Análisis y explotación de vulnerabilidades en Android

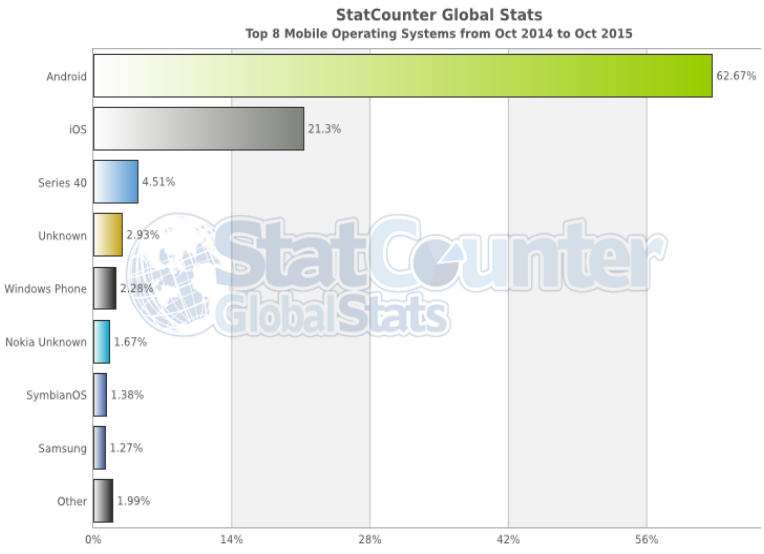
Christian González León, Instituto Politécnico Nacional, Noviembre de 2015

***Abstract* — In recent years, the amount of malware in Smartphones has increased rapidly, mainly because the complexity of maintaining modern operating systems managing these devices. This paper aims to provide information about security issues caused by implementation designs of some Android's modules, which end in exploitable vulnerabilities that could affect directly to the security of the end users.**

***Palabras clave* — Smartphones, Android, Vulnerabilidad, Malware, Seguridad.**

I. Introducción

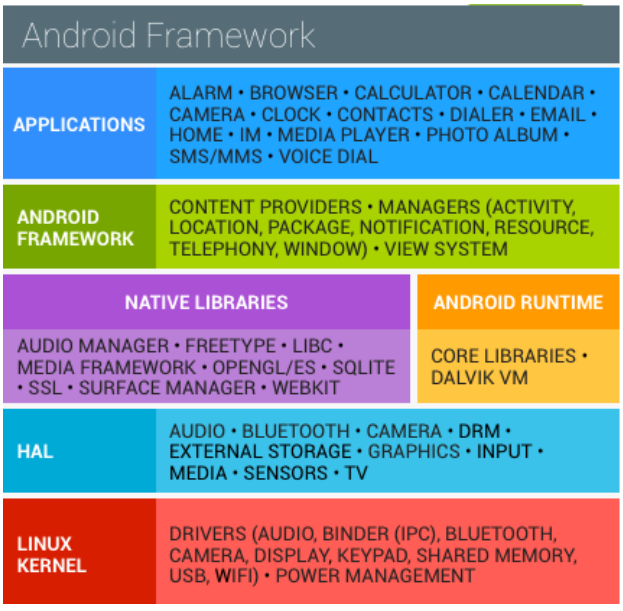
Actualmente, el mercado de los teléfonos inteligentes es basto. Solo en México, de 103.9 millones de líneas móviles en uso al cierre de 2014, 52.6 millones corresponden a los usuarios que poseen un teléfono inteligente, es decir, un 50.7% del total [1]. Además, el sistema operativo más utilizado en el mundo es Android, con un dominio claro del 62.67% para octubre de 2015, tal como se muestra en la siguiente gráfica obtenida de [2].



Las estadísticas anteriores nos hablan de un riesgo importante a la seguridad de los usuarios de los teléfonos inteligentes y especialmente a aquellos usando el sistema operativo Android, razón por la cual éste ha sido elegido como objeto de estudio. La historia ha demostrado que aquellos sistemas operativos con amplio rango de usuarios son los más propensos a recibir ataques por parte de los cibercriminales, un ejemplo es el famoso malware "ILOVEYOU" que en el año 2000 infectó aproximadamente 50 millones de computadoras corriendo Windows, provocando pérdidas millonarias [3].

II. Arquitectura del SO Android

Android es un sistema operativo basado en Linux y al igual que éste, sus diferentes componentes están separados en módulos apilados en capas, donde los niveles más bajos son los más críticos en cuanto a la seguridad se refiere, ya que la seguridad de las capas superiores depende de la de las capas inferiores. En la siguiente figura tomada de [4] se muestran las capas que conforman la arquitectura del sistema Android, junto con los componentes de cada una de ellas.



III. Modelo de seguridad en Android

Las apps que todo usuario de Android utiliza en sus terminales son desarrolladas utilizando el lenguaje de programación *Java* y el *framework* de desarrollo provisto por Google Inc [5]. Al final, una app es distribuida en un paquete (archivo *APK*) que contiene el código y los recursos requeridos por ella para ser ejecutada [6]. Además, al iniciar una app se crea una nueva instancia de la máquina virtual *Dalvik* dedicada específicamente a ejecutar el código de dicha app.

El primer aspecto relevante en la seguridad en Android es el sistema de permisos heredado de Linux y conocido como la *Sandbox* de Android [7]. Este sistema limita en gran medida el acceso al sistema de archivos e impide que los procesos puedan acceder a los recursos de otros procesos, como la memoria y la CPU. Sin embargo, existen procesos con privilegios elevados que pueden acceder a recursos del dispositivo sin problemas, principalmente servicios del sistema y aplicaciones preinstaladas en los dispositivos.

En adición a lo anterior, las aplicaciones que son ejecutadas por *Dalvik* deben manifestar los permisos de los que van a disponer en un archivo dentro de su *APK* llamado *AndroidManifest.xml* [8]. Estos permisos definen los recursos del dispositivo que van a utilizar y que el usuario debe aceptar antes de instalarlas, como el acceso a la cámara, leer o escribir en el almacenamiento externo, el acceso a internet, etc. Android impide que las apps utilicen recursos no manifestados, provocando un fallo en las mismas o su cierre, aunque esto solo sucede durante el desarrollo de las aplicaciones y no en su distribución.

IV. Concepto de Vulnerabilidad

Las vulnerabilidades son errores de programación cometidos durante el desarrollo de un software que pasan desapercibidos al programador y que inicialmente no representan un problema desde el punto de vista del usuario final, pero que los cibercriminales son capaces de detectar y que potencialmente permiten realizar un sin número de actividades no deseadas en los sistemas operativos corriendo estos softwares vulnerables; como acceder al sistema de archivos, alterar el comportamiento normal del sistema y en el caso más grave controlar el dispositivo atacado remotamente.

Pero las vulnerabilidades no solo se manifiestan por errores de programación, en algunos casos el propio diseño de estos programas es la causa de ellas. A veces simplemente no se consideró la validación de un dato o archivo de entrada y que al ser procesado, genera un comportamiento inesperado que termina siendo una vulnerabilidad explotable[a].

V. Vulnerabilidades comunes en Software

A continuación se enlistan y explican a groso modo algunas vulnerabilidades comunes encontradas en los sistemas informáticos que pueden llegar a afectar, y en algunos casos han afectado a Android, como a cualquier sistema operativo moderno.

A. Desbordamiento de búfer

Sucede cuando un programa no valida el tamaño de los datos de entrada y al superar el tamaño en memoria reservado para ellos, se sobre-escribe la dirección de memoria que el procesador utiliza para ejecutar la próxima instrucción del programa, permitiendo a un atacante tomar el control del flujo del mismo y ejecutar código arbitrario [9].

B. Desbordamiento de entero

En programación existen diferentes tipos de datos para representar valores numéricos enteros y almacenarlos en memoria y éstos tienen un rango de valores posibles limitado. Cuando se trata de almacenar un valor o realizar una operación matemática que excede la capacidad de los tipos de datos, se genera un *desbordamiento de entero*, que por sí solo no es muy peligroso, pero sí de ese entero depende una operación con memoria, se puede propiciar un *desbordamiento de búfer* [10].

C. Usar después de liberar

En la mayoría de los programas se realizan reservas de memoria en tiempo de ejecución, utilizando datos llamados punteros que referencian a las localidades de memoria reservadas. La vulnerabilidad, *Usar después de liberar,* es el resultado de acceder al contenido de la dirección de memoria reservada después de que ésta ha sido liberada [11], si un atacante es capaz de escribir datos en dicha localidad, se puede llegar a ejecutar el código arbitrario plantado por el atacante.

D. Condición de carrera

Esta vulnerabilidad existe cuando el cambio en el orden de dos o más eventos puede causar un cambio de comportamiento en un programa [12]. Se crea en escenarios donde diferentes procesos acceden a datos compartidos al mismo tiempo; como archivos, bases de datos, memoria, etc. En estas circunstancias un atacante podría insertar código malicioso en regiones compartidas de memoria y en otros casos tomar ventaja de pequeños lapsos de tiempo entre operaciones para interferir con la secuencia en que se éstas se realizan.

[a] Explotación. En el contexto utilizado, se refiere al aprovechamiento de una vulnerabilidad encontrada en un programa informático para obtener acceso a los recursos del sistema y potencialmente controlarlo.

En fin, existen multitud de vulnerabilidades explotables que no caen en una categoría específica, pero que al ser descubiertas representan un gran problema de seguridad en los sistemas afectados, ya que en la mayoría de los casos una vulnerabilidad en un programa específico pone en riesgo al sistema en su totalidad.

VI. Vulnerabilidades de día cero

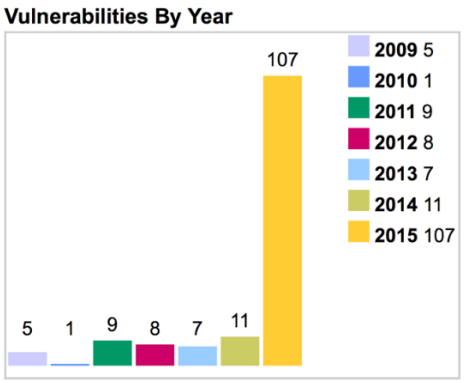
Aunque este concepto surge de aquellas vulnerabilidades explotadas el mismo día en que son descubiertas (en la mayoría de los casos no por el desarrollador[b]), se extiende también a aquellas vulnerabilidades que no han sido solucionadas en un periodo de tiempo específico.

El peligro de este tipo de vulnerabilidad reside en que puede surgir el caso en que los cibercriminales la encuentren y exploten antes de que el desarrollador este enterado de las mismas y pueda distribuir un parche[c] a sus usuarios.

NOTA: Este término usualmente se refiere a vulnerabilidades no detectadas por el desarrollador, sin embargo y aunque el desarrollador esté al tanto de la vulnerabilidad, mientras el usuario final no sea provisto con un parche que la solucione, el riesgo que representa es el mismo que aquel del *día cero*, es por esto que el termino se utiliza de forma equivalente en ambos casos.

VII. Vulnerabilidades importantes en la historia de Android

El propósito de esta sección es establecer un estado del arte en la explotación de vulnerabilidades en Android a través de su historia. Cabe destacar que aunque se han hecho públicas más de 140 vulnerabilidades desde 2009 hasta 2015, como se muestra en la siguiente figura tomada de [13], en este artículo solo se mencionan una por año desde 2013, esto para establecer una visión general, más que un reporte detallado.

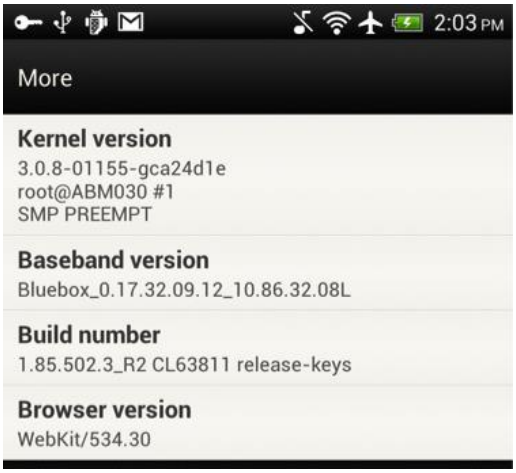


1) APK Duplicate file (Julio de 2013)

Como ya se mencionó anteriormente, las aplicaciones se distribuyen empaquetadas en archivos con extensión *APK*, que no son más que archivos comprimidos en formato *ZIP*.

Antes de que una *APK* sea descomprimida e instalada en un dispositivo, las firmas criptográficas[d] de sus contenidos son verificadas, buscando que coincidan con las establecidas en un archivo de manifiesto incluido en la misma *APK*. En caso de que las firmas no coincidan, la aplicación instaladora termina el proceso de instalación. Este proceso de verificación es una medida de seguridad tomada para evitar que los contenidos de una *APK* sean modificados después de que fueron firmados por las herramientas de desarrollo para Android, ya que podrían contener archivos y código diferentes al original.

El equipo de investigación de la compañía de seguridad *BlueBox* descubrió que cuando había archivos duplicados dentro de una *APK*, solo se verificaba la firma del primero de esos archivos y la última versión era la que se extraía y utilizaba para la app [14], con lo cual vieron la forma de modificar la aplicación encargada de mostrar información del software del dispositivo, y que por cierto cuenta con privilegios elevados, añadiendo la cadena de texto "Bluebox" dentro de la misma [15], tal como se puede ver en la siguiente figura:



Esta vulnerabilidad solo afecta a la versión de Android *Jelly Bean (Gummy Bear)* y anteriores, aunque fue publicada en Julio, su descubrimiento fue en Febrero y para el lanzamiento de *Jelly Bean (Michael),* en Julio 24, ya estaba parchada.

[b] El término *desarrollador* se utiliza para referirse a compañías que fabrican software o a personas independientes que realizan la misma tarea.

[c] Parche. Se refiere a una actualización que los desarrolladores de software proveen a sus usuarios y que soluciona un error detectado en dicho software que puede ser o no una vulnerabilidad.

[d] Firma Criptográfica. Son cadenas de texto generadas y que identifican a un archivo como único, de tal forma que se puede verificar su integridad y procedencia.

2) The Futex Vulnerability (Junio de 2014)

Esta es una vulnerabilidad descubierta por Nicholas Allegra [16] relacionada con un mecanismo en el Kernel de Linux que permite realizar interbloqueos en programas que necesiten ejecutar instrucciones en paralelo sobre regiones compartidas de memoria.

Permitió que el Hacker estadounidense, Geo Hotz, desarrollara la conocida app *TowelRoot*, que explota precisamente esta vulnerabilidad para conceder privilegios de *root* a dispositivos Android con el Kernel vulnerable.

Afecta a la versión *KitKat* y anteriores de Android.

Nota: Dada la naturaleza de esta vulnerabilidad, por su relación con una parte muy compleja del Kernel de Linux, he decidido excluir una explicación detallada, ya que por sí sola da material para un artículo completo.

3) Stagefright (Julio de 2015)

Todos los formatos de audio y video soportados por Android son procesados para su reproducción por un servicio escrito en el lenguaje de programación C++ llamado *libstagefright*. Este servicio se ejecuta en segundo plano y las apps de usuario pueden utilizarlo cuando requieren reproducir u obtener información contenida en los metadatos[e] de estos tipos de archivos.

El experto en seguridad, Joshua Drake, de la compañía *Zimperium - Mobile Security*, realizo una profunda investigación sobre este servicio y termino encontrando múltiples vulnerabilidades asociadas a él [17], incluyendo *Desbordamientos de entero* (de los cuales ya hemos hablado) y *Sobre-lectura de Búfer*[f].

El problema en realidad viene a la hora de procesar dichos archivos. Un fichero MP4 especialmente construido puede contener código malicioso que podría ejecutarse con los mismos privilegios del servicio vulnerable en cuestión, los cuales son bastante elevados, justo antes de *root* [18].

Los vectores de ataque[g] para desencadenar la explotación de la vulnerabilidad son muchos, pero el más peligroso fue aquel que no requería de interacción con el usuario. Este último era a través un MMS o mensaje multimedia, ya que el contenido de estos mensajes es procesado un vez que es recibido e incluso sin necesidad de que la pantalla del dispositivo este encendida. Con esto en mente, es fácil plantearse un escenario hipotético donde exista un malware lo suficiente complejo como para acceder a la lista de contactos del usuario, y con solo obtener el número de dos de ellos, terminamos obteniendo un malware con una tasa de distribución exponencial.

Esta vulnerabilidad es bastante nueva y actualmente la gran mayoría de dispositivos en uso siguen afectados, incluso una versión muy reciente de Android: *Lollipop*.

VIII. Explotando una vulnerabilidad real en una Tablet Android

Ahora que conocemos algunas de las vulnerabilidades que han o que siguen afectando a los dispositivos móviles Android, es momento de plantear la pregunta **más** importante de esta investigación:

¿Qué tan difícil es explotar una vulnerabilidad en Android?

Aunque se piense que se requieren conocimientos altamente especializados en arquitectura de computadoras y sistemas operativos, en realidad en algunos casos solo es necesario saber utilizar el lenguaje de programación *Java* y tener conocimientos básicos sobre el *framework* de aplicaciones de Android, y esto se debe a que, como ya se demostró en la sección anterior, existen ciertas vulnerabilidades en la capa de aplicaciones de la arquitectura de Android que tienen un impacto moderado en la seguridad del SO.

Para probar lo dicho en el anterior párrafo, en esta sección se **mostrarán** los resultados de una *prueba de concepto*[h] desarrollada en forma de app que se aprovecha de una vulnerabilidad en el sistema de instalación de aplicaciones descubierta por la compañía de seguridad *Palo Alto Networks*, nombrada por ellos como *Android Installer Hijacking,* y que afecta a la versión de Android *Jelly Bean (Gummy Bear)* y anteriores [19].

¿En qué consiste la vulnerabilidad a explotar?

En la sección anterior hablamos de la vulnerabilidad *APK Duplicate file*, donde se modificaba una *APK* para burlar parcialmente la protección, mediante firmas criptográficas, del sistema de instalación de apps.

En este caso se hace un bypass[i] completo a esta protección mediante el reemplazo de una app durante su

[e] En este contexto, los metadatos conforman la información que describe el contenido de un archivo, como autor, fecha de creación, tamaño, etc.

[f] Sobre-lectura de Búfer – Se produce cuando se intenta leer el contenido un búfer pasados los límites en memoria del mismo.

[g] Vector de ataque. Mecanismo utilizado para desencadenar una vulnerabilidad y su subsecuente explotación.

[h] POC. Demostración de la veracidad de una vulnerabilidad.

[i] Bypass. Burlar o evadir los controles de seguridad en un sistema informático.

instalación por una tercera, después de que la primera ha sido verificada previamente, e instalando la segunda app sin necesidad de que haya pasado por el proceso de instalación convencional. Es decir, el dispositivo puede terminar con una aplicación instalada con un sin número de permisos que el usuario no tuvo oportunidad de revisar y aceptar.

Vector de ataque para propiciar la explotación

Para explotar esta vulnerabilidad y poder instalar malware en un dispositivo, es necesaria la interacción con el usuario, y por ende la utilización de ingeniería social[j].

Para nuestra prueba de concepto se montó un escenario con los siguientes elementos:

a. App administradora (Aplicación inofensiva). Hace el primer contacto con el usuario, se encarga de manipularlo para que instale una segunda app aparentemente benigna.
b. App señuelo (Connection1). Es una app que servirá como señuelo y que aparentemente no representa ningún peligro, porque la lista de permisos manifestados en su *APK* son pocos o nulos.
c. Malware (EjemploService). Reemplazará a la App señuelo durante su instalación y puede o no tener el mismo nombre que ésta. Potencialmente utiliza multitud de permisos que el usuario no puede declinar para proteger su seguridad.

Desarrollo de la prueba de concepto

Nota: No se mencionan detalles de implementación, pero se brinda una explicación que puede permitir la comprensión de la lógica global del ataque.

Se creó la siguiente interfaz de usuario que nos permitirá observar el experimento detalladamente:

Al iniciarse, esta aplicación crea un nuevo directorio vacío, llamado POC, dentro del almacenamiento externo de la Tablet.

Cuando se toca el botón "Iniciar servicio monitor", se crea un servicio mediante la API[k] IntentService de Android que se ejecutará en segundo plano. Una vez iniciado, el servicio comienza a monitorear un directorio; aquel preparado previamente y cuyo directorio raíz contiene a las *APKs* Connection1 y EjemploService, de quienes sus funcionalidades no tienen relevancia para con esta prueba de concepto.

En la siguiente pantalla se puede ver la salida del depurador[l] de Android Studio[m], donde se muestra lo dicho anteriormente.

```
DirMonitorService: SERVICIO MONITOR INICIADO.
DirMonitorService: DIRECTORIO A MONITORIAR: /mnt/sdcard/DCIM/POC
DirMonitorService: CONTENIDOS DEL DIRECTORIO PADRE DEL DIRECTORIO A MONITORIAR
FileManager:  /mnt/sdcard/DCIM/Camera
FileManager:  /mnt/sdcard/DCIM/100ANDRO
FileManager:  /mnt/sdcard/DCIM/browser-images
FileManager:  /mnt/sdcard/DCIM/EjemploService.apk
FileManager:  /mnt/sdcard/DCIM/Connection1.apk
FileManager:  /mnt/sdcard/DCIM/POC
```

En este punto solo hace falta tocar el botón "Iniciar experimento". Este botón desencadena las siguientes operaciones:

1. Copiar el archivo "Connection1.apk" al directorio POC.
2. Se le solicita a Android que inicie el proceso de instalación de la *APK* copiada en el directorio POC.
3. Después de que el instalador muestra los permisos requeridos por la app, se sobre-escribe el archivo *APK* correspondiente a "Connection1" con el contenido de "EjemploService.apk".
4. Se destruye el servicio monitor.

En la siguiente figura observamos la salida del depurador, donde se muestran los pasos anteriores:

```
: Iniciando POC
rver: ALGO HA PASADO: 'CREATE' - 'Connection1.apk'
 ARCHIVO '/mnt/sdcard/DCIM/Connection1.apk' COPIADO A '/mnt/sdcard/DCIM/POC/Connection1.apk'.
: Harmles APK Sheme: file
: Harmfull APK Sheme: file
: Intento para instalar APK iniciado.
 ARCHIVO '/mnt/sdcard/DCIM/POC/Connection1.apk' ELIMINADO.
 ARCHIVO '/mnt/sdcard/DCIM/EjemploService.apk' COPIADO A '/mnt/sdcard/DCIM/POC/Connection1.ap
 ARCHIVO '/mnt/sdcard/DCIM/EjemploService.apk' ESCRITO.
```

[j] Técnicas utilizadas para manipular a un usuario y lograr que realice una acción requerida por un atacante.

[k] API - Interfaz de programación de aplicaciones. Provee acceso a funcionalidades pre-programadas que facilitan el desarrollo de software.

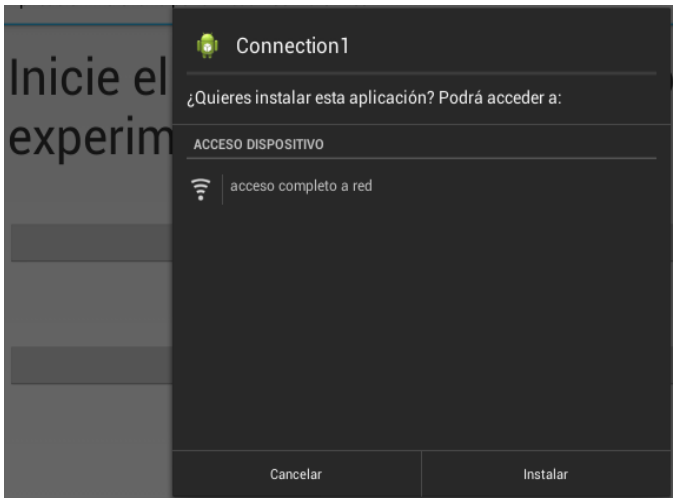
[l] Depurador. Herramienta utilizada por desarrolladores para detectar errores en sus programas o simplemente para saber el estado de los mismos.

[m] Android Studio. Software utilizado para desarrollar aplicaciones para el sistema operativo Android.

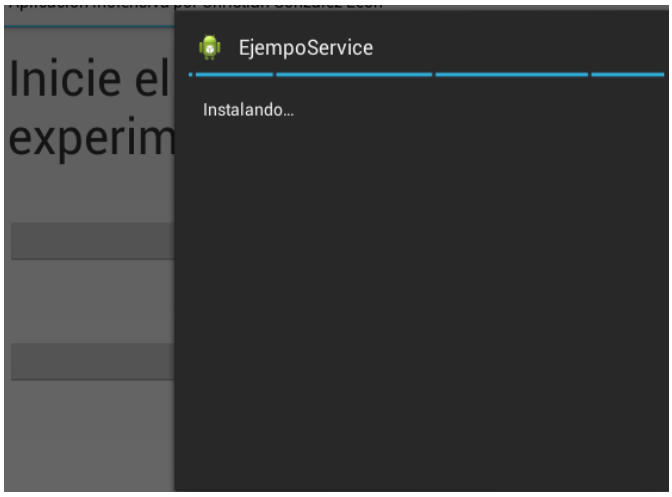
Las operaciones que se realizan sobre el archivo "Connection.apk" se pueden apreciar en la siguiente figura.

```
POC_FileObserver: ALGO HA PASADO: 'MODIFY' - 'Connection1.apk'
POC_FileObserver: ALGO HA PASADO: 'ACCESS' - 'Connection1.apk'
POC_FileObserver: ALGO HA PASADO: 'DELETE' - 'Connection1.apk'
POC_FileObserver: ALGO HA PASADO: 'CREATE' - 'Connection1.apk'
POC_FileObserver: ALGO HA PASADO: 'MODIFY' - 'Connection1.apk'
DirMonitorService: SALIENDO DEL SERVICIO MONITOR.
```

En la siguiente figura se muestra la pantalla de interacción con el usuario, donde acepta los permisos que la app requiere, o cancela la instalación.



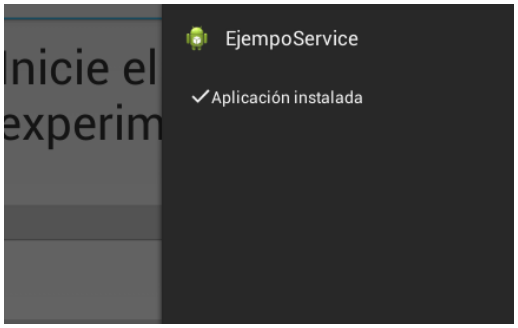
Una vez que el usuario toca el botón "Instalar", en realidad **se instala el malware**, ya previamente se sustituyó el contenido de la app original, veamos las capturas de pantalla tomadas durante el experimento:



El depurador muestra lo siguiente repetidas veces y de forma consecutiva durante el tiempo en que el instalador despliega la barra de progreso de la figura anterior:

```
ALGO HA PASADO: 'ACCESS' - 'Connection1.apk'
ALGO HA PASADO: 'ACCESS' - 'Connection1.apk'
ALGO HA PASADO: 'ACCESS' - 'Connection1.apk'
ALGO HA PASADO: 'ACCESS' - 'Connection1.apk'
ALGO HA PASADO: 'ACCESS' - 'Connection1.apk'
```

A pesar de que en la anterior y siguiente pantalla se puede ver que la aplicación señuelo y el malware tienen nombres diferentes, esto es fácilmente configurable desde Android Studio y puede permitir que ambas apps muestren el mismo nombre durante su instalación, haciendo imposible para el usuario detectar que ha sido infectado con malware.



¿Por qué utilizar una Tableta para el experimento y no un Smartphone?

Android es un sistema operativo que corre en multitud de dispositivos, no solo Smartphones. Uno de estos dispositivos son las denominadas Tablet, que permiten una mejor visualización del contenido mostrado, gracias a que cuentan con una pantalla de mayor tamaño.

Pues bien, en México existe un programa social administrado por la Secretaría de Educación Pública llamado "Programa de Inclusión y Alfabetización Digital" [20], el cual esta activo desde el año 2013.

En el ciclo escolar iniciado en la segunda mitad del 2014, se comenzó una iniciativa que consiste en entregar gratuitamente Tabletas a niños cursando el quinto año de la educación primaria, así como a los profesores. En la primera dotación de tabletas se entregaron 131 mil 188 dispositivos [21], la gran mayoría corriendo Android en su versión *Jelly Bean (Gummy Bear)* [22] y vendidas por la empresa *IUSA MEDICION S.A. DE C.V* [23]:

La vulnerabilidad explotada justamente afecta a esta versión de Android y anteriores. La tableta utilizada en el experimento utiliza precisamente esta versión.

Es decir, multitud de niños de 11 a 12 años son actualmente vulnerables a ataques por malware, algunos de los cuales exponen a los usuarios a contenido solo para

adultos, tal como fue demostrado por FireEye en su descubrimiento de un nuevo malware que comienza ganando privilegios *root,* para que en cierto momento se descarguen aplicaciones sin el consentimiento del usuario y que muestran este tipo de contenido [24].

En la prueba de concepto desarrollada, el malware instalado en lugar de la app original, podría en un escenario real contener contenido peligroso para los infantes, como pornografía, violencia explicita, etc.

IX. CONCLUSIÓN

El sistema operativo Android fue diseñado pensando en la seguridad como un factor prioritario, sin embargo, errores en la implementación de las aplicaciones y en módulos del sistema de un nivel más bajo, generan comportamientos no deseados que terminan en vulnerabilidades explotables que pueden poner en serio peligro a los usuarios, desde molestar con simple publicidad invasiva, hasta robar su información sensible con fines delictivos como la extorción o el secuestro.

Lo mejor que los usuarios pueden hacer para prevenir ser infectados por malware a través la explotación de vulnerabilidades, es instalar las actualizaciones que los fabricantes de sus dispositivos ponen a su disposición. Aunque en realidad siempre aparecen nuevas vulnerabilidades de día cero que podrían burlar cualquier medida de seguridad.

También se recomienda no adquirir terminales que ejecuten versiones antiguas de Android, ya que éstas son las más propensas a ser contener vulnerabilidades explotables.

Con el presente trabajo se espera que los lectores hagan conciencia sobre los riesgos que la utilización de dispositivos desactualizados, susceptibles a vulnerabilidades explotables, representan para su seguridad y se les exhorta a que busquen por su cuenta más fuentes de información sobre el tema, para que conozcan las prácticas recomendadas para evitar ser infectados con malware.

REFERENCIAS

[1] The competitive Intelligence Unit (2015), Ecosistema Competitivo del Mercado de Smartphones 4T14, URL: http://the-ciu.net/nwsltr/350_1Distro.html, Fecha de consulta: 26/11/2015.

[2] StatsCounter GlobalStats, Top Mobile Operating Systems from Oct 2014 to Oct 2015, URL: http://gs.statcounter.com/#mobile_os-ww-monthly-201410-201510-bar, Fecha de consulta: 26/11/2015.

[3] Wikipedia La Enciclopedia Libre, ILoveYou, URL: https://es.wikipedia.org/wiki/IloveYou, Fecha de consulta: 27/11/2015.

[4] Android Open Source Project, Security, URL: https://source.android.com/security/, Fecha de consulta: 27/11/2015.

[5] Android Developers, Introduction to Android, URL: http://developer.android.com/intl/es/guide/index.html, Fecha de consulta: 27/11/2015.

[6] Android Developers, Building and Running Overview, URL: http://developer.android.com/intl/es/tools/building/index.html, Fecha de Consulta: 27/11/2015.

[7] Android Open Source Project, Security and Kernel security, URL: https://source.android.com/security/overview/kernel-security.html, Fecha de Consulta: 27/11/2015.

[8] Android Developers, System Permissions, URL: http://developer.android.com/intl/es/guide/topics/security/permissions.html, Fecha de consulta: 27/11/2015.

[9] Welivesecurity en español, Qué son y cómo funcionan los Buffer Overflow, URL: http://www.welivesecurity.com/la-es/2014/11/05/como-funcionan-buffer-overflow/, Fecha de consulta: 28/11/15.

[10] Security et alii, Explotation: Integer Overflow, URL: http://securityetalii.es/2010/03/12/exploitation-integer-overflow/, Fecha de consulta: 28/11/15.

[11] Security Intelligence, Use-After-Frees: That pointer may be pointing to something bad, URL: https://securityintelligence.com/use-after-frees-that-pointer-may-be-pointing-to-something-bad/, Fecha de consulta: 28/11/15.

[12] Mac Developer Library – Secure Coding Guide, Race Conditions and Secure File Operations, URL: https://developer.apple.com/library/mac/documentation/Security/Conceptual/SecureCodingGuide/Articles/RaceConditions.html#//apple_ref/doc/uid/TP40002585-SW1, Fecha de consulta: 28/11/15.

[13] CVE Details – The ultimate security vulnerability datasource, Google>Android>Vulnerability Statistics, URL: https://www.cvedetails.com/product/19997/Google-Android.html?vendor_id=1224, Fecha de consulta: 28/11/15.

[14] Paul Ducklin, Anatomy of a security hole – Google's "Android Master Key" debacle explained, URL: https://nakedsecurity.sophos.com/2013/07/10/anatomy-of-a-security-hole-googles-android-master-key-debacle-explained/, Fecha de consulta: 29/11/15.

[15] Jeff Forristal, Android Master Key Exploit – Uncovering Android Master Key, URL: https://bluebox.com/uncovering-android-master-key-that-makes-99-of-devices-vulnerable/, Fecha de consulta: 29/11/2015.

[16] Nicholas Allegra, Linux PI futex self-requeue bug, URL: https://hackerone.com/reports/13388, Fecha de consulta: 29/11/2015.

[17] Zimperium – Mobile Security, Experts Found a Unicorn in the Heart of Android, URL:

http://blog.zimperium.com/experts-found-a-unicorn-in-the-heart-of-android/, Fecha de consulta: 29/11/15.

[18] Black Hat (2015, August 25), Stagefright: Scary Code in the Heart of Android [Archivo de video], Recuperado de: https://www.youtube.com/watch?v=71YP65UANP0.

[19] Palo Alto Networks, Android Installer Hijacking Vulnerability Could Expose Android Users to Malware, URL: http://researchcenter.paloaltonetworks.com/2015/03/android-installer-hijacking-vulnerability-could-expose-android-users-to-malware/, Fecha de consulta: 29/11/2015.

[20] Secreataría de Educación Pública – México, Preguntas frecuentes – Dotación de tabletas – Cilo escolar 2014/2015, URL: http://basica.sep.gob.mx/preguntas/index.html, Fecha de consulta: 30/11/2015.

[21] Educación future – Periodismo de interes público, Reparte SEP tabletas a niños de primaria, URL: http://www.educacionfutura.org/reparte-sep-tabletas-a-ninos-de-primaria/, Fecha de consulta: 30/11/2015.

[22] Programa de inclusion y Alfabetización Digital, Manual de usuario – Tableta RECREO TR10CS1, URL: http://www.dee.edu.mx:8080/piad/resource/pdft/Manua_Usuario_Tableta_RECREO.pdf, Fecha de consulta: 31/11/2015.

[23] Dirección General de Comunicación Social, Comunicado 072.- Concluyó el proceso de licitación para adquirir tabletas para alumnos de 5º de primaria, URL: http://www.comunicacion.sep.gob.mx/index.php/comunicados-2014/48-marzo-2014/459-comunicado-072-concluyo-el-proceso-de-licitacion-para-adquirir-tabletas-para-alumnos-de-5-de-primaria, Fecha de consulta: 30/11/2015.

[24] Fire Eye, Guaranteed Clicks: Mobile App Company Takes Control of Android Phones, URL: https://www.fireeye.com/blog/threat-research/2015/09/guaranteed_clicksm.html, Fecha de consulta: 30/11/2015.