrollar versiones basadas en este sistema. Entre las comunidades más importantes por volumen de usuarios, se encuentran *CyanogenMod* [8] (con un número total de instalaciones cercano a las 2.000.000 [1]) o *MIUI* [9].

Estos sistemas personalizados son imágenes del sistema operativo Android que se instalan en los dispositivos. Estas versiones suelen estar modificadas por programadores o por los mismos fabricantes del dispositivo con el fin de aportar mejoras al sistema original, bien sea modificando las aplicaciones nativas de Android o introduciendo nuevas. Se denominan *ROMs* o *ROM cocinadas*.

Gracias a estas *ROMs*, está creciendo el interés por conseguir permisos de administrador en los dispositivos o, como se llama en las comunidades, por *rootear* [10] (en Android) o *jailbreak* [11] (en iOS[2]) para tener control total de los mismos. Todas las *ROMs cocinadas* requieren y deben tener, acceso a *root* [12], ya que si no, no podrían estar instaladas. Cabe mencionar que estos dispositivos, por defecto, no traen acceso al usuario administrador o *root*.

Este tipo de técnicas también se aplican a otros sistemas móviles como *Blackberry*[3] de RIM.

La denominación de *jailbreak* se popularizo gracias a la familia de dispositivos Apple. El *software* de estos dispositivos es de carácter privativo (es decir, el código fuente no esta disponible), por lo que, en teoría, es más complicado encontrar vulnerabilidades. Aún así, existen múltiples métodos para lograr permisos de administrador. Existen también, comunidades que se dedican exclusivamente a encontrar este tipo de vulnerabilidades y dar acceso total a los usuarios. El ejemplo más extendido es el de *jailbreakme*[4] que cada vez que sale una nueva versión de este *software*, hasta que logren encontrar una vulnerabilidad para lograr el acceso a *root*.

En la plataforma de RIM[5], que también es de carácter privativo pero más orientado a las empresas, solo existen

[1] http://stats.cyanogenmod.com/
[2] http://www.apple.com/es/ios/
[3] http://us.blackberry.com/apps-software/blackberry7/
[4] http://www.jailbreakme.com/
[5] http://us.blackberry.com/

indicios de dispositivos *rooteados*. Es el caso de la 'tablet' *Blackberry PlayBook*. Aunque la noticia tuvo gran interés, RIM emitió un comunicado diciendo que no había tenido noticias al respecto [13]. Además, los teléfonos de RIM, que son los más extendidos de esta plataforma, no han sido vulnerados por este tipo de acciones hasta la fecha.

Nosotros nos vamos a centrar en los sistemas Android, por lo que usaremos la denominación *rootear* [10] para hablar del proceso de obtención de permisos de administrador. Aunque el significado y las opciones son muy parecidos en todos los sistemas.

Cabe mencionar que, aunque el sistema de seguridad de Android es robusto, debido a su naturaleza *open source* y a que las aplicaciones se distribuyen por un medio en el cual hay millones de aplicaciones, el *Android Market* no esta exento de fallos ni de aplicaciones maliciosas.

Este trabajo trata de mostrar un ataque a la aplicación SuperAgent, basado en un tipo de vulnerabilidad y en el uso de ingeniería social para lograr acceso completo al sistema Android.

El resto de apartados se ordenan de la siguiente manera. En el punto 2, se explicará la arquitectura de Android, centrándonos principalmente en los permisos y en la máquina *Dalvik*. En el punto 3 detallaremos el proceso para eludir la aplicación SuperUser y finalizaremos con las conclusiones.

II. La arquitectura de Android

A la hora de plantear la seguridad de la plataforma, uno de los elementos que tuvieron en cuenta es que los desarrolladores de aplicaciones que poblarían el ecosistema Android no entienden de seguridad. Esta premisa, que viene a mostrar la escasa preocupación por la seguridad esperada, se tuvo muy en cuenta a la hora de plantear el modelo de desarrollo de la plataforma.

A continuación vamos a ver algunos de los aspectos más relevantes de seguridad a la hora de desarrollar aplicaciones, por ser éstas uno de los puntos en los que los terminales son más vulnerables.

A. Control de la seguridad en aplicaciones

Son dos los elementos que permiten este control: los permisos y un sistema de máquina virtual denominado *Dalvik*.

Los permisos son necesarios para interactuar con la *API* del sistema, los datos y el sistema de paso de mensajes.

La *API* del sistema [14], describe 8.648 métodos distintos, de los cuales sólo algunos de ellos necesitan permisos específicos para poder acceder a ellos. Para acceder a los lugares donde se almacenan los datos del sistema, denominados *Content providers*, se requieren permisos específicos (e.j., para poder acceder a los contactos almacenados es necesario poseer el permiso `READ_CONTACTS`).

Por otro lado, la ejecución de las aplicaciones se realiza dentro de un sistema de máquina virtual denominado *Dalvik*. Cada una de las aplicaciones se ejecuta dentro de su propia máquina virtual con un único usuario del sistema (`UID`), lo que evita que éstas puedan acceder a recursos de otras aplicaciones.

Esta máquina virtual permite la ejecución de las aplicaciones en un entorno controlado, favoreciendo la supervisión de las aplicaciones.

Con este tipo de medidas, se controla desde el diseño la ejecución de las aplicaciones, la fase de instalación y el propio uso. De esta manera se conforma una arquitectura segura de la plataforma.

B. Permisos de las aplicaciones

El sistema operativo Android otorga un usuario del sistema (denominado `uid`) a cada una de las aplicaciones que se instalan en el sistema. Estos permisos son otorgados tanto a los ejecutables como a los recursos de los que hace uso la aplicación. Partiendo de este punto, se otorgan los permisos necesarios a las aplicaciones para su ejecución.

En lo que respecta a las aplicaciones, existen dos tipos generales de permisos:

- *En tiempo de ejecución*: estos permisos son aprobados por el usuario cuando se accede a algún recurso del sistema que pudiera considerarse sensible (p.ej., acceso al GPS). Este modelo es el que se implementa en los sistemas iOS de Apple.
- *En tiempo de instalación*: estos permisos los concede el usuario a la aplicación en el momento de la instalación.

El sistema operativo Android se caracteriza por dicho sistema de permisos en tiempo de instalación, el cual otorga una serie de ventajas [15]:

- Consentimiento expreso del usuario.
- Defensa en profundidad, ya que define el máximo nivel de privilegio necesario para ejecutar la aplicación. Gracias a ello, un atacante debe buscar fallos de seguridad en la plataforma para llevar a cabo tareas que se encuentren fuera del ámbito de la aplicación.
- Revisión de triaje, i.e., selección y clasificación de aplicaciones basándose en prioridades. Esto facilita la labor de los revisores ya que pueden centrarse en las aplicaciones que requieren permisos que pueden conllevar un mayor riesgo.

Los permisos de Android poseen además dos características adicionales importantes. La primera de ellas es que los permisos de instalación están clasificados en cuatro niveles: *normal*, *peligroso*, *firmado*, *firma-o-sistema*. Los permisos normales son aceptados por defecto. Estos permisos son otorgados para proveer al sistema con una defensa en profundidad y revisión de triaje. Los permisos de firma permiten a los desarrolladores controlar los permisos que se proveen a otras aplicaciones a través de las interfaces exportadas. Para ello, deben estar firmados con la misma firma de desarrollador. Por último, los permisos que se encuentran en la categoría firma-o-sistema son aquellos que están firmados con la clave de *firmware*. Este grupo pretende evitar qué aplicaciones de terceros hagan uso de las funciones del sistema. De todos ellos, únicamente los permisos que se solicitan dentro del nivel peligroso se presentan al usuario.

Fig. 1. Esquema del funcionamiento de Android. Cada aplicación se ejecuta en su propia máquina virtual, denominada Dalvik

C. La máquina virtual Dalvik

La palabra *rootear* [10] tiene su origen en la palabra inglesa *root*, y es heredada de los sistemas UNIX. Un sistema basado en Linux/UNIX normalmente es multiusuario, es decir, muchos usuarios pueden estar interactuando con un mismo sistema usando configuraciones propias y sin que ningún usuario estorbe a otro. En este caso, cada aplicación es un usuario diferente que se ejecuta sobre la maquina virtual Dalvik (ver Fig.1) y tiene sus propios privilegios dados por los permisos de la aplicación.

Entre todos los usuarios hay uno especial que es el administrador del sistema y, más conocido como usuario *root*. Este usuario tiene capacidades para administrar el sistema a su antojo, tiene plenos privilegios para gestionar y administrar el sistema, las actualizaciones, los usuarios, los grupos, los recursos, etc. sin necesidad de confirmaciones. Lo equivalente en los sistemas *Windows* es la cuenta de administrador.

En los dispositivos portátiles que poseen base *UNIX*, como son los *smartphones* o las *tablets*, al *rootearlos* se consigue acceso a este usuario y, por consiguiente, la capacidad para administrar todos los aspectos posibles del dispositivo. Mediante este método, el usuario del *smartphone* logra obtener un control total y absoluto de su dispositivo para hacer con él lo que le plazca. Algunas de las circunstancias más comunes para *rootear* [10] el terminal suelen ser el realizar tareas especificas tales como, instalar aplicaciones especiales o realizar acciones que no permite el sistema operativo por defecto, (e.j. la edición de interfaces de usuario, el control sobre el consumo de recursos y de la batería en concreto, cambiar el sistema operativo por uno *customizado* por la comunidad). Las posibilidades que ofrece son muchas, pero a la vez, los riesgos que conlleva también. Todo poder conlleva una gran responsabilidad.

De hecho, estas manipulaciones tienen su lado perverso. La opción de poder hacer lo que se desee en un terminal, en ocasiones, es muy seductora. Al igual que existen herramientas muy potentes para hacer determinadas tareas benignas, también existen otras herramientas, por desgracia muchas más, que justamente hacen todo lo contrario.

Es por esto, que la idea de los permisos en el sistema

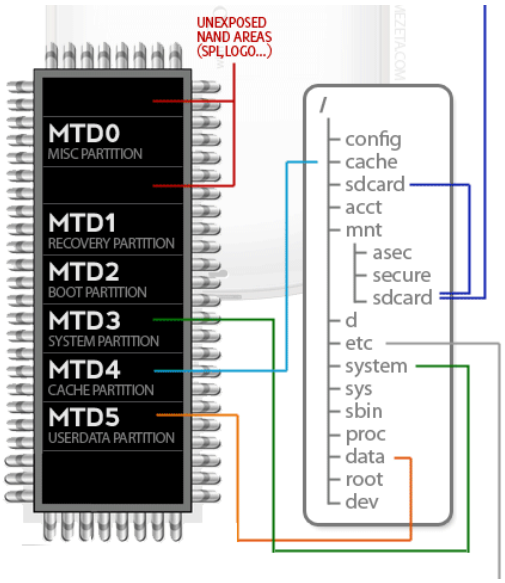

Fig. 2. Esquema de las particiones del sistema operativo Android y localización física en memoria

Android [16] es tan necesaria, siendo de hecho, un pilar fundamental [17]. Los permisos están pensados para que un usuario sepa a qué recursos va a acceder dicha aplicación y además, si es posible que deba pagar por ello [18].

En un sistema *rooteado* estos permisos desaparecen. Una aplicación que desee tener acceso a la memoria, a los contactos o a cualquier otro recurso, simplemente con ejecutar el comando en terminal con el binario *su* lo logra, y además si no hay algún tipo de sistema que alerte de esta acción, el usuario final ni tan siquiera se entera.

El sistema operativo Android en los dispositivos, está dividido en particiones de la memoria NAND (ver Fig.2) con particiones diferenciadas y relacionadas entre si. El directorio en el cual se almacenan los binarios del sistema es */system*, donde se almacena la información del usuario es */data* y la información de cada aplicación se almacena en */data/data* creando una estructura de archivos que tiene como raíz el nombre del paquete de la aplicación. Existe un binario llamado *su* capaz de otorgar privilegios de administrador a las aplicaciones que lo soliciten. Como previamente se ha mencionado en el apartado I, este binario no viene instalado por defecto en los dispositivos que se comercializan. Para lograr tener este binario en nuestro sistema, se usan mecanismos que se saltan la seguridad del dispositivo y acceden a carpetas restringidas en las cuales, se instala.

En este punto es donde entran herramientas de control de permisos como *SuperUser*[6]. Este tipo de aplicaciones gestionan, mediante sistemas en continua ejecución, el acceso a dicho ejecutable. Sirven como alarma para avisarnos cuando alguna aplicación requiere acceso a *su*, deteniendo su ejecución hasta recibir nuestro beneplácito o, en caso contrario, bloqueando su acceso al binario *su* impidiendo que termine

[6]http://androidsu.com/superuser/

su ejecución.

Todas las *ROM's* poseen este tipo de programas. Esto se debe a que es una de las maneras más extendidas para controlar el acceso a un elemento tan sensible del sistema como es *su*. Los usuarios, normalmente, no suelen instalar este tipo de aplicaciones, ya que, como antes se ha dicho, ya vienen instaladas en las *ROM's*.

Recientemente, en la *ROM* de *CyanogenMod9* se ha introducido un elemento de seguridad extra que permite al usuario, desde el propio núcleo del sistema o Kernel, decidir qué puede tener acceso a *root* [19]. Pero este sistema no avisa de los accesos ni bloquea accesos especificos, por lo que no es un sistema totalmente fiable que pueda sustituir al *SuperUser* totalmente.

Para el desarrollo de este trabajo, nos hemos centrado en atacar la herramienta *SuperUser*, por ser la más extendida y usada por múltiples *ROMs cocinadas*.

III. Eludir SuperUser

Antes de entrar en materia, mencionar que esta técnica funciona para las versiones de SuperAgent hasta la versión 3.0.7(41). También mencionar que es necesario engañar al usuario de alguna manera, para que este permita el primer acceso. Una vez logrado este primer acceso, esta técnica permite a nuestra aplicación dar permisos de *root* a si misma de forma permanente y a otras aplicaciones, de manera permanente o temporal.

Terminada esta breve explicación, entramos a detallar los pasos realizados para esta investigación.

El experimento partió de la base de que «una aplicación de terceros no debe administrar el acceso a un apartado tan importante en un sistema tan complejo como es el acceso al usuario administrador». Esto se debe a que dicha aplicación, al no estar dentro del *Kernel*, se podía sortear o saltar de alguna manera con un simple acceso a *root*.

El primer paso para poder eludir una aplicación fue obtener su código fuente. En este caso se descompiló el archivo *clases.dex* dentro del *.apk* y se analizó su funcionamiento.

La aplicación funciona en base a servicios y elementos que permanecen a la escucha constante de peticiones al binario *su* que esta en `/system/bin` o `/system/xbin` (estas carpetas tienen la misma finalidad). Cada vez que una aplicación solicita acceso a dicho binario, el *receiver* o sistema a la escucha del *SuperUser*, activa un mecanismo de notificación para informarnos de que una aplicación, en este caso la activa, requiere acceso al usuario administrado mostrándonos una ventana de opciones para que el usuario decida que hacer. La información del resultado de la acción del usuario (aceptar o rechazar el acceso) se almacena en `/databases` de la carpeta de *SuperUser* que la aplicación generaba en el sistema en `/data/data`. Este sistema de permisos se basa en dos tablas *sqlite3* principales. Estas tablas son las que poseen la información sobre si a dicha aplicación que solicita acceso a *su* ya se le han concedido permisos de acceso o denegación y, si estos accesos son permanentes. Mencionar que la información dentro de estas bases de datos no ésta cifrada de ninguna manera y se puede acceder fácilmente.

Tabla I
Elementos de la tabla *su*

Campo	Descripción
uid	Identificador del usuario/aplicación
package	Nombre del paquete
name	Nombre del programa
allow	Define la ejecución de la app. 1 : Permiso permanente como root' 0 : Permiso temporal como root' -1 : Denegación de permiso como root'

En el código de la *apk* también se han encontrado elementos *PIN* para que los *recivers* obtengan los datos específicos relacionados con la aplicación y no se equivoquen con otras llamadas. También se han encontrado los métodos para crear las bases de datos *sqlite3*.

El sistema que hemos desarrollado para eludir el control del *SuperUser* consiste en una aplicación que solicita un primer acceso al usuario *root* para poder obtener acceso a dichas bases de datos y, de esta manera, poder saltarse los sistemas de seguridad, el aislamiento de cada aplicación y los permisos en Android. La información que buscábamos se almacena en `/data/data/com.noshufou.android.su/databases`.

Una vez logrado el primer acceso, se recuperan los datos temporales que el sistema *SuperUser* ha concedido a nuestra aplicación. Estos datos sirven para conseguir permisos permanentes, en caso de que el usuario no los hubiera concedido. La consulta a ejecutar es `Select _id,uid,package,name,exec_uid,exec_cmd, allow from apps where apps.name = "SqliteEjemplo"`. Esta información constituye entre otros elementos el uid, paquete, nombre y si está permitido temporalmente (-1) o permanentemente (1) (véase Tabla. I) de la aplicación que ha solicitado el acceso a *su*. Seguidamente, accedemos a la tabla de permisos localizada en `/data/data/com.noshufou.android.su/databases` e insertamos los datos de nuestra aplicación (son los datos temporales que se nos han concedido previamente), marcándola como permitida (valor $allow = 1$) para que no sea necesaria una comprobación de permisos. La consulta a ejecutar es `INSERT INTO apps (_id,uid,package,name,exec_uid,exec_cmd, allow) VALUES` donde los VALUES son los datos obtenidos de la anterior consulta.

Esto es posible hacerlo sin ningún tipo de permisos de Android gracias a que tenemos permisos de *root*.

En `/data/data/com.noshufou.android.su/databases` existen varias tablas, pero las más importantes son *su.db* que sirve para mostrar al usuario las aplicaciones que han sido registradas por *SuperUser* y *permissions.sqlite*, esta la que realmente almacena los permisos de negación o aceptación concedidos a cada aplicación.

Por otra parte, la actualización para mostrar al usuario los permisos concedidos a cada aplicación, depende de la base de

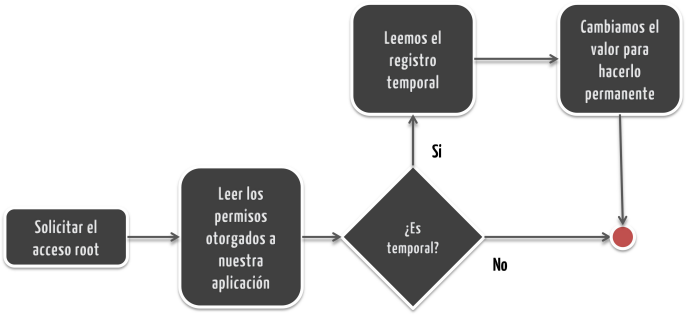

Fig. 3. Organigrama del ataque planteado

datos *su.db*, y es opcional. Cuando no se quiera mostrar a que aplicación se le han concedido permisos para ejecutarse como *root* no se deberá actualizar esta base de datos.

De esta manera, la aplicación desarrollada tendrá garantizado el acceso a *root* en las próximas ejecuciones sin necesidad, ni de avisar al usuario, ni que tenga que permitírselo, ni que lo tenga que saber.

La única información que el usuario va a recibir, va a ser un *Toast* o *popup* informando de la concesión de permisos de *root* a una aplicación, pero tampoco dice exactamente a cual.

En la Figura 3) se presenta un acercamiento al código final desarrollado.

Los pasos son:

1) Solicitar el acceso a *root* para no depender de los permisos propios
2) Leer los permisos específicos concedidos a la aplicación
3) Verificar si son temporales o permanentes
4) Si son temporales, modificar esa temporalidad para hacerlos permanentes

Lo que hemos logrado es obtener acceso total al terminal, pudiendo instalar, borrar, copiar, etc., todo lo que deseemos. Además, debido a que ya tenemos permisos de *root*, podemos hacer que cualquier aplicación que deseemos tenga también acceso a este nivel de seguridad.

Esto permite que, aplicaciones potencialmente perniciosas, que antes estaban controladas por el sistema de gestión de permisos, tengan total libertad de actuación en nuestros terminales. De hecho, ya han empezado a aparecer aplicaciones que colaboran entre si con fines maliciosos [20].

IV. Conclusión

La gran acogida experimentada por los *smartphones* a supuesto que estos nuevos dispositivos se vean cada vez más presentes en nuestra sociedad. Entre los sistemas operativos que potencian a estos teléfonos inteligentes, Android parece haberse convertido en el más popular de todos ellos.

La seguridad primaria en forma de permisos con la que cuenta Android, es un sistema robusto, siempre y cuando se lean los permisos que necesita y no se acepten sin más. Aún así, se puede ver comprometida por una mala practica del eslabón más débil de la cadena de seguridad, el usuario.

La facilidad y las amplias posibilidades de personalización de estos dispositivos, han aumentado el deseo por conseguir un control total del teléfono, mediante el acceso al usuario administrador, acabando de este modo con el sistema de permisos de Andorid.

Para intentar mantener estas medidas lo más intactas posibles, existen herramientas como SuperUser, que velan por el acceso al usuario administrador. Esta herramienta es, actualmente, de las mejores aplicaciones para este tipo de situación y, por ello, la mayoría de las *ROMs* la traen instalada por defecto. Pero, como todas las herramientas desarrolladas por el hombre, no es perfecta ni infalible, aunque si muy efectiva.

En este trabajo hemos construido una aplicación capaz de evitar este tipo de herramientas de control de acceso a *root*. De este modo, hemos mostrado la posibilidad de que aplicaciones maliciosas puedan lograr acceso total al sistema, permitiendo ejecutar cualquier comando en el terminal.

References

[1] A. Fumero, "Introducción: la red en el móvil," *Telos: Cuadernos de comunicación e innovación*, no. 83, pp. 43–49, 2010.

[2] C. del Mercado de las Telecomunicaciones (CMT), "Informe sector - comunicaciones moviles." [Online]. Available: http://informeanual.cmt.es/docs/1.4INFORMESECTOR-COMUNICACIONESMOVILES.pdf

[3] B. Hofmann-Wellenhof, H. Lichtenegger, and J. Collins, "Global positioning system. theory and practice." *Global Positioning System. Theory and practice., by Hofmann-Wellenhof, B.; Lichtenegger, H.; Collins, J.. Springer, Wien (Austria), 1993, 347 p., ISBN 3-211-82477-4, Price DM 79.00. ISBN 0-387-82477-4 (USA).*, vol. 1, 1993.

[4] K. Finkenzeller *et al.*, *RFID handbook: Fundamentals and applications in contactless smart cards, radio frequency identification and near-field communication*. Wiley, 2010.

[5] J. Ondrus and Y. Pigneur, "An assessment of nfc for future mobile payment systems," in *Management of Mobile Business, 2007. ICMB 2007. International Conference on the*. IEEE, 2007, pp. 43–43.

[6] "700.000 activaciones diarias, analizamos el galaxy nexus, la invasión android," http://www.xatakamovil.com/xatakamovil/700000-activaciones-diarias-analizamos-el-galaxy-nexus-la-invasion-android.

[7] "China sobrepasa a eeuu en activaciones android e ios." [Online]. Available: http://www.xatakandroid.com/mercado/china-sobrepasa-a-eeuu-en-activaciones-android-e-ios

[8] cyanogenmod, "Cyanogen mod." [Online]. Available: http://www.cyanogenmod.com/

[9] miui, "Miui." [Online]. Available: http://miui.es/

[10] A. Nazar, M. Seeger, and H. Baier, "Rooting android–extending the adb by an auto-connecting wifi-accessible service."

[11] S. Esser, "Exploiting the ios kernel," 2011.

[12] L. I. Proyect, "What is root?" [Online]. Available: http://www.linfo.org/root.html

[13] "Declaración de rim sobre la noticia del root en el blackberry playbook," http://lablackberry.com/2011/12/declaracion-de-rim-sobre-la-noticia-del-root-en-el-blackberry-playbook.html. [Online]. Available: http://lablackberry.com/2011/12/declaracion-de-rim-sobre-la-noticia-del-root-en-el-blackberry-playbook.html

[14] A. Team, "Android developers references," Google, Tech. Rep., 2011. [Online]. Available: http://developer.android.com/reference/

[15] A. Felt, K. Greenwood, and D. Wagner, "The effectiveness of application permissions," in *2nd USENIX Conference on Web Application Development*, 2011, p. 75.

[16] M. Nauman, S. Khan, and X. Zhang, "Apex: Extending android permission model and enforcement with user-defined runtime constraints," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ACM, 2010, pp. 328–332.

[17] J. Tomás Gironés, "Los permisos en android," 2011.

[18] J. Tomás Gironés, "Seguridad en android," 2011.

[19] "Security and you." [Online]. Available: http://www.cyanogenmod.com/blog/security-and-you

[20] “Security alert: New android Malware—OsSpy—Works as a malicious team | NQ mobile U.S. security research center,” http://research.nq.com/?p=464. [Online]. Available: http://research.nq.com/?p=464