UNIVERSIDAD AUTÓNOMA DEL ESTADO DE MÉXICO

FACULTAD DE INGENIERÍA

ANÁLISIS DEL ESTADO DEL ARTE DE LAS VULNERABILIDADES DEL SISTEMA OPERATIVO ANDROID

TESINA

QUE PARA OBTENER EL TÍTULO DE

INGENIERO EN COMPUTACIÓN

PRESENTA:

Rafael Iniesta Estrada

ASESOR:

Dr. Marcelo Romero Huertas

TOLUCA MÉXICO, NOVIEMBRE 2020

RESUMEN

Android es un sistema operativo móvil desarrollado por Google; es uno de los más conocidos junto con iOS de Apple. Está basado en Linux, que junto con aplicaciones middleware está enfocado para ser utilizado en dispositivos móviles como teléfonos inteligentes, Tablets, Google TV y otros dispositivos.

Una de las claves de la popularidad de Android es que, como Linux, es una plataforma de código abierto, lo que permite a fabricantes, operadores y desarrolladores dar mayor funcionalidad a sus Smartphone. Cinco millones de terminales al mes salen con Android, lo que no deja de preocupar a los que tienen su propio sistema cerrado, como Nokia (Symbian), Apple (iOS) o RIM (Blackberry). Además Android es un sistema gratuito y multiplataforma; por multiplataforma entendemos que el sistema operativo puede ser usado en distintas plataformas, y por plataforma entendemos que es una combinación de hardware y software usada para ejecutar aplicaciones; en su forma más simple consiste únicamente de un sistema operativo, una arquitectura, o una combinación de ambos. Android es gratuito al poder ir instalado gratuitamente en cualquier dispositivo móvil.

Por todo lo mencionado anteriormente, es el sistema operativo más atacado y vulnerable del mercado, ya que al ser de código abierto (gratuito) cualquiera puede hacer uso del mismo para explotar las vulnerabilidades que se llegan a observar en el sistema operativo, de igual forma crear virus, aplicaciones las cuales puedan hacerle daño al usuario, secuestrar sus datos entre algunos otros mecanismos que utilizan para corromper la seguridad de dicho sistema al ser gratuito

El enfoque principal de este trabajo consiste en documentar las vulnerabilidades del sistema operativo Android más relevantes y comunes con la finalidad de proteger los dispositivos móviles de los usuarios, de igual forma se realiza una guía de recomendaciones de manera preventiva para ayudar a los usuarios a erradicar dichas vulnerabilidades sin que afecten a los dispositivos, mostrar los puntos necesarios para poner en marcha cada una de las recomendaciones de acuerdo a lo que se le pueda presentar al usuario.

INTRODUCCIÓN

El software para computadoras puede clasificarse en general en dos clases: los programas de sistema y los programas de aplicación. Los programas de sistema controlan la operación de la computadora, mientras que los programas de aplicación resuelven problemas específicos de los usuarios. El programa fundamental de todos los programas de sistema es el sistema operativo, el cual controla todos los recursos de la computadora y proporciona la base sobre la cual pueden ejecutarse los programas de aplicación.

Un sistema de computación moderno consta de uno o más procesadores, cierta memoria principal (a menudo conocida como *memoria central*), relojes, terminales, discos, interfaces de red y otros dispositivos de entrada/salida.

Los verdaderos clientes del sistema operativo son los programas de aplicaciones, los cuales son administrados directamente por el sistema operativo.

Un sistema operativo es un programa (software) encargado de poner en funcionamiento el equipo de cómputo, puesto que gestiona los procesos básicos del sistema. Así mismo se encarga de gestionar para el usuario el hardware. El sistema operativo comienza a trabajar en cuanto se inicia el equipo de cómputo y es esencial para que el usuario trabaje con el, puesto que el sistema operativo administra el hardware, el sistema de archivos y los procesos de las aplicaciones.

Existen diversos tipos de sistemas operativos, los cuales se derivan de acuerdo a la arquitectura hardware y la aplicación para la cual se desee orientar. Algunos sistemas operativos distintivos en el mercado son:

1. La familia Windows de Microsoft, por ejemplo: Windows XP, Windows Vista; los cuales son sistemas operativos de mayor uso para los usuarios finales e incluye una amplia gama de arquitectura hardware (Desde teléfonos celulares hasta servidores).

2. La familia Mac OS, los cuales son sistemas operativos propiedad de la empresa Apple, su uso incluye desde teléfonos inteligentes hasta servidores.

3. Unix: Es un sistema operativo empleado por las supercomputadoras y equipos de cómputo robustos, el cual es propiedad de la empresa AT&T y se considera un sistema operativo estable, robusto y seguro.

4. GNU/Linux: Sistema operativo cuyo uso está aumentando con los usuarios finales, debido a que es un software de licencia libre y puede ajustarse a las necesidades funcionales de forma estable y segura.

5. FreeBSD: Sistema operativo gratuito y según algunos autores uno de los más seguros.

6. Android: El Sistema operativo Android, es de código abierto, gratuito y no requiere del pago de licencias. Hoy en día es el más usado, siendo en consecuencia, uno de los más vulnerables en el mercado. Los dispositivos donde funciona actualmente son: los teléfonos y relojes inteligentes, tabletas, Smart TV y automóviles.

La parte más importante del Sistema Operativo se llama núcleo o kernel. Asigna tareas al procesador siguiendo un orden y administrando los tiempos que lleva cada tarea.

Independientemente del tamaño y complejidad de la PC y el sistema operativo, todos los sistemas operativos realizan las mismas cuatro funciones básicas:

- Control de acceso de hardware
- Administración de archivos y carpetas
- Proporción de una interfaz de usuario
- Administración de aplicaciones

SISTEMA OPERATIVO MOVIL

Partiendo de la definición de Sistema Operativo: Capa compleja entre el hardware y el usuario concebible también como una máquina virtual que facilita al usuario o al programador las herramientas e interfaces adecuadas para realizar sus tareas informáticas, abstrayéndose de los complicados procesos necesarios para llevarlas a cabo. Un sistema operativo móvil, es un sistema que controla un dispositivo móvil, de la misma forma en que las computadoras utilizan un Windows o un Linux entre otros. Estos sistemas operativos móviles son mucho más simples que los que usan las computadoras y generalmente van enfocados a la conectividad inalámbrica, los formatos multimedia para móviles y la manera en que se introduce la información en ellos.

Entre los sistemas operativos móviles más usados son Android, iOS, Windows Phone, kaiOS y Samsung los cuales resaltan como los sistemas operativos móviles con una mayor cuota de mercado en todo el mundo (StatCounter, 2020).

Android es el sistema operativo más utilizado a nivel mundial con un 74.43% por encima de Windows Phone y iOS de Apple lo cual está revolucionando actualmente el mercado mundial del Smartphone, haciendo de este sistema operativo móvil un gigante de la industria de las telecomunicaciones. El elemento más notable de Android es que es una plataforma de código abierto y cualquier elemento que haga falta o se pierda puede ser provisionado por un sinfín de usuarios que hacen parte de la comunidad Android a nivel mundial.

Los nombres que fueron asociados a sus diferentes versiones tales como: Apple Pie, Banana Bread, Cupcake, Donut, Eclair, Froyo, Honeycomb, Gingerbread, Ice Cream-Sandwich, Jelly Bean, KitKat, Lollipop, Marshmallow, Nougat, Oreo y Pie que lo hacen un sistema operativo especial dado la originalidad utilizada en los mismos.

Android es un entorno de software integrado para dispositivos móviles y no es una plataforma de hardware como lo puede pensar mucha gente, incluye un sistema Linux basado en el kernel del mismo sistema operativo, interfaz de usuario, es una plataforma muy rica en aplicaciones para el usuario final, bibliotecas de código, entornos de aplicaciones, soporte multimedia y mucho más, incluyendo por supuesto la funcionalidad de telefonía celular.

Al final del año 2019, fueron identificadas 414 vulnerabilidades en el sistema operativo Android, una reducción considerable al año 2018 que fueron encontradas 613 vulnerabilidades 40.5 % , donde la vulnerabilidad de desbordamiento de búfer fue muy explotada (CVE Details,2020).

El 6.76% de las fallas que podría implicar la denegación del servicio (DoS), ejecución de código malicioso con un 13.76% el 45% de estos correspondieron a un estado crítico, lo que implica que podría tener un grave impacto para la información almacenada en el equipo.

El *malware,* por su parte, se concentró a nivel mundial en un 37.4%, en países de Latinoamérica, México (25%), Perú (15%), Brasil (15%), Colombia (7%), Argentina (6%) y el resto de los países de Latinoamérica con el 32% (welivesecurity, 2020).

No obstante, el informe detalla que las amenazas en la *Play Store* no han dejado de ser frecuentes; por el contrario, son comunes los casos de troyanos disfrazados de aplicaciones benignas que logran saltar los controles de seguridad de *Google* para afectar a los usuarios.

Dado el crecimiento de software malicioso detectado es relevante identificar las vulnerabilidades y posibles soluciones para prevenir a los usuarios del sistema operativo Android

1.1 Objetivo general

Documentar las vulnerabilidades del Sistema Operativo Android para identificar recomendaciones que permitan proteger los dispositivos inteligentes donde se ha instalado.

1.2 Organización del documento

La presente tesina está integrada por tres capítulos adicionales al presente:

El Capítulo I –"Sistema Operativo Android", describe las generalidades del Sistema Operativo Android.

El Capítulo II –"Vulnerabilidades del Sistema Operativo Android", Se documentan las vulnerabilidades del Sistema Operativo Android.

El Capítulo III – "Discusión", Se analiza el impacto de las vulnerabilidades documentadas del Sistema Operativo Android.

Para finalizar el documento, se incluyen las conclusiones y un anexo con un glosario de términos.

TABLA DE CONTENIDO

CAPÍTULO I. SISTEMA OPERATIVO ANDROID

En este capítulo se describen las generalidades, características, arquitectura, sistema de protección, administración del CPU, las versiones y los componentes de una aplicación del sistema operativo Android.

1.1 GENERALIDADES DEL SISTEMA OPERATIVO ANDROID

Android es un sistema operativo móvil desarrollado por Google; es uno de los más conocidos junto con iOS de Apple. Está basado en Linux, que junto con aplicaciones middleware está enfocado para ser utilizado en dispositivos móviles como teléfonos inteligentes, Tablets, Google TV y otros dispositivos.

Fue desarrollado por Android Inc., empresa que en 2005 fue comprada por Google, aunque no fue hasta 2008 cuando se popularizó, gracias a la unión al proyecto de Open Handset Alliance, un consorcio formado por 48 empresas de desarrollo hardware, software y telecomunicaciones, que decidieron promocionar el software libre. Pero ha sido Google quien ha publicado la mayor parte del código fuente del sistema operativo, gracias al software Apache, que es una fundación que da soporte a proyectos de software de código abierto.

Dado que Android está basado en el núcleo de Linux, tiene acceso a sus recursos, pudiendo gestionarlo, gracias a que se encuentra en una capa por encima del Kernel, accediendo así a recursos como los controladores de pantalla, cámara, memoria flash, etc.

En los últimos años los teléfonos móviles han experimentado una gran evolución, desde los primeros terminales, grandes y pesados, pensados sólo para hablar por teléfono en cualquier parte, a los últimos modelos, con los que el término “medio de comunicación” se queda bastante pequeño.

Es así como nace Android. Android es un sistema operativo y una plataforma software, basado en Linux para teléfonos móviles. Además, también usan este sistema operativo Tablets, Netbooks, reproductores de música e incluso PC’s. Android permite programar en un entorno de trabajo (Framework) de Java, aplicaciones sobre una máquina virtual Dalvik (una variación de la máquina de Java con compilación en tiempo de ejecución). Además, lo que le diferencia de otros sistemas operativos, es que cualquier persona que sepa programar puede crear nuevas aplicaciones, *widgets*, o incluso, modificar el propio sistema operativo, dado que Android es de código libre, por lo que sabiendo programar en lenguaje Java, va a ser muy fácil comenzar a programar en esta plataforma.

1.2 CARACTERÍSTICAS DEL SISTEMA OPERATIVO ANDROID

Las características más sobresalientes del sistema operativo Android son las siguientes: código abierto, núcleo basado en el Kernel de Linux, adaptable a muchas pantallas y resoluciones, utiliza SQLite para el almacenamiento de datos, ofrece diferentes formas de mensajería, navegador web basado en WebKit incluido, soporte de java y muchos formatos multimedia, soporte de HTML, HTML5, Adobe Flash Player, etc., incluye un emulador de dispositivos, herramientas para depuración de memoria y análisis del rendimiento del software, catálogo de aplicaciones gratuitas o de paga en el que pueden ser descargadas e instaladas (Google Play), Bluetooth, Google Talk desde su versión HoneyComb, para realizar videollamadas y multitarea real de aplicaciones.

1.3 ARQUITECTURA DEL SISTEMA OPERATIVO ANDROID

La Figura 1 nos muestra cómo está compuesta la arquitectura del sistema operativo Android, donde podemos ver que consta de 4 capas las cuales son: aplicaciones, librerías, *Android Runtime* y el *Kernel de Linux*, a continuación se describe cada una de ellas.

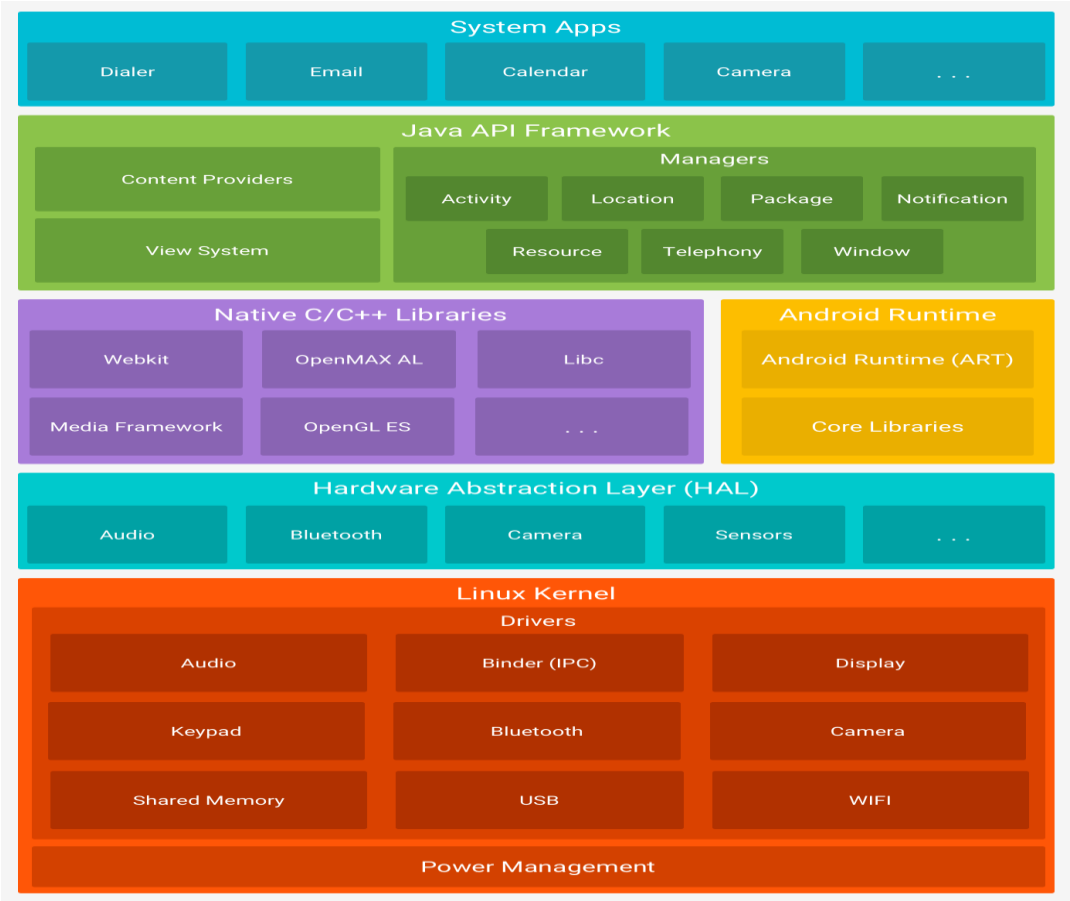


Figura 1. Arquitectura de Android (Developers, 2020)

• Aplicaciones: incluyen un cliente de correo electrónico, programa de SMS, calendario, mapas, navegador, contactos y otros. Todas ellas escritas en Java.

• Marco de trabajo de aplicaciones: los desarrolladores tienen acceso completo a los mismos APIs del Framework usados por las aplicaciones base. La arquitectura está diseñada para simplificar la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y cualquier otra aplicación puede luego hacer uso de esas capacidades.

• La capa de abstracción de hardware (HAL) brinda interfaces estándares que exponen las capacidades de hardware del dispositivo al marco de trabajo de la API de Java de nivel más alto. La HAL consiste en varios módulos de biblioteca y cada uno de estos implementa una interfaz para un tipo específico de componente de hardware, como el módulo de la cámara o de Bluetooth. Cuando el marco de trabajo de una API realiza una llamada para acceder a hardware del dispositivo, el sistema Android carga el módulo de biblioteca para el componente de hardware en cuestión.

• Bibliotecas o librerías: incluye un conjunto de bibliotecas de C/C++ usadas por varios componentes del sistema.

• Android Runtime: incluye un set de bibliotecas base que proporcionan la mayor parte de las funciones disponibles en las bibliotecas base del lenguaje Java. Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik.

• Kernel de Linux: Android depende de Linux para los servicios base del sistema como seguridad, gestión de memoria, gestión de procesos, pila de red y modelo de controladores. También actúa como capa de abstracción entre el hardware y el resto de la pila de software.

1.4 SISTEMA DE PROTECCIÓN DE ANDROID

En el año 2012, Google revelo que cuenta con un sistema de protección denominado "Bouncer", que detecta aplicaciones con código malicioso en la misma plataforma una vez que son subidas.

Bouncer provee un escaneo automático del Android Market buscando software potencialmente malicioso sin alterar la experiencia del usuario en el Android Market o hacer que los desarrolladores pasen por un proceso de aprobación de aplicaciones.

El software básicamente funciona escaneando cualquier tipo de comportamiento sospechoso de una aplicación y alertando en caso de encontrar cualquier indicio de malware o spyware.

De esta forma se consigue un banco de aplicaciones más fiable donde el usuario puede descargar con cierta tranquilidad las aplicaciones de su gusto.

1.4.1 ADMINISTRACIÓN DEL CPU

- Kernel: El núcleo o Kernel proporciona el acceso a los distintos elementos del hardware del dispositivo. Ofrece distintos servicios a las superiores como son los controladores o drivers para el hardware, la gestión de procesos, el sistema de archivos y el acceso a la gestión de la memoria.

- Middleware: El middleware es el conjunto de módulos que hacen posible la propia existencia de aplicaciones para móviles. Es totalmente transparente para el usuario y ofrece servicios claves como el motor de mensajería y comunicaciones, códec multimedia, intérpretes de páginas web, gestión del dispositivo y seguridad.

- Aplicaciones: El entorno de ejecución de aplicaciones consiste en un gestor de aplicaciones y un conjunto de interfaces programables abiertas y programables por parte de los desarrolladores para facilitar la creación de software.

- Interfaz de usuario: Las interfaces de usuario facilitan la interacción con el usuario y el diseño de la presentación visual de la aplicación. Los servicios que incluye son el de componentes gráficos (botones, pantallas, listas, etc.) y el del marco de interacción.

Aparte de estas capas también existe una familia de aplicaciones nativas del teléfono que suelen incluir los menús.

1.5 VERSIONES DEL SISTEMA OPERATIVO ANDROID

Las versiones del sistema operativo Android se enlistan y describen a continuación:

a) **Android 0.5 El inicio de todo:** Esta primera versión llego bajo el nombre de Android 0.5 Milestone 3. La interfaz estaba adaptada a terminales de pantalla "pequeña", y solo se incorporaban aplicaciones como Google Maps, un navegador, y otras herramientas esenciales de un teléfono.

   Junto a Milestone 5 llegó uno de los cambios más importantes en la historia de Android, Fue liberada en febrero de 2008. Introdujo por primera vez uno de los componentes esenciales del sistema operativo: el panel de notificaciones.

   Dado que esta edición estaba orientada a su utilización en emuladores, y no estaba previsto su funcionamiento en dispositivos físicos, muchas de las funcionalidades del sistema aún no estaban disponibles. En esta versión **Android no pudo incluir el gesto *pinch to zoom* para ampliar en imágenes o páginas web.**

b) **Android 0.9 Beta**: Android 0.9 Beta fue la última versión de pruebas antes de que el sistema operativo viese la luz públicamente. **Pantalla de inicio paginada que permitía añadir widgets nativos**, un mayor número de aplicaciones, y por fin, **un cajón de aplicaciones** deslizable desde la parte inferior.

   El panel de notificaciones introdujo la opción de eliminar todos los avisos, y los iconos de la barra de estado fueron modificados para una lectura más clara.

   Se incluyó el dialer, la app de contactos, la alarma o el navegador. Además, Android 0.9 Beta fue la primera versión del sistema en la que Google Maps comenzaba a funcionar, **preparándose así para dar el salto, por primera vez, a un *hardware* de verdad.**

c) **Android 1.0 Apple Pie**: Llega en octubre de 2008 en el HTC Dream o Google G1.
   En Apple Pie, se introdujo por primera vez la suite de aplicaciones de Google, comúnmente conocida como GApps. Entre ellas, además, se encontraba una herramienta: Android Market, una tienda, exclusiva de aplicaciones y juegos y donde los desarrolladores ni siquiera podían cobrar por sus creaciones, el plan de Google era hacer de ella una plataforma repleta de contenido más allá de aplicaciones.

   Se encontraban otras aplicaciones como Calendar, Gmail, Ajustes, y una versión muy temprana de YouTube. Se incluyeron opciones de seguridad para la pantalla de inicio, entre ellas el patrón de desbloqueo y aparecieron pequeños detalles a nivel de interfaz como el aviso por batería baja a partir del 15%.

d) **Android 1.1 Banana Bread:** Android 1.1 está considerada la primera actualización incremental del sistema operativo, dado que llegaba para solucionar la mayoría de errores descubiertos en Android 1.
   Búsqueda por voz gracias a Google Voice Search, cuyo funcionamiento consistía en comandos de voz que posteriormente se traducirían en búsquedas en el buscador.

   Google introdujo por primera vez el soporte para aplicaciones y juegos de pago en la tienda de apps, convirtiendo así el Market en una vía de obtención de ingresos para los desarrolladores.
   Google Latitude, que permitía a los usuarios compartir su ubicación con el resto del mundo.

e) **Android 1.5 Cupcake:** El añadido más importante de esta versión, fue un componente: el teclado virtual. El panel de notificaciones también fue rediseñado como en prácticamente todas las versiones de Android, recibiendo un fondo que mostraba una nueva textura, y tarjetas de notificaciones mostrando una apariencia más limpia y suave.

   Por otro lado, los desarrolladores de terceros recibieron la posibilidad de crear sus propios widgets para la pantalla de inicio y ofrecer su descarga a través del Android Market.

   Se introdujo una funcionalidad denominada Live Folders, que consistía en iconos en la pantalla de inicio, en los que, al pulsar sobre ellos, mostrarían información en tiempo real sobre la aplicación en cuestión.

   Grabar vídeo con la cámara del móvil. Entre otras novedades de esta versión, también encontrábamos una nueva barra para hacer zoom en páginas web, imágenes y mapas, la posibilidad de copiar o pegar texto en el navegador nativo. Algunas apps también recibieron nuevas funciones, como YouTube, que a partir de ahora permitiría subir vídeos desde el móvil.

f) **Android 1.6 Donut:** Incluye soporte para diferentes tamaños de pantalla. Muchas de las mejoras que trajo Android 1.6 estaban centradas en aplicaciones del sistema. Se introdujo el soporte para redes CDMA. Fue la primera versión compatible con sistemas de texto a voz, permitiendo así a las aplicaciones reproducir sonido basado en el texto escrito. Android permitiría a los usuarios observar un informe sobre el estado de la batería y ver qué aplicaciones y componentes del sistema estaban consumiendo más energía.

g) **Android 2.0 y 2.1 Éclair**: Introdujo una renovada pantalla de bloqueo, con panel rotatorio que emulaba el dial de los antiguos teléfonos analógicos. Para desbloquear el terminal, habría que deslizar desde el centro del panel hacia abajo, mientras que deslizando a izquierda o derecha se silenciaría el terminal o se abriría la app de teléfono. Introdujo una barra de búsqueda de Google en la parte superior.

   La innovación más relevante fue la navegación GPS en Google Maps. El brillo automático, diferentes escenas en la app de cámara nativa o una aplicación de Facebook preinstalada.

   Se daba comienzo a la familia de dispositivos Nexus, con un teléfono bajo el nombre Nexus One, se introdujo Android 2.1 con las novedades de fondos de pantalla animados. Rediseñar aplicaciones y añadir animaciones en toda la interfaz. En Android 2.1 las aplicaciones de galería, noticias y tiempo, e incluso los widgets preinstalados, pasaron a tener una nueva apariencia mucho más moderna.

   Se reservó para la actualización denominada como Android 2,1 Update 1. Se trataba del sistema **pinch-to-zoom**, que permitía ampliar o reducir el tamaño de imágenes, mapas o páginas web pellizcando sobre la pantalla con dos dedos.

h) **Android 2.2 Froyo:** Froyo fue la versión encargada de marcar el inicio de una nueva era para el sistema operativo, mejorando notablemente el rendimiento gracias al añadido del compilador Dalvik JIT, que convertía en código de bytes de java en lenguaje nativo en tiempo real.

   Novedades como un dock de aplicaciones en la pantalla de inicio, con dos iconos laterales para ejecutar el navegador y la app de teléfono que no eran personalizables, y uno central destinado a abrir el cajón de apps.

Controlar las descargas en Android Market, gracias a un botón de actualizar todas las apps o habilitar las actualizaciones automáticas, así como la llegada de Adobe Flash Player a Android. Se introdujo el soporte para mover aplicaciones a las tarjetas microSD y permitiría controlar el sistema a través de comandos de voz de los usuarios.

i) **Android 2.3 Gingerbread:** Los colores grises y blancos desaparecían para dar cabida a tonos más oscuros, y al verde intenso como color de acento que inundaba toda la interfaz del sistema.

Gracias a una colaboración con Samsung nació el Nexus S, fue el primer Nexus en contar con conectividad NFC. La interfaz que más cambió fue el panel de notificaciones, pasando a tener un fondo oscuro para dar más importancia a los avisos en forma de tarjetas. El color verde también inundó los iconos de la barra de estado, y por primera vez se introdujo el efecto **overscroll** al llegar al final de una lista o menú deslizable.

j) **Honey Comb: Android Versión 3.0**: La versión 3.0 de Android fue la primera y la única en ser exclusiva de tablets. Honeycomb vio la luz en febrero de 2011.

Adoptando las líneas que pasaron a denominarse Holo, y que darían vida a la interfaz de Android durante los próximos años. Dado que se trataba de una edición pensada específicamente para Tablets, ningún Smartphone recibió jamás la actualización a Android 3.0. La interfaz al completo abandonaba el color verde de acento, para dar paso a diferentes tonos de azul brillante, que contrastaban con el negro de los menús del sistema y el fondo de las aplicaciones y la pantalla de bloqueo volvió a cambiar.

Fue la carencia de botones lo que llevó a Google a incorporar una barra de navegación virtual en pantalla, botonera virtual que incorporaría los botones de menú, home y atrás. Google quiso potenciar la multitarea a base de un nuevo menú de aplicaciones recientes, que mostraba una miniatura de cada aplicación y su contenido.
Se introdujo un panel de ajustes rápidos, accesible desde la barra del sistema.

k) **Ice Cream Sandwich: Android Versión 4.0:** El cajón de aplicaciones ahora estaba dividido en dos pestañas, de apps y widgets, y el panel de notificaciones pasaba a tener un fondo transparente. El launcher ahora permitía añadir carpetas de aplicaciones, o redimensionar los widgets para ahorrar espacio u ocupar más.

Android 4.0 Ice Cream Sandwich fue la primera versión con soporte para Android Beam, un sistema que permitía compartir contenido entre dos dispositivos Android juntando sus espaldas, gracias a la conexión NFC.

Un nuevo enfoque para sus servicios de contenido multimedia, que pasaban a formar parte de la familia Google Play. De este modo, Android Market pasó a ser Google Play Store, Google Books pasó a llamarse Google Play Books, Google Music ahora sería Google Play Music, y aparecía un servicio de series y películas denominado Google Play Movies & TV.

l) **Android 4.1 y 4.2 Jelly Bean:** Project Butter. Consistía en un proyecto a través del cual los ingenieros del equipo de Android lograron que las animaciones del sistema se mostrasen a 30 fps, intentando así ponerse a la altura de iOS en cuanto a sensación de fluidez general. Se modificó el panel de notificaciones, que ahora mostraba un reloj digital de gran tamaño en la parte superior, y por primera vez las notificaciones eran ampliables para observar su contenido.

Añadir diferentes cuentas de usuario a un mismo dispositivo.
Google Now consistía en un panel repleto de tarjetas con información relevante para los usuarios, que irían apareciendo a lo largo del día, antes incluso de que el usuario necesitase consultar esa información.

En Android 4.1 se implementaron los servicios de Google Play, permitiendo así a Google actualizar las características esenciales del sistema.

En cuanto a los cambios de Android 4.2, nos quedamos con la posibilidad de añadir widgets a la pantalla de bloqueo, y una nueva interfaz destinada a Tablets.

m) **Android 4.4 KitKat:** Sistema denominado Proyect Svelte, que se transformaba en una importante reducción en el uso de memoria RAM, gracias a diferentes optimizaciones que permitían al sistema operativo ejecutarse con una cantidad de memoria de solo 340 MB.

La novedad más llamativa fue el comando de voz "Ok Google", que permitiría ejecutar la búsqueda de Google sin necesidad de tocar el teléfono.

KitKat se olvidaba de los acentos azules de anteriores versiones del sistema, para dar paso al blanco, dando así un nuevo enfoque al diseño Holo. Por primera vez, además, Google Now pasaba a estar integrado en el launcher. Apareció el modo inmersivo que ocultaba las barras del sistema.

n) **Android 5 Lollipop:** Material Design fue el punto de inflexión más grande de la historia de Android. La idea principal era que el sistema operativo, y las aplicaciones, webs y diferentes plataformas, tuviesen una misma apariencia, cuyas interfaces estuvieran basadas en elementos físicos metafóricos como la tinta y el papel, incluyendo sombras, texturas y elevaciones virtuales sobre un lienzo tridimensional.

Nuevo panel de ajustes rápidos, una barra de navegación más minimalista, con un color principal que ocupaba las cabeceras de cada app al completo.

Introdujo la máquina virtual ART, ART estaba escrito desde cero, y estaba pensado para mejorar la velocidad de apertura de aplicaciones y optimizar el consumo de memoria en hardware moderno, notable aumento en el rendimiento, fue el soporte para aplicaciones java de 64 bits.

Mejorar la duración de la batería en los dispositivos, gracias al sistema Project Volta. Herramientas de rastreo de consumo de energía, para ver qué actividades, tareas y servicios estaban consumiendo más batería.

Las notificaciones fueron rediseñadas, permitiendo a los usuarios interactuar con ellas desde la pantalla de bloqueo en forma de tarjetas, y realizar acciones rápidas como contestar mensajes, posponer tareas o recordatorios y mucho más.

Control sobre las notificaciones, gracias a los modos de "Prioridad", "Sonido para todo" y "Completamente silencioso".

o) **Android 6 Marshmallow:** Formato del cajón de aplicaciones, que regresaba a la lista con scroll vertical original, acompañada de un selector rápido para encontrar las aplicaciones de forma más sencilla.
Now on Tap, un sistema que permitía acceder, a través de una pulsación larga sobre el botón home, a un menú que identificaba el contenido que aparecía sobre la pantalla.
Sistema de permisos granular, que permitía al usuario observar y activar los diferentes permisos requeridos por las aplicaciones de forma manual.

Adoptable Storage, que transformaba la tarjeta sd externa en una ampliación de la memoria interna, en la cual poder instalar aplicaciones o almacenar datos de forma sencilla.
Formato Peek, que consistían en alertas que se mostraban en forma de carteles sobre la parte superior de la pantalla.
Introdujo el sistema DOZE, el dispositivo se encontraba estático en un lugar, desconectado y con la pantalla apagada, para habilitar un modo de bajo rendimiento que restringía los permisos de fondo.
Actualizaciones de seguridad mensuales, gracias a la inclusión de Fingerprint API, que implantaba la compatibilidad con lectores de huellas dactilares.

p) **Nougat Versión 7.0**: Se incorporaban nuevas funcionalidades en el sistema como la ejecución de dos aplicaciones en pantalla partida, a través de una pulsación larga sobre el botón de multitarea. Se añade la posibilidad de contestar a los mensajes desde las notificaciones.

La aparición de accesos directos rápidos que permitían acceder a funciones rápidas de las aplicaciones a través de una pulsación rápida desde el icono.

Mejoraba la forma en la que los dispositivos recibían actualizaciones OTA, función prestada de Chrome OS, que llevaba el nombre de Seamless Updates y consistía en la división del almacenamiento del sistema en dos particiones diferentes. La primera estaría orientada al almacenamiento de los datos del sistema e información del usuario, mientras que la secundaria estaría deshabitada por defecto, y solo sería usada a la hora de recibir una actualización.

La nueva versión del sistema se instala en la segunda partición mientras que el usuario puede seguir utilizando la principal.

q) **Android 8 Oreo:** Gestión de las notificaciones, mejorando el orden de estas gracias a distintos niveles de prioridad, y ofreciendo al usuario la posibilidad de posponer avisos o elegir manualmente qué tipo de notificaciones de cada aplicación quieren recibir, gracias a los canales de notificaciones.

Formatos de aviso como los Notification Dots o Notification Badges, dos sistemas que permitían saber si teníamos algún aviso directamente mirando al icono de la aplicación, o ver su contenido a través de una pulsación larga sobre este.

Introdujo el modo Picture in Picture, que permite mostrar diferentes tipos de contenido de vídeos, mapas o videollamadas en una pequeña ventana flotante, mientras el usuario sigue utilizando otras aplicaciones.

Project Treble. Su objetivo es el de modularizar el propio sistema operativo, separando los drivers y el resto de código relacionado con el hardware de los dispositivos "vendor", del propio Android. Así, se consigue que las actualizaciones del sistema sean más sencillas, rápidas y asequibles de desarrollar y liberar por parte de los diferentes fabricantes de smartphones.

Android 8.1, la principal novedad de esta versión incremental fue la llegada de la API para redes neuronales, que permitía a los desarrolladores implementar sistemas de IA y Machine Learning en sus apps.

r) **Android 9.0 Pie:** La primera edición de Android que introduce un nuevo tipo de navegación por gestos.
Esta versión es la encargada de implantar una renovada filosofía de diseño que ha sido bautizada como Google Material Theme, y que promete aportar una mayor coherencia a toda la interfaz, con fondos más claros y tarjetas cuyas esquinas son más redondeadas.

Tipografía Product Sans en aún más componentes de la interfaz, ofrece a los fabricantes una mayor flexibilidad a la hora de desarrollar y diseñar sus apps.

Un sistema de posicionamiento en interiores a través de la tecnología Wi-Fi RTT, soporte nativo para los codecs HDR BP9 y HEIF, restricciones de acceso al micrófono, cámara y sensores a las aplicaciones en segundo plano, compatibilidad con nuevos formatos de pantalla –véase, pantallas con "notch"– o mejoras en el rendimiento gracias a ART y de ahorro de energía a través de DOZE.

Nuevo panel de ajustes rápidos, respuestas rápidas a mensajes desde las notificaciones, el primer editor de capturas de pantalla nativo o un control de volumen mejorado.

También se introducen una colección de novedades basadas en inteligencia artificial, como el brillo adaptativo o sistemas de ahorro de batería inteligentes.

s) **Android 10:** Novedades y cambios importantes, que se centran principalmente en mejorar la privacidad y la seguridad de los usuarios otorgando un mayor control sobre los permisos que obtienen las aplicaciones, como el de ubicación o el acceso al portapapeles del sistema.

Android Q tiene seis versiones beta diferente, Será en agosto de 2019 cuando Google libere la versión definitiva de la nueva edición del sistema operativo.

La segunda beta de Android Q trajo consigo un mayor número de novedades de cara al usuario, y de entre todas destacaban las burbujas flotantes de notificaciones compatibles con todas las aplicaciones y trabajaba en perfeccionar la navegación por gestos nativa de Android.

La tercera beta se encontraba el esperado tema oscuro, ahora sí disponible para ser activado desde los ajustes del sistema, y con la posibilidad de forzar la nueva apariencia en cualquier aplicación instalada en el dispositivo. También se renovaba por completo el sistema de gestos, para pasar a ser más intuitivo y fácil de usar.

Introdujo también mejoras en términos de accesibilidad, como la generación automática de subtítulos en cualquier contenido en vídeo, independientemente de la aplicación en la que se reprodujese.

La más importante fue Mainline enviar actualizaciones de seguridad a los teléfonos a través de Google Play Store.

1.6 Componentes de una aplicación en Android

Para diseñar una aplicación en Android, es necesario tener claros los elementos que la componen y la funcionalidad de cada uno de ellos. Ya hemos visto el ejemplo del "Hola Android", por lo que podemos intuir algunos de ellos. Uno de los aspectos más importantes a tener en cuenta es su funcionamiento. Android trabaja en Linux, y cada aplicación utiliza un proceso propio. Se distinguen por el ID, un identificador para que solo ella tenga acceso a sus archivos. Los dispositivos tienen un único foco, la ejecución principal, que es la aplicación que está visible en la pantalla, pero puede tener varias aplicaciones en un segundo plano, cada una con su propia pila de tareas. La pila de tareas es la secuencia de ejecución de procesos en Android. Se componen de actividades que se van apilando según son invocadas, y solo pueden terminarse cuando las tareas que tiene encima están terminadas, o cuando el sistema las destruye porque necesita memoria, por lo que tienen que estar preparadas para terminar en cualquier momento. El sistema siempre eliminará la actividad que lleve más tiempo parada. En caso de que el sistema necesitase mucha memoria, si la aplicación no está en el foco, puede ser eliminada por completo a excepción de su actividad principal

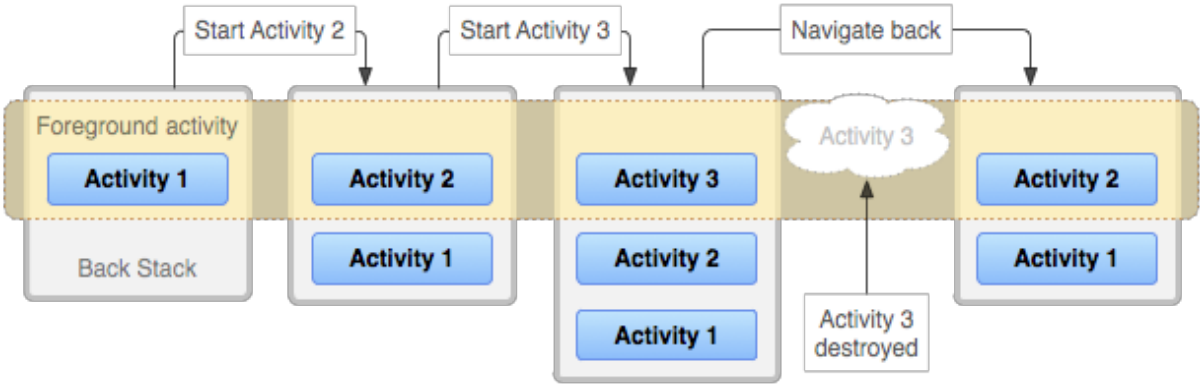


Figura 2. Pila de actividades Android

Fuente: Developers Android (2020).

Una representación de cómo cada nueva actividad en una tarea agrega un elemento a la pila de actividades cuando el usuario presiona el botón Atrás, se elimina la actividad actual y se reanuda la anterior.

Una de las características principales del diseño en Android es la reutilización de componentes entre las aplicaciones, es decir, dos aplicaciones diferentes pueden utilizar una misma componente, aunque esté en otra aplicación para así, evitar la repetición innecesaria de código, y la consiguiente ocupación de espacio. Los componentes son los elementos básicos con los que se construyen el proyecto. Hay cuatro tipos, pero las aplicaciones se componen principalmente de actividades. Habrá tantas actividades como ventanas distintas tenga la aplicación. Sin embargo, por si solos, los componentes no pueden hacer funcionar una aplicación. Para ello están los intents. Una representación de cómo cada nueva actividad en una tarea agrega un elemento a la pila de actividades Cuando el usuario presiona el botón Atrás, se elimina la actividad actual y se reanuda la anterior.

Todos ellos deben declararse en el AndroidManifest.xml (junto con otros elementos que se mostrarán después) con el mismo nombre que lleve la clase asociada. Por ejemplo, la clase MainActivity, será definida en el Android Manifest con el mismo nombre.

CAPÍTULO II. VULNERABILIDADES DEL SISTEMA OPERATIVO ANDROID

En este capítulo se describen las vulnerabilidades identificadas en el estado del arte del sistema operativo Android.

¿Qué es una vulnerabilidad?

Es una debilidad o fallo en un sistema de información que pone en riesgo la seguridad de la información pudiendo permitir que un atacante pueda comprometer la integridad, disponibilidad o confidencialidad de la misma, por lo que es necesario encontrarlas y eliminarlas lo antes posible.

¿Qué es una amenaza?

Una amenaza de seguridad es la posibilidad de ocurrencia de un evento que podría explotar una vulnerabilidad para romper la seguridad de un sistema, de forma intencional o accidental, con el objetivo de afectar a la confidencialidad, integridad o disponibilidad del sistema.

Las amenazas de seguridad móvil pueden ser físicas o basadas en software y pueden comprometer los datos de los teléfonos inteligentes, tabletas u otros dispositivos similares.

2.1 TAXONOMÍA DE LAS VULNERABILIDADES DE ANDROID

El estudio en general consiste en 660 vulnerabilidades detectadas, extraídas de los boletines de seguridad de Android del sitio web oficial, de agosto de 2015 a noviembre de 2016 se encontraron 16 boletines publicados por Google donde se detectaron 564 vulnerabilidades, se realizó una taxonomía de las vulnerabilidades detectadas

Algunas de las vulnerabilidades no reservadas (que figuran en los boletines) no aparecen etiquetadas como relacionadas con Android en el sitio web de CVE details porque afectan al Kernel de Linux o a componentes de terceros (por ejemplo, controladores). Encontramos 31 vulnerabilidades en los boletines no etiquetados como relacionados con Android en los detalles de CVE.

Por lo tanto, utilizando los identificadores CVE de los boletines que directamente recabaron la información de los detalles de CVE, sin confiar en el filtro Android. Al final, obtuvimos información para un total de 660 vulnerabilidades (629 etiquetadas relacionadas con Android más 31 no etiquetados con Android, pero enumerados en los boletines).

Thomas et al. Extrajo las actualizaciones del sistema operativo instaladas en 20,000 dispositivos Android para medir el tiempo de entrega de las actualizaciones de seguridad para 11 vulnerabilidades, y para establecer un modelo de dispositivos inseguros; los resultados sugieren que, en promedio, 87.7% de los dispositivos están expuestos al menos una de las vulnerabilidades analizadas.

Investigó la vulnerabilidad con identificador CVE-2012-6636 en la interfaz JavaScript de la API de Java en la vista Web; 102,000 se analizaron estáticamente para medir el número de aplicaciones en las que la vulnerabilidad podía ser explotada. Además, la vida útil de la vulnerabilidad se analizó utilizando un enfoque similar a las actualizaciones del sistema operativo instaladas.

Jiménez et al. Analizaron 32 vulnerabilidades de la base de datos CVE para identificar los problemas, componentes implicados, la complejidad del código de los parches y así como los métodos o funciones de código implicados en la vulnerabilidad.

El estudio presentado en este documento es complementario a estudios anteriores en el sentido de que se analiza un conjunto más grande de vulnerabilidades(extraídos de CVE) (660) y se incluyen diferentes perspectivas en el estudio, como el tiempo de supervivencia de las vulnerabilidades, subsistemas y componentes del Sistema Operativo Android implicados en las vulnerabilidades, una amplia taxonomía de problemas de seguridad basados en la jerarquía de vulnerabilidades de Enumeración de debilidad común (CWE), y una lista de lecciones aprendidas orientado al sistema operativo Android y aplicaciones de los programadores.

La Figura 3 nos muestra la información que se considera para realizar la clasificación de las vulnerabilidades que se presentan

Cve ld: La identificación de la vulnerabilidad.

Puntuación: Una puntuación de 0 a 10 que indica la gravedad de la vulnerabilidad.

Descripción: Una descripción textual de la vulnerabilidad.

Enlaces de parche: [Hay una parada de parche (s), destinado a fijar esta vulnerabilidad].

Tipo: [El tipo de vulnerabilidad deducido automáticamente].

Impacto confidencial: Posibilidad de acceder a la información (Ninguno / parcial / completa).

Impacto de integridad: Capacidad de modificar la información en el dispositivo (Ninguno / parcial / completa).

Impacto de la disponibilidad: Impacto en la disponibilidad del dispositivo (Ninguno / parcial / completa).

Figura 3. Información almacenada para las vulnerabilidades extraídas de CVE detalles

El estudio aborda tres preguntas de investigación fundamentales que se denotan como (RQ):

RQ1: ¿Qué tipos de vulnerabilidades de seguridad afectan a Android?, Impacto sobre confidencialidad, integridad y disponibilidad. El tipo de vulnerabilidad de seguridad que afectan, así como controladores de hardware, aplicaciones, etc.

RQ2: ¿Cuáles son los subsistemas más afectados de Android?, se centra en la arquitectura de Android, lo de terceros lo deja de lado, su objetivo es dar a conocer a los desarrolladores de aplicaciones, ¿Cuáles son los servicios con más riesgo en las APIS?, el mejorar la validación de ellas, desarrollar herramientas para detección de vulnerabilidades y codificación segura.

RQ3: ¿Cuánto tiempo se tardan las vulnerabilidades de seguridad en Android?, Se realiza un estudio de la supervivencia de las vulnerabilidades, así como el número de días desde la introducción de la vulnerabilidad, una gran supervivencia de la vulnerabilidad indicaría el impulso de herramientas para identificar la prevalencia.

Dos autores (UN1 Y UN2) analizaron la información de las vulnerabilidades documentadas en la NVD (Base de datos nacional de vulnerabilidades), en donde en primer lugar se realiza una clasificación de alto nivel (la vulnerabilidad que afecta los componentes del Sistema Operativo desarrollados por Google o terceros.

Se da una categoría más de grano fino, para identificar la capa de arquitectura afectada por ejemplo el tiempo de ejecución, subsistema Dalvik).

Los dos autores analizaron de manera manual la mitad de las 660 vulnerabilidades en donde se analiza directamente, los tipos de vulnerabilidad, las capas afectadas de Android y los subsistemas afectados de igual forma.

Los dos autores discutieron sobre 47 Vulnerabilidades en las cuales no estaban de acuerdo que hacen un 7% del 100%, donde posteriormente se llegó a un acuerdo para la clasificación mostrada en la taxonomía de las vulnerabilidades.

Sobre las preguntas de investigación estas son las respuestas a dichas preguntas:

RQ1: Mediante la representación de la taxonomía de los tipos de vulnerabilidades son identificadas en el análisis manual, no se validan manualmente ya que se extraen del CVE de la base de datos NVD.

RQ2: Se muestra un código de colores que muestra la distribución de las vulnerabilidades a través de los subsistemas de Android.

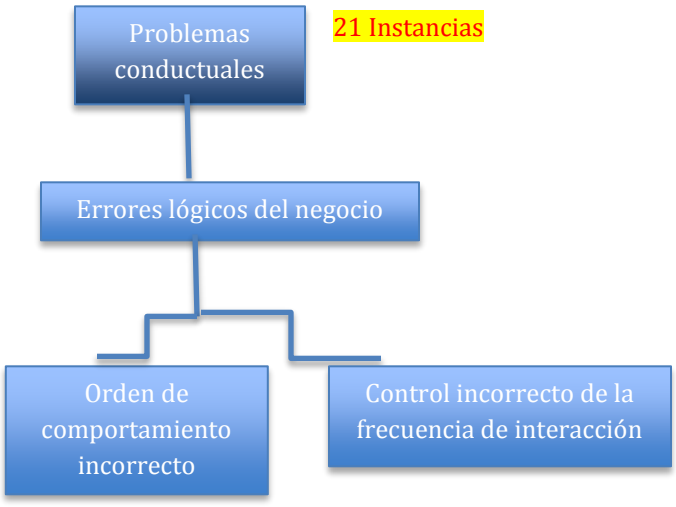
RQ3: Para responder a RQ3 necesitamos información que no está disponible en la CVE Detalles de fuente de datos. En particular, es necesario identificar las confirmaciones en la que se ha introducido cada vulnerabilidad. En cuanto a la confirmación que se fijan cada vulnerabilidad, que se ve desde los Boletines de seguridad de Android publicado cada mes.

La taxonomía muestra 660 vulnerabilidades, solo se informa la clasificación de 510 vulnerabilidades ya que las vulnerabilidades se vieron durante nuestro análisis manual.

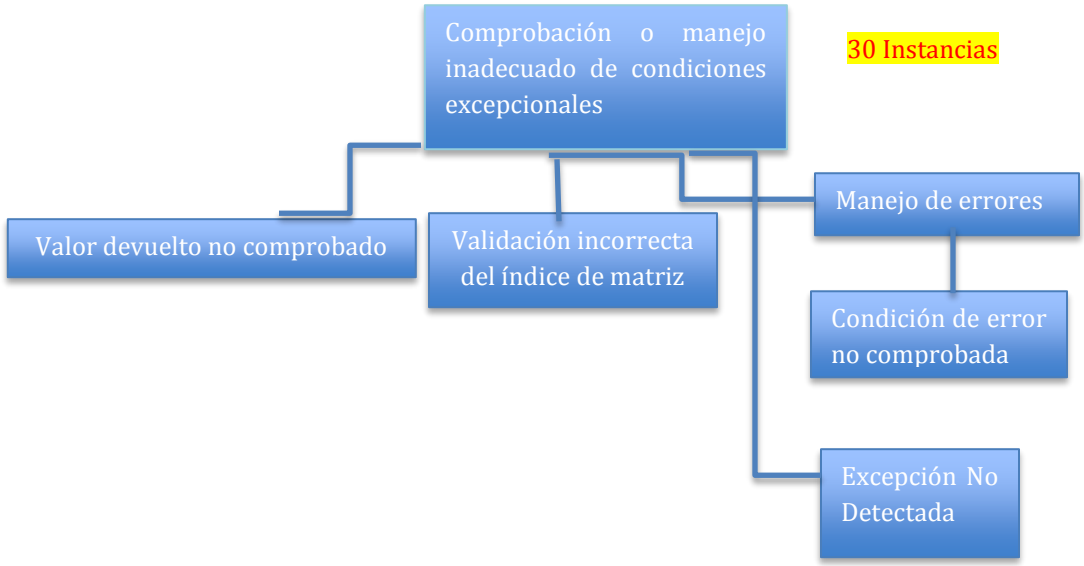
Las vulnerabilidades que afectan con mayor frecuencia, son a la memoria, con 103 casos (20%). Vulnerabilidades relacionadas con permisos, privilegios y control de acceso con 58 casos (11%). La validación de entrada inadecuada con 51 casos (10%), vulnerabilidades causadas por una validación que falta o que afectan el flujo del programa.

A continuación se muestra la taxonomía por ramas de la clasificación de las vulnerabilidades del Sistema Operativo Android:

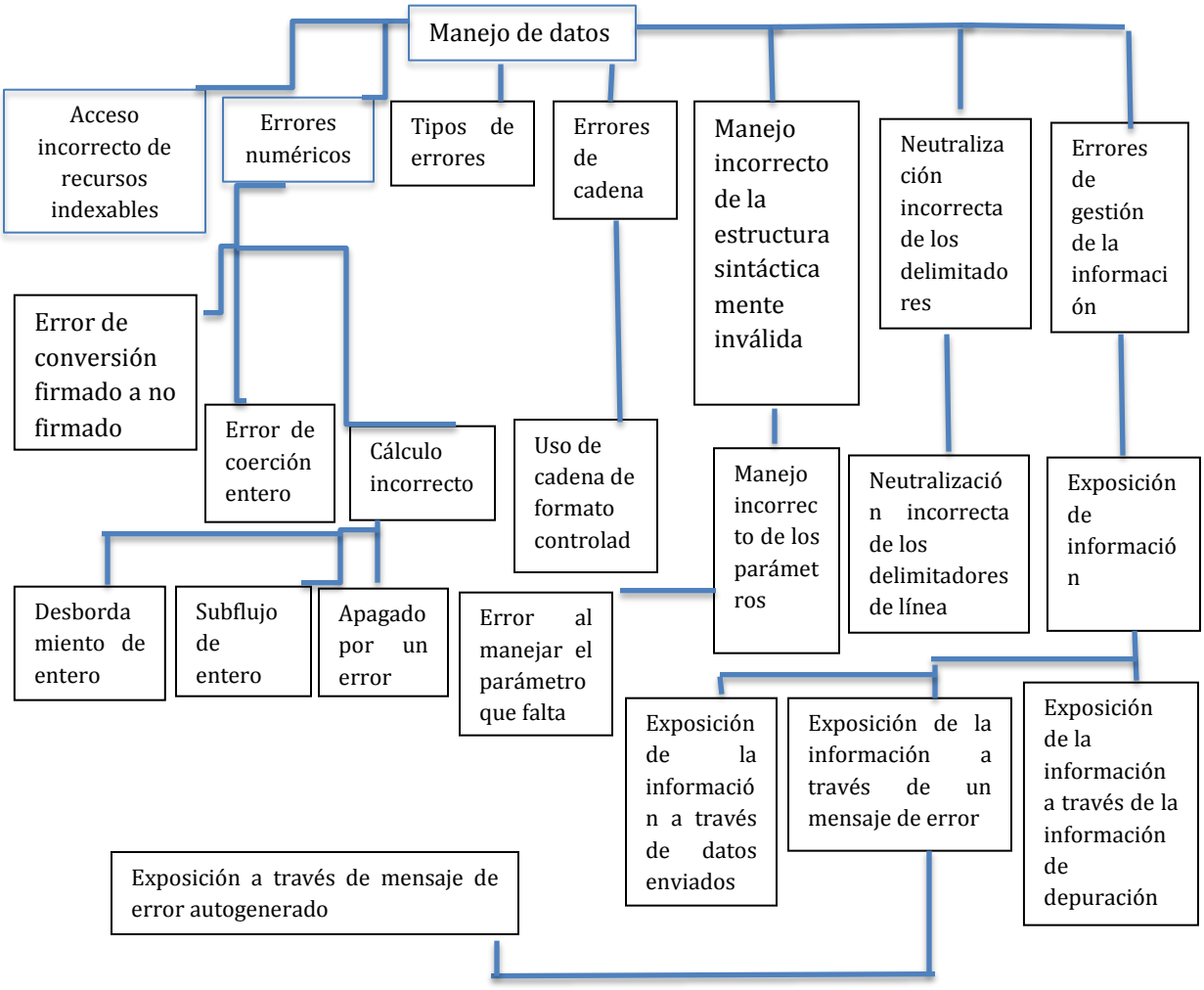
a) Rama 1



b) **Rama 2**

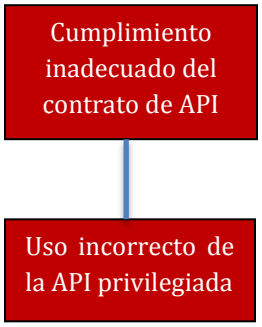


c) **Rama 3**



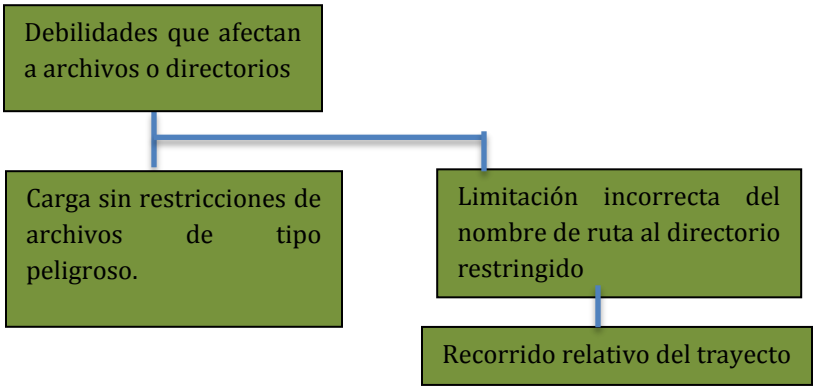
d) Rama 4

4 Instancias

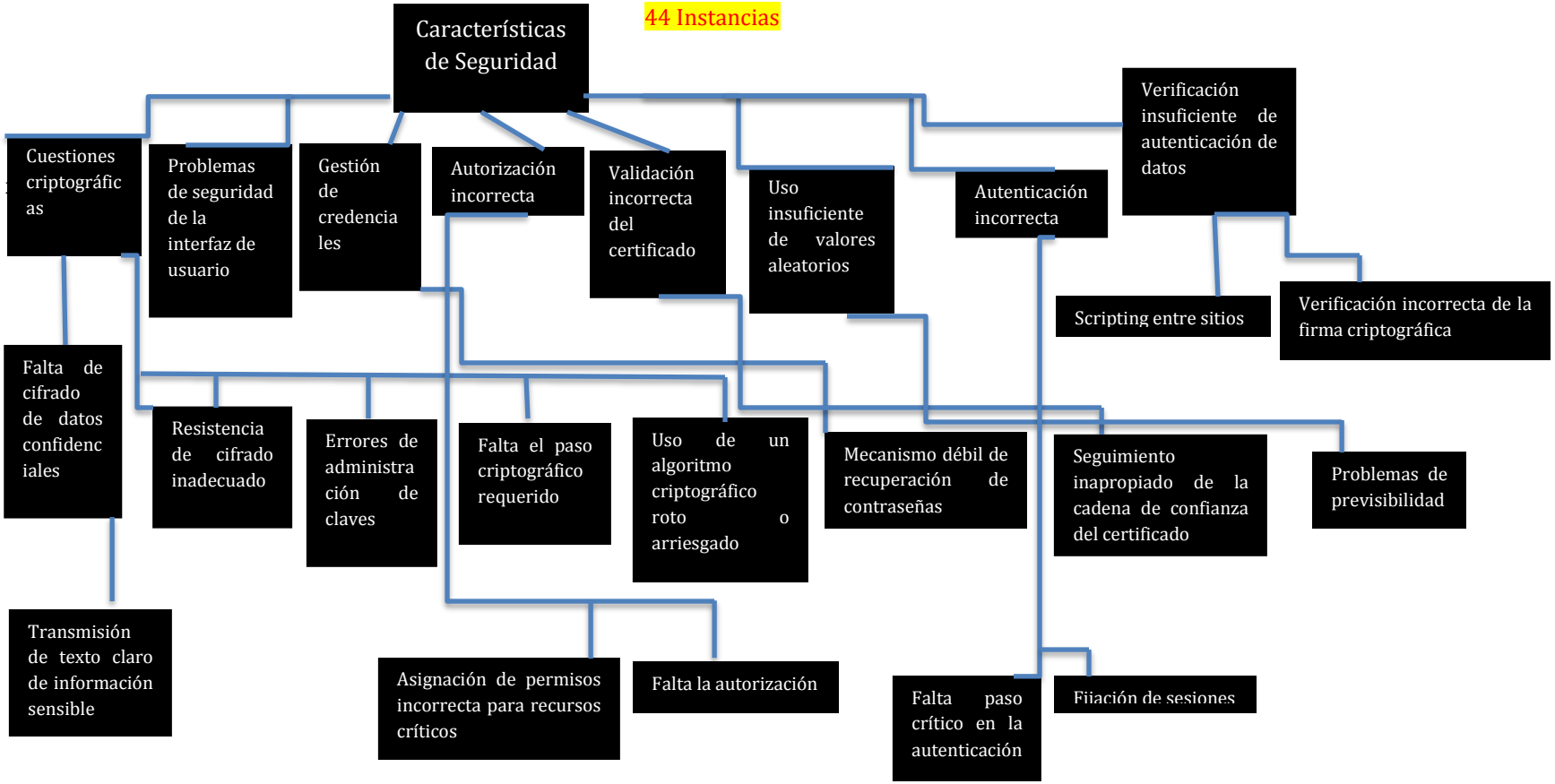


e) **Rama 5**

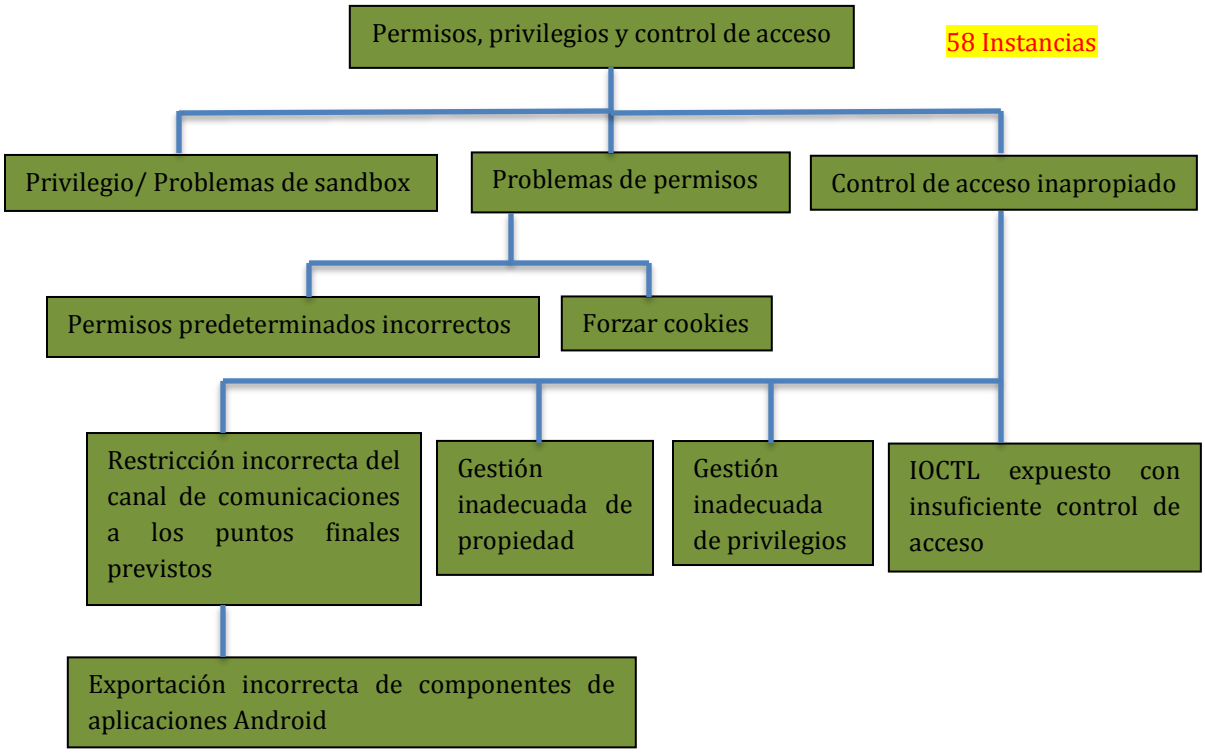
3 Instancias



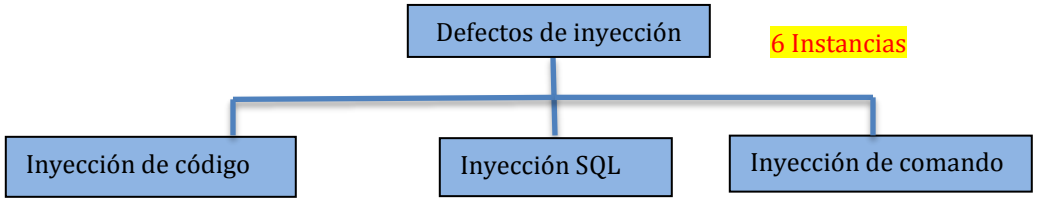
f) Rama 6


Características de Seguridad
44 Instancias
Cuestiones criptográficas
Problemas de seguridad de la interfaz de usuario
Gestión de credenciales
Autorización incorrecta
Validación incorrecta del certificado
Uso insuficiente de valores aleatorios
Autenticación incorrecta
Verificación insuficiente de autenticación de datos
Scripting entre sitios
Verificación incorrecta de la firma criptográfica
Falta de cifrado de datos confidenciales
Resistencia de cifrado inadecuado
Errores de administración de claves
Falta el paso criptográfico requerido
Uso de un algoritmo criptográfico roto o arriesgado
Mecanismo débil de recuperación de contraseñas
Seguimiento inapropiado de la cadena de confianza del certificado
Problemas de previsibilidad
Transmisión de texto claro de información sensible
Asignación de permisos incorrecta para recursos críticos
Falta la autorización
Falta paso crítico en la autenticación
Fijación de sesiones

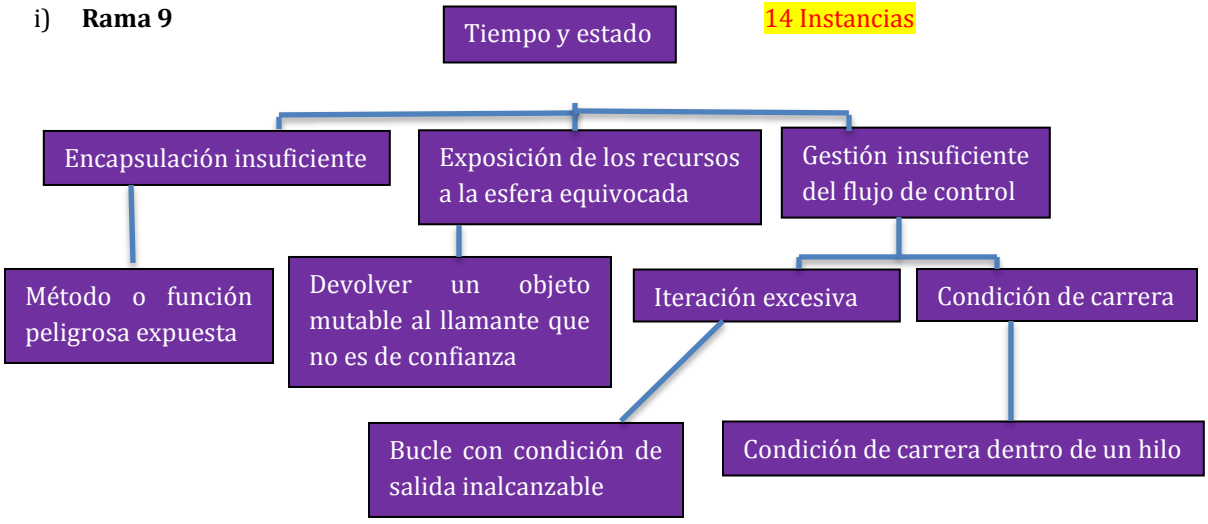
g) Rama 7



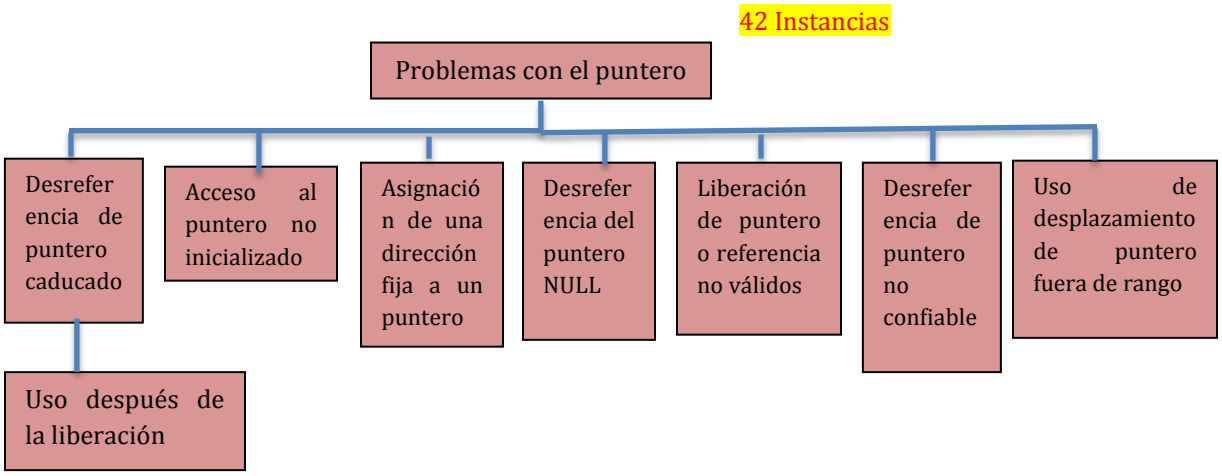
h) Rama 8



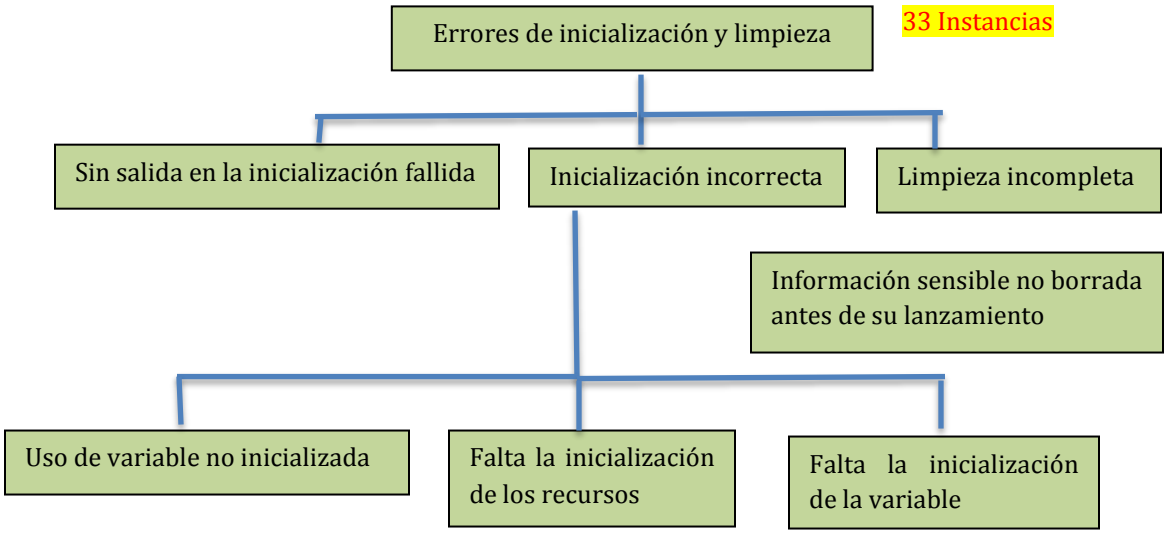
i) **Rama 9**



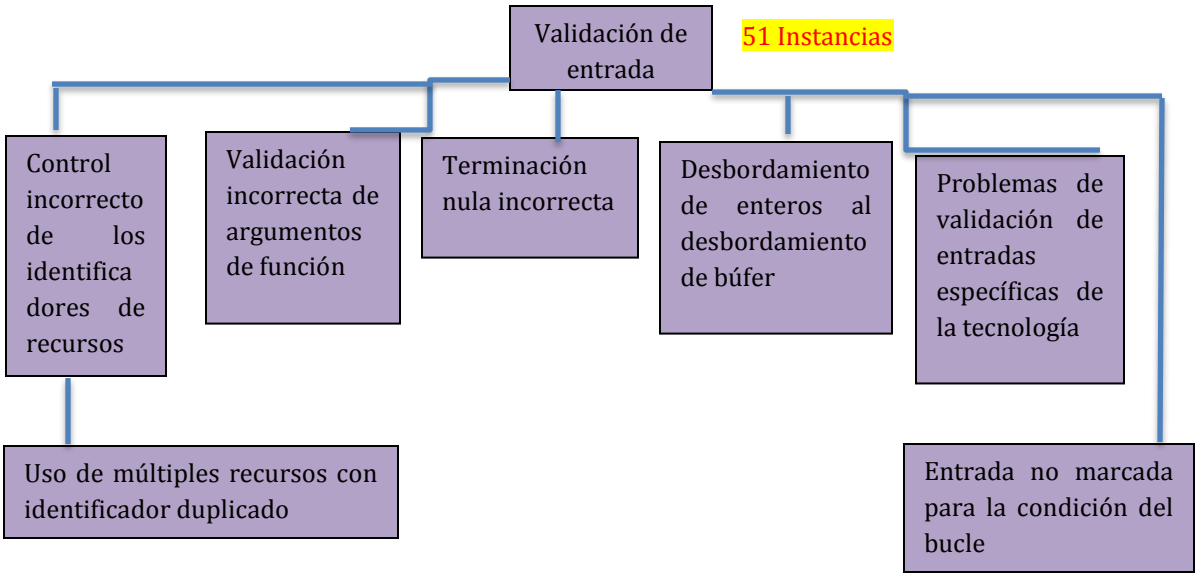
j) Rama 10



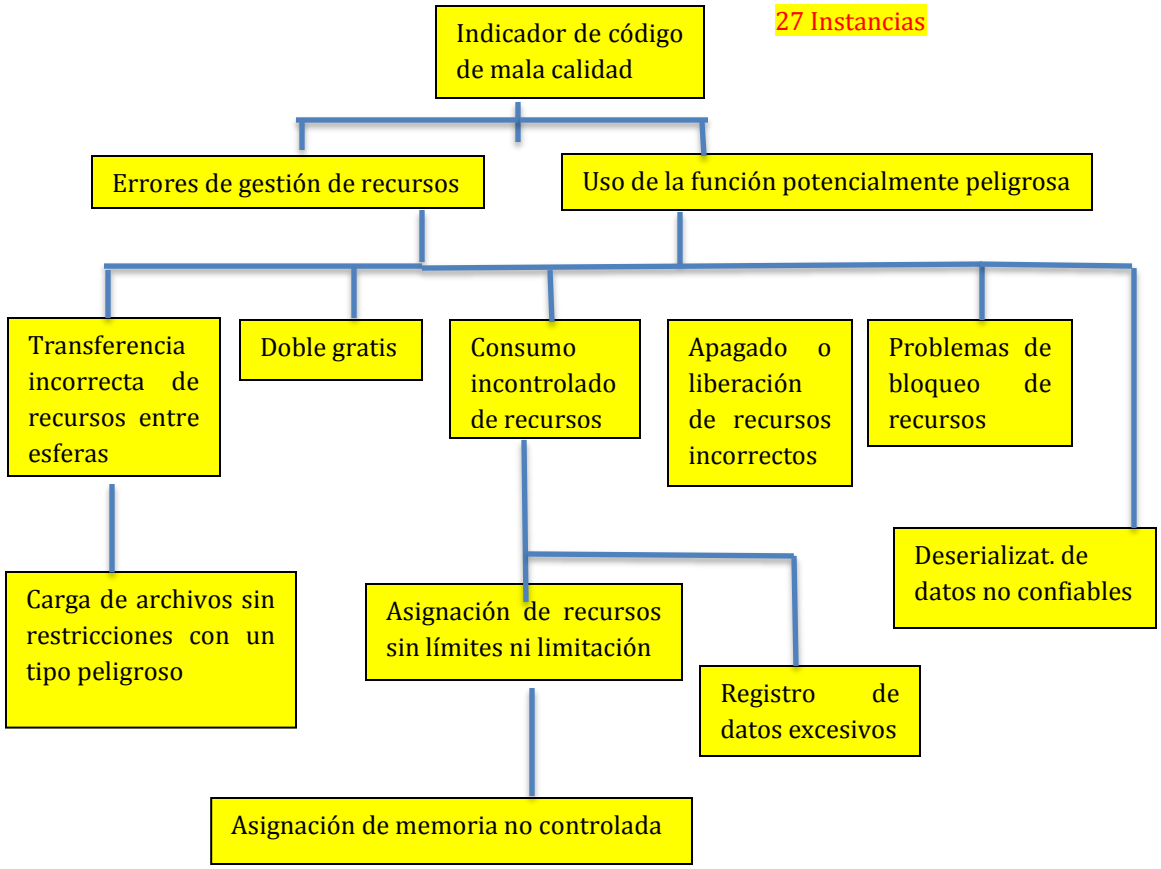
k) Rama 11



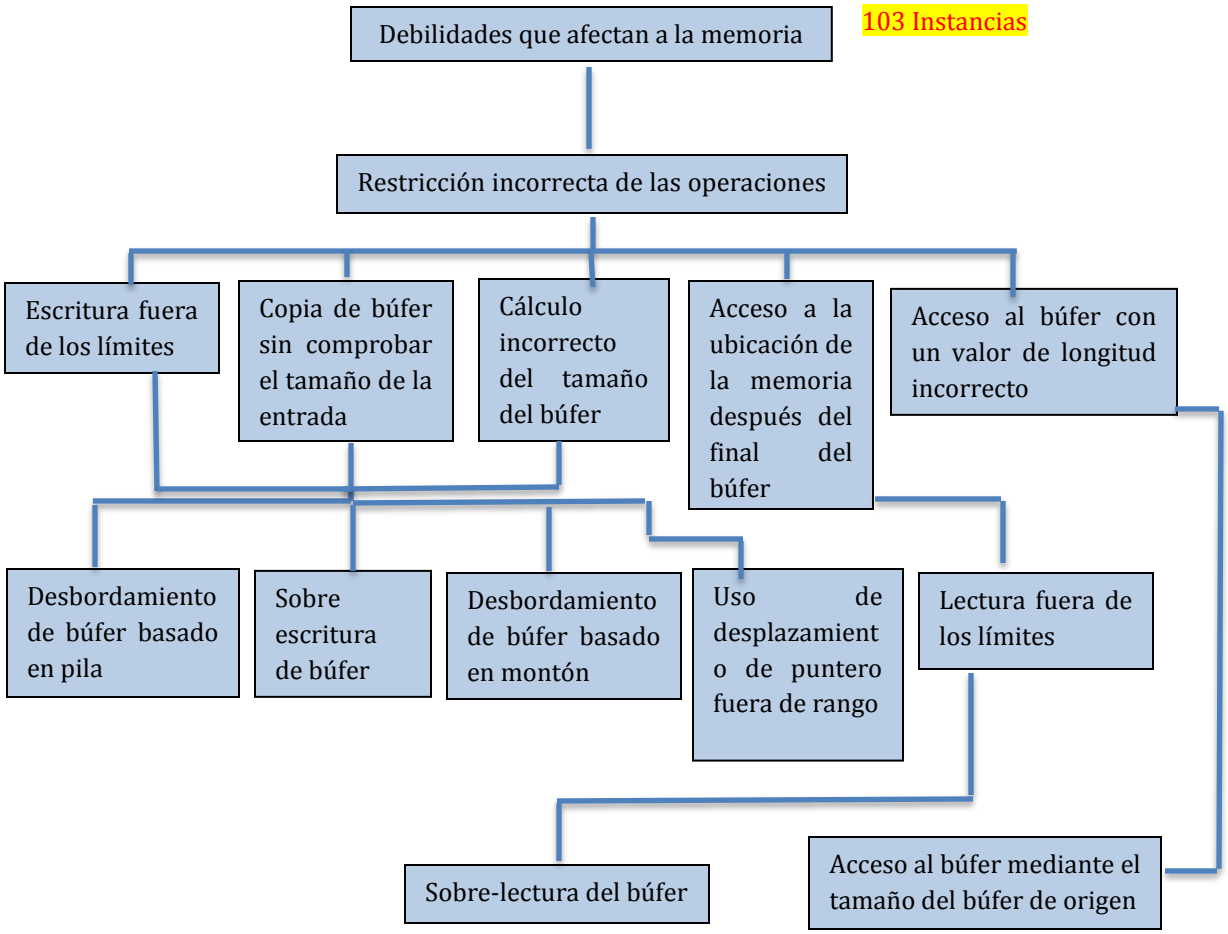
l) Rama 12



m) Rama 13



n) Rama 14



Por último, otras vulnerabilidades menos difusas son las que caen en las categorías: Indicador de código de mala calidad (27 instancias), problemas de comportamiento (21), tiempo y estado (14), defectos de inyección de código (6), cumplimiento incorrecto del contrato de API (4) y algunas que afectan a archivos o directorios (3).

A continuación en la Figura 4 se informa sobre el impacto de la confidencialidad, integridad y disponibilidad de las vulnerabilidades estudiadas en la taxonomía. El color verde nos indica que no hay impacto, el color naranja que hay un impacto parcial y el rojo que existe un impacto completo.

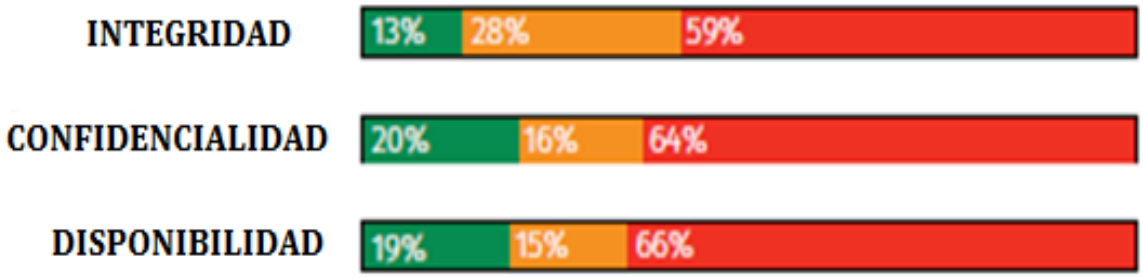


Figura 4. RQ1: impacto de la explotación de la vulnerabilidad

La mayoría de las vulnerabilidades relacionadas con Android pueden comprometer seriamente la confidencialidad e integridad de la información almacenada en el dispositivo y pueden provocar un agotamiento total de los recursos del dispositivo. Esto pone de relieve la necesidad de técnicas y herramientas que admitan la detección de vulnerabilidades relacionadas con Android en diferentes etapas: Desarrollo, envío al mercado y ejecución en el dispositivo.

La Figura 5 representa (utilizando un estilo de mapa de color), de manera manual las capas y subsistemas identificados del sistema operativo Android que impactaron con 634 vulnerabilidades. Para la construcción del mapa de color, desde 660 vulnerabilidades, hemos excluido 26 en los que no se fue capaz de identificar manualmente la capa.

Para el mapa de color, utilizamos dos esquemas de color: blanco a rojo para las capas, con blanco que representa el valor más bajo y el rojo el más alto; y blanco a amarillo para los subsistemas (es decir, cajas internas), con amarillo completo que significa que un subsistema es responsable del 100% de las vulnerabilidades en la capa correspondiente. Tenga en cuenta que los colores de los subsistemas se normalizan sobre la base de las vulnerabilidades totales que afectan a una capa.

Linux kernel es la capa más afectada, con 261 de las 634 vulnerabilidades (41%). Vale la pena notar que el proyecto Android Open Source incluye modificaciones al kernel original para habilitar las funciones móviles.

A continuación en la Figura 5 se muestra el mapa de vulnerabilidades por colores de acuerdo a las capas o subsistemas de Android que se vieron afectados:

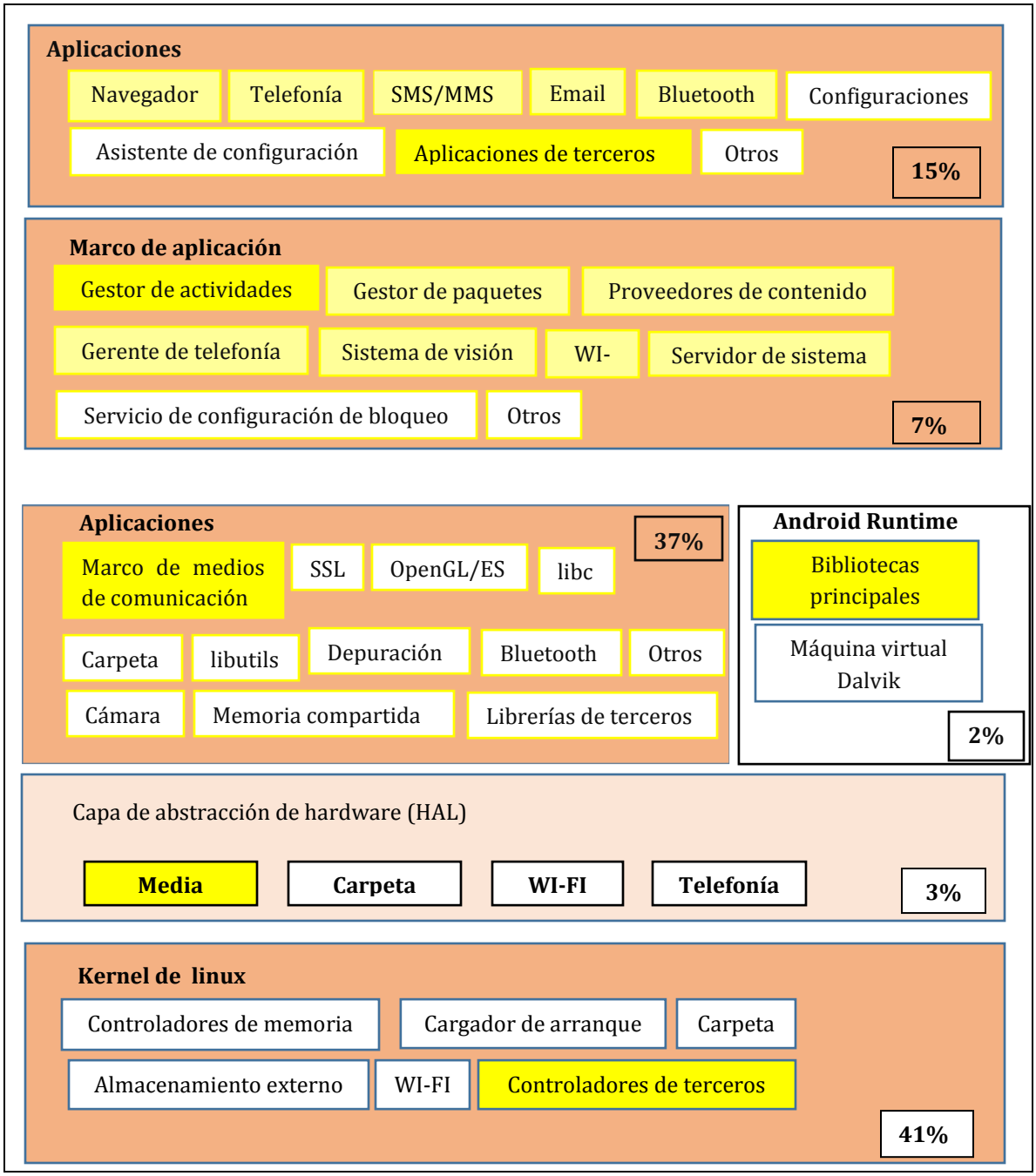


Figura 5. Mapa de vulnerabilidades en las capas/subsistemas de Android.

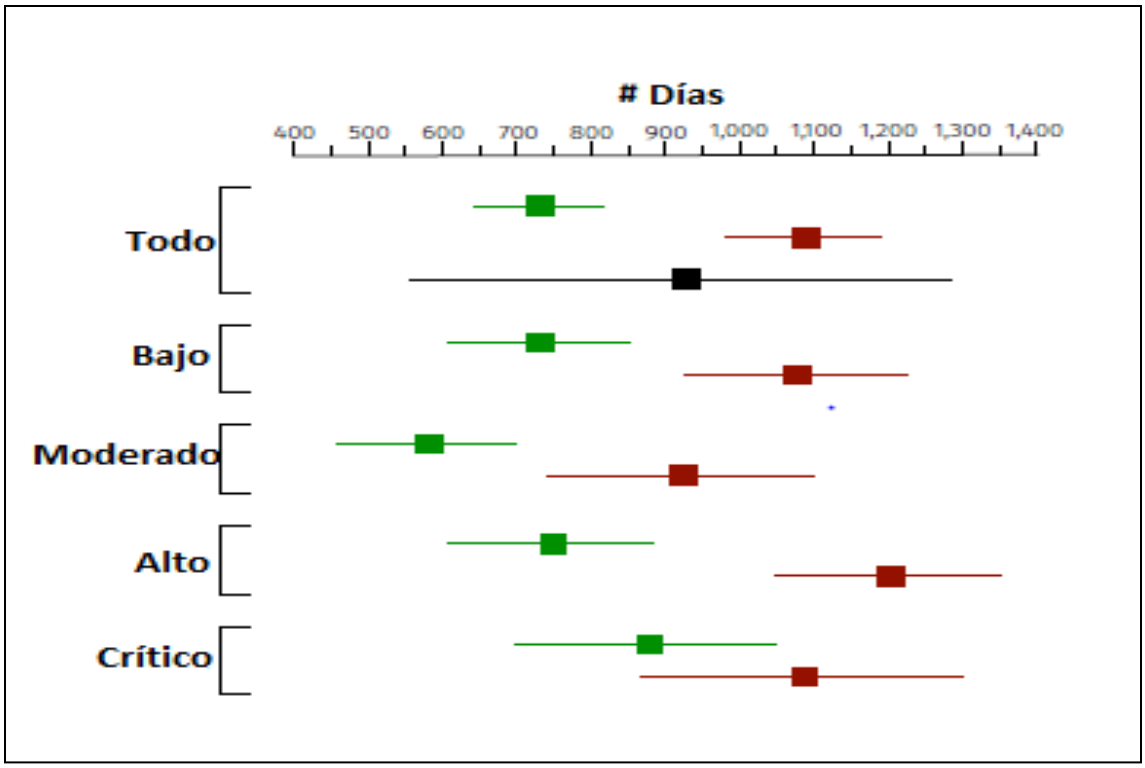


Figura 6. RQ3: Supervivencia en días de vulnerabilidades relacionadas con Android. Verde (rojo) representa estimaciones mínimas (máximas) en el intervalo de confianza del 95%. Negro se muestran los resultados del modelo de efecto aleatorio

2.2 CATALOGO DE VULNERABILIDADES DE SEGURIDAD EN ANDROID

(Özkan, 2019a) es una interfaz web fácil de usar para consultar los datos de vulnerabilidad a través del identificador (CVE), proporcionando proveedores, productos, versiones y las entradas de las vulnerabilidades relacionadas con ellos.

Los datos de vulnerabilidad de CVE se toman de las fuentes XML de la base de datos de vulnerabilidad nacional (NVD) proporcionadas por el Instituto Nacional de Estándares y Tecnología. Además de los datos CVE de NVD, también se publican datos adicionales de varias fuentes, como exploits de *www.exploit-db.com*, módulos Metasploit , declaraciones y datos adicionales de proveedores.

(Özkan, 2019a) clasifica las vulnerabilidades utilizando principalmente la coincidencia de palabras clave y secundariamente con los números de identificador de vulnerabilidad (CVE).

A menos que se indique lo contrario, los puntajes **CVSS** que publica Ozkan se incluyen en las fuentes de la base de datos NVD, la cual es actualizada diariamente, para más detalles consultar *www.nvd.nist.gov*.

Por otro lado, (Özkan, 2019b) proporciona una interfaz web única para las definiciones de lenguaje de evaluación y vulnerabilidad abierta (OVAL). La Tabla 1 muestra vulnerabilidades representativas del año 2009 a la fecha:

I	II	III	IV	V	VI	VII	VIII	IX	X	XI	XII	XIII
CVE-2009-2348	Bypass	2009-07-17	2018-10-10	6.9	Ninguno	Local	Medio	No requerido	Completar	Completar	Completar	2009
CVE-2010-1807	DoS ejecución de código	2010-09-10	2017-09-18	9.3	Ninguno	Remoto	Medio	No requerido	Completar	Completar	Completar	2010
CVE-2011-2344	+Priv	2011-07-08	2011-07-08	10.0	Ninguno	Remoto	Bajo	No requerido	Completar	Completar	Completar	2011
CVE-2011-3874	Desbordamiento de código	2012-01-27	2012-02-06	9.3	Ninguno	Remoto	Medio	No requerido	Completar	Completar	Completar	2012
CVE-2011-1352	Desbordamiento +Priv corrupción de memoria	2013-02-05	2013-02-08	6.9	Administ rador	Local	Medio	No requerido	Completar	Completar	Completar	2013
CVE-2013-4710	DoS	2014-03-02	2014-03-10	9.3	Ninguna	Remoto	Medio	No requerido	Completar	Completar	Completar	2014
CVE-2015-8507	Desbordamiento de ejecución de código DoS	2015-12-08	2015-12-09	9.3	Ninguno	Remoto	Medio	No requerido	Completar	Completar	Completar	2015
CVE-2016-6707	Ejecución de código +priv	2016-11-25	2017-02-06	9.3	Ninguno	Remoto	Medio	No requerido	Completar	Completar	Completar	2016
CVE-2017-13160	Ejecución de código	2017-12-06	2017-12-18	9.3	Ninguno	Remoto	Bajo	No requerido	Completar	Completar	Completar	2017
CVE-2017-13285	Ejecución de código XSS	2018-04-04	2018-05-09	10.0	Ninguno	Remoto	Bajo	No requerido	Completar	Completar	Completar	2018

Tabla 1. Vulnerabilidades más representativas del año 2009 a la fecha

I. CVE ID, II. Tipos de vulnerabilidad, III. Fecha de publicación, IV. Fecha de autorización, V. Puntuación, VI. Nivel de acceso ganado, VII. Acceso, VIII. Complejidad, IX. Autenticación, X. Confidencialidad, XI. Integridad, XII. Disponibilidad y XIII. Año.

2.3 REPOSITORIOS DE INFORMACIÓN SOBRE VULNERABILIDADES

Umansankar, (2017) afirma que la popularidad de la plataforma Android está aumentando día con día, en los últimos meses, Android ha batido al sistema operativo Windows en popularidad en términos de uso de Internet. También Android proporciona soporte para diversos dispositivos ubicuos. Así que un gran número de dispositivos inteligentes conectados están en la plataforma Android. Conduce a un gran aumento en la cantidad de manejo de datos por dispositivos Android. Los datos que maneja Android pueden ser datos sensibles, críticos o datos personales, por lo que los actos maliciosos de aplicaciones no deseadas se convierten en una gran amenaza para la seguridad, la privacidad de los usuarios y el sistema.

Para ello, examinamos y analizamos diversas listas de vulnerabilidad y de base de datos.

Todas las bases de datos de vulnerabilidades utilizan un identificador común para representar la vulnerabilidad que se llama CVE ID y tienen una puntuación del Sistema Común de Puntuación de Vulnerabilidad (CVSS), basada en el valor de CVSS, se identifica el nivel de impacto de la vulnerabilidad, puede ser clasificados en Crítico, Alto, Moderado, Bajo, Sin Impacto de Seguridad (NSI) basado en el nivel de gravedad. A continuación, el tipo de vulnerabilidad, el tipo de solución, el vector de acceso, la complejidad de acceso, el tipo de impacto, la autenticación, la integridad, la confidencialidad, las fechas de la solución de publicación y la actualización, etc., también se incluyen con la lista de vulnerabilidades. Dará un conocimiento claro sobre la vulnerabilidad.

En el análisis de vulnerabilidades en la plataforma Android se ha encontrado que en los últimos años las vulnerabilidades en la plataforma Android esta drásticamente en aumento. La Tabla 2 muestra el número de vulnerabilidades en los últimos 6 años.

Año	**# de vulnerabilidades**
2013	7
2014	13
2015	125
2016	523
2017	200
2018	43

Tabla 2. Número de vulnerabilidades reportada en años recientes.

Analizamos la lista de vulnerabilidades y bases de datos como OWASP móvil top 10, Base de datos nacional de vulnerabilidades, boletines de seguridad de Android, CVE Detalles del sitio web etc. Después de analizar las bases de datos y listas de vulnerabilidades identificamos que el número de vulnerabilidades identificadas, así como el número de ataques cibernéticos en la plataforma Android está aumentando en gran medida, especialmente en los últimos años. Después de un análisis detallado llegamos a saber que, el tipo de vulnerabilidad de ganar privilegios tiene la mayor incidencia con 25%, a continuación, la ejecución remota de código es 21%, denegación de servicio 17%, desbordamiento de buffer del 15% y así sucesivamente

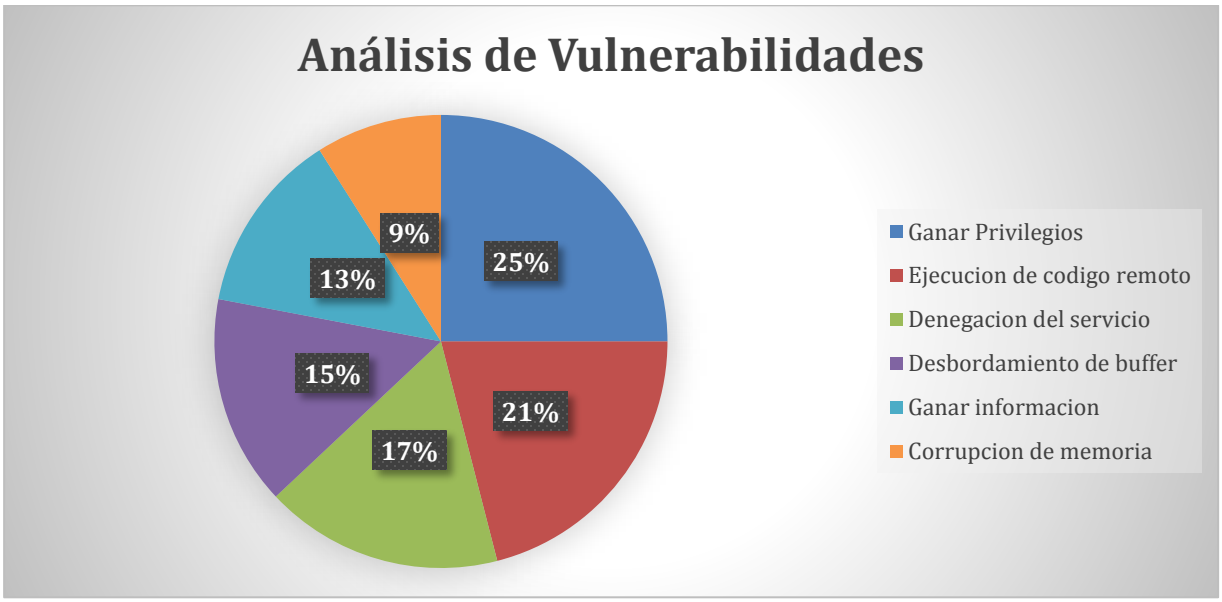


Figura 7. Análisis grafico de las vulnerabilidades

a) Ganar privilegios

En este ataque, el atacante obtiene privilegios al explotar la vulnerabilidad en el sistema operativo o la aplicación. Obtiene privilegios de recursos o servicios que normalmente no están disponibles o protegidos de aplicaciones normales. De esta manera las aplicaciones maliciosas pueden eludir los permisos y obtener acceso a los datos vitales del usuario y del sistema y hacer acciones no autorizadas.

b) Ejecución remota de código

En la ejecución remota de código muestra que el atacante puede ejecutar cualquier comando o código en un dispositivo de destino con su propia elección. Las vulnerabilidades presentes en plataformas como CVE-2016-3820 hacen posible esta explotación. CVE-2016-3820 es una vulnerabilidad de ejecución de código remoto en mediaserver de Android.

c) Denegación de Servicio

En este tipo de ataque, el atacante/aplicación maliciosa hace que el servicio no esté disponible para el usuario legítimo.

d) Desbordamiento de búfer

El ataque de desbordamiento de búfer se utiliza principalmente en relación con la ganancia de privilegios del atacante, en caso de ataque de desbordamiento de búfer el atacante escribe una gran cantidad de datos en el búfer y hacer que fallen de esta manera / aplicación maliciosa puede eludir privilegios / permisos.

CVE o identificador	Referencias	Severidad	Actualización de Google en dispositivos	Actualización AOSP versiones	Fecha reportada
CVE20170587	A-35219737	Critico	Todo	6.0, 6.0.1, 7.0, 7.1.1, 7.1.2	4 de Enero de 2017
CVE20170588	A-34618607	Critico	Todo	4.4.4, 5.0.2, 5.1.1, 6.0, 6.0.1, 7.0, 7.1.1, 7.1.2	21 de Enero de 2017
CVE20170589	A-34897036	Critico	Todo	5.0.2, 5.1.1, 6.0, 6.0.1, 7.0, 7.1.1, 7.1.2	1ero de Febrero de 2017
CVE20170590	A-35039946	Critico	Todo	5.0.2, 5.1.1, 6.0, 6.0.1, 7.0, 7.1.1, 7.1.2	6 de Febrero de 2017

Tabla 3. Ultimas vulnerabilidades reportadas

OWASP móvil superior 10 se crea como parte del proyecto de seguridad móvil de OWASP, es la lista de las 10 principales vulnerabilidades en la plataforma móvil. Se enumeran en la Tabla 4.

Ranking	Vulnerabilidad
1	M1 – Plataforma incorrecta Sintaxis
2	M2 – Almacenamiento inseguro de datos
3	M3 – Comunicación insegura
4	M4 – Autenticación insegura
5	M5 – Criptografía insuficiente
6	M6 – Autorización insegura
7	M7 – Pobre calidad de código
8	M8 – Manipulación de código
9	M9 – Ingeniería inversa
10	M10 – Funcionalidad inversa

Tabla 4. Las 10 vulnerabilidades más representativas en el año 2016 de acuerdo a la plataforma OWASP Mobile.

Con el fin de evaluar y analizar las vulnerabilidades de Android que necesitamos para crear un marco eficaz para analizar de forma dinámica la plataforma. Aquí se utiliza la fusión de Android OWASP para la creación de un marco. Es una plataforma Debian que es dedicada para Android en pruebas de penetración, y análisis forense, etc. Las herramientas de Fussion Droid detectan software malicioso que se utilizan para el análisis dinámico de la aplicación.

2.3.1 Plataforma CVE Details

La plataforma CVE Details proporciona una interfaz web fácil de usar para los datos de vulnerabilidad CVE. Puede buscar proveedores, productos y versiones y ver entradas de CVE, vulnerabilidades relacionadas con ellos. Puede ver estadísticas sobre proveedores, productos y versiones de productos. Los detalles de CVE se muestran en una sola página fácil de usar.

Los datos de vulnerabilidad de CVE se obtienen de las fuentes XML de la Base de datos nacional de vulnerabilidades (NVD) proporcionada por el Instituto Nacional de Estándares y Tecnología. Los datos adicionales de varias fuentes como exploits de www.exploit-db.com, declaraciones de proveedores y datos adicionales proporcionados por proveedores, los módulos Metasploit también se publican además de los datos NVD CVE.

Cvedetails.com clasifica las vulnerabilidades mediante la concordancia de palabras clave y números CWE si es posible, pero se basan principalmente en palabras clave.

A menos que se indique lo contrario, las puntuaciones CVSS que figuran en este sitio son "puntuaciones base CVSS" proporcionadas en las fuentes de NVD. Los datos de vulnerabilidad se actualizan a diario mediante las fuentes de NVD. Visite nvd.nist.gov para obtener más detalles.

De la revisión literaria realizada en esta plataforma, se observa que del año **2009 al 2019** se han documentado **2563** vulnerabilidades (ver Figura 8), lo cual es preocupante para Android que es el sistema operativo más utilizado en todo el mundo, los usuarios se ven expuestos a dichas vulnerabilidades de acuerdo a las estadísticas de vulnerabilidades documentadas en la plataforma CVE Details.

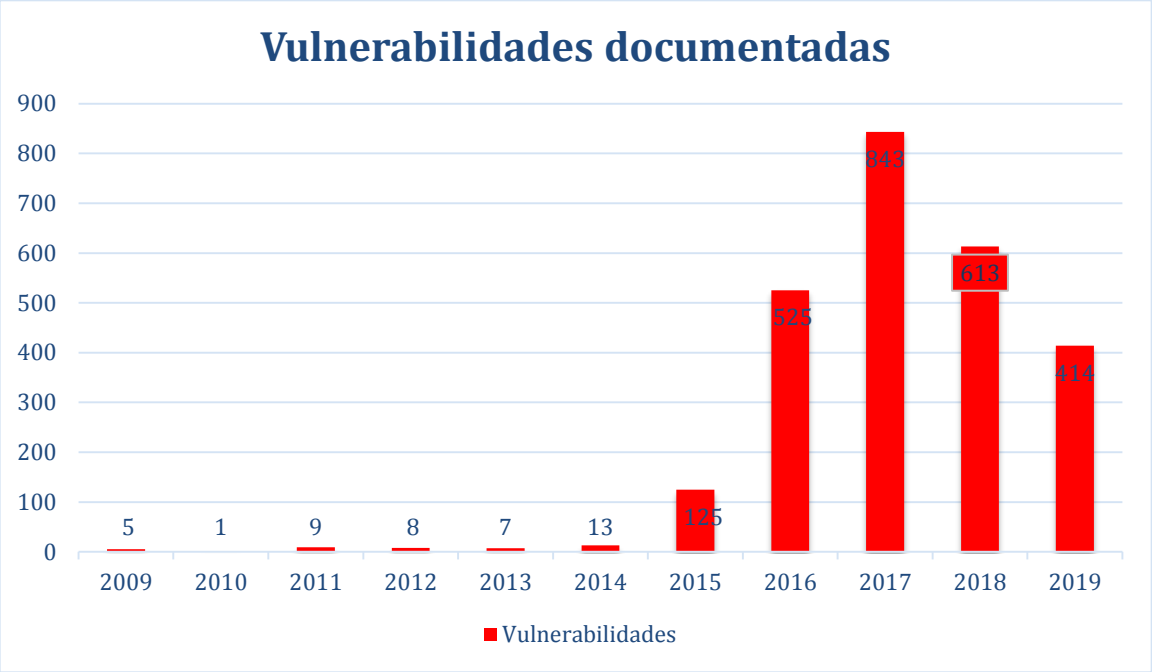


Figura 8. Vulnerabilidades documentadas del año 2009-2019 (CVE Details, 2020)

Más aún, de acuerdo a la plataforma CVE Details, las vulnerabilidades con más impacto en Android al 2019 son: Ejecución de código **534** (20.8%), Desbordamiento de búfer **503** (19.6%), DoS **329** (12.8%), Obtener privilegios **315** (12.3%), Obtener información **313** (12.2%), Bypass **152** (5,9%), Corrupción de la memoria **150** (5.9%).

2.4 VULNERABILIDADES EN EL KERNEL

Una vulnerabilidad en el Kernel de Android permite a una aplicación maliciosa **ganar privilegios de root** en el sistema. Por tanto, una vulnerabilidad de elevación de privilegios en el Kernel permite a una aplicación maliciosa local ejecutar código arbitrario en el núcleo.

La mayoría de los dispositivos basados en Android 6,0 ejecutan el Kernel de linux v3.18. Debido a que las versiones del Kernel varían dependiendo de la versión de Android que tenga instalado un dispositivo son diferentes las vulnerabilidades en el Kernel a las que pueda estar expuesto (Xataka Android, 2018).

Los expertos de Lookout indican que todas las versiones de Android con el Kernel de Linux 3.6 a la última se ven afectados por el defecto de Linux CVE-2016-569

La falla de red TCP/IP permite a los atacantes acceder al punto de comunicación entre dos entidades y se pueden explotar para secuestrar el tráfico y manipularlo si los datos no están encriptados. Los atacantes sólo necesitan enviar paquetes simulados a ambos lados de la conexión por el simple hecho de saber sus direcciones IP y puertos de destino. La vulnerabilidad permite a los atacantes obtener tráfico sin cifrar y manipular el tráfico cifrado para espiar a las víctimas.

De acuerdo con los expertos de seguridad del puesto de observación, la vulnerabilidad de Linux afecta al 80% de los dispositivos Android, que parece haber sido introducido en la versión Android 4.4 (también llamado KitKat) y que todavía está presente en las versiones actuales.

Finalmente, debe hacerse notar que la vulnerabilidad en el Kernel de Android podría ser explotada por atacantes para copiar ilegalmente los datos, insertando el malware en las descargas y páginas web, ejecutando así una amplia gama de ataques (Clases ordenador, 2018).

Las vulnerabilidades en el Kernel según Hei et al., (2013), se examinaron los códigos fuente de dos paquetes: GTP7500 OpenSource. zip y GT-P7510 OpenSource. Zip, y encontramos dos vulnerabilidades en la función de nvhost_ioctl_ctrl_module_regrdwr en el archivo dev.c. La función nvhost_ioctl_ctrl_module_regrdwr tiene 2 sub-funciones: nvhost_write_module_regs y nvhost_read_module_regs. La primera vulnerabilidad está en la subfunción nvhost_write_module_regs. El usuario Get (offs, desplazamientos) en la línea 561 se utiliza para obtener el desplazamiento de los usuarios. El nvhost_write_module_regs (& CTX-> dev-> cpuaccess, args-> ID, offs, Batch, vals) en la línea 569 determinan la ubicación basada en OFFS de los usuarios (los offs se utilizan como desplazamiento para escribir en los registros). Dado que no hay ninguna comprobación de límites en "offs", crea una vulnerabilidad de desbordamiento de búfer del Kernel que permite el acceso a memoria arbitraria. Los hackers pueden explotar la vulnerabilidad para escalar los privilegios del Kernel. La subfunción del módulo de lectura nvhost tiene una vulnerabilidad similar.

Después de explotar la vulnerabilidad, tenemos el control total del dispositivo Android, es decir, como un usuario **root** en el shell. Podemos acceder a los registros del Kernel durante la ejecución de una prueba difusa. Escribimos el código fuente de la prueba difusa por nosotros mismos. Analizando registros del Kernel durante ese período, encontramos la segunda vulnerabilidad, que está en la línea 598: BUG ON (_IOC_SIZE (cmd)-> NVHOST_IOCTL_ CTRL_MAX_ARG_SIZE). El programa no puede marcar el tamaño de IOC_SIZE (cmd), y esto puede causar un ataque DoS para bloquear el sistema operativo. A continuación se muestra el código de explotación de regrdwr.c:

```
[<c0008b2c>] (start_kernel+0x294/0x2ec) from [<00008080>] (0x8080)
[DBG] upload cause : UPLOAD_CAUSE_KERNEL_PANIC
(kernel_sec_set_upload_cause) : upload_cause set c8
Rebooting in 10 seconds..misc_sec_operation d36124e0 1ff000 52 0
misc_sec_operation: filp_open failed. (-13)
sec_open_param PARAM OPEN FAIL
(kernel_sec_get_debug_level) The debug value is invalid(0x0)!! Set default level(LOW)
(kernel_sec_hw_reset) Upload Magic Code is cleared for silet reset.
(kernel_sec_hw_reset)
(kernel sec hw reset) The forced reset was called. The system will be reset !!
```

Figura 9. El principal código de explotación de regrdwr.c

2.4 .1 EXPLOTANDO LAS VULNERABILIDADES EN EL KERNEL

De acuerdo al autor Hei et al., (2013), cita que se distinguen 2 vulnerabilidades principales del Kernel del sistema operativo Android:

a) Vulnerabilidad 1

Puesto que Android se basa en un Kernel de Linux modificado y por lo tanto, aplica el control de acceso discrecional (DAC) en el nivel de sistema de archivos, que se basa en los identificadores de usuario (UID) y los identificadores de grupo (GID). Si el UID = 0, esto significa que el usuario obtiene el privilegio de nivel de raíz, que es el objetivo de explotar la vulnerabilidad de escalar privilegios. La primera vulnerabilidad se conoce como escalar los privilegios.

La primera vulnerabilidad se conoce como escalar los privilegios. Dado que conocemos esta vulnerabilidad relacionada con la dirección, podemos escanear el registro de los kallsyms y encontrar el desplazamiento de la función sys-setuid. Esto significa que podemos encontrar la dirección de la función sys-setuid. Si insertamos código malicioso aquí, entonces podemos ejecutar el código malicioso para cambiar el UID. Sobrescriben el código de setuid usando NewValues [0] = 0 para obtener el privilegio **root**, luego setuid = 0. Después de eso, creamos un shell. Ahora, tenemos el control total del sistema operativo Android. Probamos la primera vulnerabilidad en regrdwr. c. La figura 3 muestra el código principal de regrdwr. c.

Al explotar el código como se describió anteriormente, confirmamos que la vulnerabilidad de escalar los privilegios se encuentra disponible actualmente en varias versiones del sistema operativo Android.

Llevamos a cabo las pruebas mediante el uso de un dispositivo Android real, una tableta Samsung Galaxy 10.1. La figura 3 es una copia de pantalla del resultado de explotar la vulnerabilidad. La Figura 10 muestra que después de ejecutar el código de explotación, el UID fue cambiado de 7d0 a 0. Esto validó que escalamos correctamente a privilegios de **root**.

```
#  ./regrdwr
 dumpbin->
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
---------current  uid = 7d0
---------return = 0
dumpbin->
0 0 0 a 4 50 84 e5 c 50 84 e5 4 30 97 e5 ------------ return = 0
dumpbin->
0 0 0 a 4 50 84 e5 c 50 84 e5 4 30 97 e5
---------current  uid = 0
shell@android;/data/local/tmp #
130[ shell@android:/data/local/tmp #
130[ shell@android:/data/local/tmp #
```

Figura 10. El ataque UID a cambiar

b) Vulnerabilidad 2

La segunda vulnerabilidad se conoce como la vulnerabilidad de DoS. Podemos explotar fácilmente esta vulnerabilidad con una simple prueba difusa. Probamos la segunda vulnerabilidad con nvfuzz. c. La Figura 5 muestra el código principal de nvfuzz. c. El sistema operativo Android se estrelló varias veces. La Figura 6 muestra los registros del Kernel. Si un hacker inserta códigos maliciosos en aplicaciones Android que están disponibles a todos los usuarios, entonces miles de dispositivos Android con chips NVIDIA Tegra se bloqueará.

Si un hacker combina estas dos vulnerabilidades, entonces él puede bloquear una gran cantidad de dispositivos, desactivar el software antivirus, instalar cualquier malware, crear malware, e incluso publicar aplicaciones maliciosas en el mercado de aplicaciones Android con el pago de $25 como tarifa de registro.

Confirmamos la vulnerabilidad de DoS en varias versiones disponibles actualmente del Sistema Operativo Android Honeycomb por pruebas difusas. Ejecutamos las pruebas usando un dispositivo Android real: una tableta SamsungGalaxy 10.1.

```
for(i1=0;i1<4;i++)
{       for(i2=0;i2<2;i2++)
        {  for(i3=0x30;i3<0x7a;i3++)
                { l4stop= 0x100;
                  for(i4=0;i4zl4stop;i4++)
                  {icode=i1s[i1]*0x1000000+ i2s[i2]* 0x1000000+i3* 0x100+i4;
                   printf("%x,%x)\n", files [fileindex], fd, icode);
                   ioctl(%x,%x)\n", files [fileindex], fd, icode);
                   for(i=0;i<8;i++)
                         { ret=ioctl(fd,icode,data[i]);
                               if(ret !=0 && ret!= -1)
                               {        l4stop= 0x100;
                                        if(debug)
                                         perror("error ioctl\n"); continue; }
                                  else if(ret==-1)
                                 { continue; }
                                  else { l4stop= 0x100;| }}}}}}
```

Figura 11. El código principal de nvfuzz.c

La Figura 11 muestra que después corremos el código de explotación, el núcleo entra en pánico y el sistema se reinicia. Si continuamente corremos el código de explotación, el sistema no puede funcionar más. Por lo tanto, se trata de un exploit que conduce al ataque DoS.

Hemos comprobado en diferentes versiones del Sistema Operativo Android Honeycomb. Para cada versión, se corre la prueba muchas veces. Todas nuestras pruebas han causado que Android Honeycomb se bloquee. Después se analizó el código fuente de los drivers para Tegra, creemos que las dos vulnerabilidades son universales en el dispositivo Android con Tegra.

```
[<c0008b2c>] (start_kernel+0x294/0x2ec) from [<00008080>] (0x8080)
[DBG] upload cause : UPLOAD_CAUSE_KERNEL_PANIC
(kernel_sec_set_upload_cause) : upload_cause set c8
Rebooting in 10 seconds..misc_sec_operation d36124e0 1ff000 52 0
misc_sec_operation: filp_open failed. (-13)
sec_open_param PARAM_OPEN_FAIL
(kernel_sec_get_debug_level) The debug value is invalid(0x0)!! Set default level(LOW)
(kernel_sec_hw_reset) Upload Magic Code is cleared for silet reset.
(kernel_sec_hw_reset)
(kernel_sec_hw_reset) The forced reset was called. The system will be reset !!
```

Figura 12. Logs para el Experimento 2 -. Dos vulnerabilidades

2.4.2 CORRECCIÓN DE LAS VULNERABILIDADES EN EL KERNEL

La corrección para la vulnerabilidad de escalar los privilegios consiste en comprobar si los "offs" en la función "nvhost" leen el módulo de lectura y la función nvhost del módulo de escritura está fuera del límite. La solución para la vulnerabilidad de DoS es restringir el tamaño de IOC (CMD).

1. **Solución para la vulnerabilidad de elevación de privilegios**

Las funciones nvhost_read_module_regs y nvhost_write_module_regs no realizan ninguna verificación específica en las variables offs. Por lo tanto, la corrección es agregar una comprobación de la variable offs y ver si está fuera del límite.

2. **Solución de la vulnerabilidad DoS**

Debido a que el programa no puede comprobar el tamaño de IOC SIZE (CMD), un usuario malintencionado podría enviar un cmd muy largo para desbordar el Kernel. Esto provocará que el Kernel entre en pánico. Si añadimos la verificación de la longitud para marcar el tamaño del tamaño IOC (CMD), este problema se resuelve.

3. **Pruebas de las contramedidas**

Implementamos las dos contramedidas en nuestra Tablet Samsung Galaxy 10,1. En particular, para la versión Android Honeycomb 3.0.1, construimos una versión parchada que incluye dos parches, recompilando Android desde cero. Nuestras pruebas muestran que ambos parches son efectivos y evitan los códigos de explotación, por lo tanto, la fijación de las dos vulnerabilidades.

2.4.3 RESULTADOS DE LOS TRABAJOS REALIZADOS

Hay tres tendencias principales:

- Análisis estático
- Evaluación esquema de seguridad
- Detección de malware y virus

El análisis estático incluye el uso de las metodologías de caja blanca o caja negra para detectar comportamientos maliciosos en aplicaciones Android antes de instalarlos en los dispositivos. Un artículo de Enck et al. Tiene un estudio horizontal de las aplicaciones de Android para descubrir el robo de datos personales. Proponen Scandroid, que es una herramienta de razonamiento automático para encontrar violaciones de seguridad de las aplicaciones de Android. El análisis estático podría ayudar a identificar vulnerabilidades en el kernel.

La segunda tendencia de la investigación actual es estudiar el acceso los esquemas de control de accesos y permisos de Android. Por ejemplo, propone un esquema para evaluar los privilegios reales de aplicaciones de Android y desarrolla una herramienta llamada Polizón, para detectar el exceso de privilegios en aplicaciones Android compiladas. Nauman et al. Proponen Apex, que es un marco de aplicación de políticas para Android que permite a un usuario conceder permisos de las aplicaciones e imponer restricciones sobre el uso de los recursos.

Ongtang et al. Presentan una infraestructura denominada interacción de aplicaciones seguras (SAINT) para gobernar asignación de permisos durante la instalación. Muchas obras se centran en los problemas de escalabilidad de privilegios. Proponen un marco de seguridad llamado monitoreo extendido en Android (XManDroid) para extender el mecanismo de monitoreo nativo de Android para detectar el ataque de escalar los privilegios.

El ataque de escalar privilegios en Android fue propuesto por primera vez por Davi et al. En el que demostraron un ejemplo del ataque. Mostraron que una aplicación genuina explotada en tiempo de ejecución o una aplicación maliciosa puede escalar permisos concedidos. Sin embargo, no sugieren ninguna defensa para el ataque. Las aplicaciones desfavorecidas bajo el control del usuario malintencionado pueden realizar operaciones indirectamente invocando otras aplicaciones que poseen los privilegios deseados.

Los ataques publicados son llamadas telefónicas no autorizadas, envió de mensajes de texto a descargas ilegales de archivos maliciosos, y grabación de voz consciente del contexto. La mayoría de los ataques de escalar privilegios aprovechan las interfaces vulnerables de aplicaciones privilegiadas. Este ataque se conoce a menudo como ataque adjunto confundido.

Sin embargo, en general, el adversario puede diseñar sus propias aplicaciones maliciosas que colaborar para montar un ataque de colusión: las aplicaciones con permisos no críticos pueden coludir para generar un conjunto de permisos de que les permita realizar acciones no autorizadas. Algunas aplicaciones de colusión pueden explotar canales cubiertos o abiertos del sistema básico de Android para evitar la detección.

Tenga en cuenta que el modelo de distribución de aplicaciones Android permite cualquier persona que se ha registrado como un desarrollador de Android (y pago $25 de cuota) para publicar aplicaciones en el mercado Android. Este esquema permite a los adversarios cargar fácilmente aplicaciones de maliciosos en la tienda del mercado: por ejemplo, Android DroidDream troyano del año 2011 (que contiene una raíz exploit) ha sido identificado en más de 50 aplicaciones oficiales del mercado Android y se ha descargado más de 10.000 veces antes de que se haya detectado. La literatura se centra en posibles amenazas y soluciones para mitigar el problema de la escalada de privilegios.

En la tercera tendencia de detección de virus y malware, Dagon et al. Evaluó muchos virus móviles y malware que potencialmente podría afectar a los dispositivos Android. Crowdroid es propuesto como un

detector de malware que ejecuta un análisis dinámico en los comportamientos de la aplicación. Inspecciona archivos ejecutables de Android para extraer sus llamadas de función y compararlos con ejecutables de malware para fines de clasificación.

También se podrían generar firmas específicas de malware para explotar las vulnerabilidades descritas en este documento. Todos estos trabajos son para proteger la privacidad y seguridad del usuario. Los autores piensan que es posible que necesitemos un modo de privacidad original en los dispositivos móviles Android. La literatura presenta un ataque DoS que hace dispositivos totalmente insensibles bifurcando repetidamente el proceso de cigoto. Las vulnerabilidades que se revelan en este documento requieren que la función de depuración USB está habilitada.

2.5 VULNERABILIDADES EN AUTENTICACIÓN Y SEGURIDAD

Android tiene funciones de seguridad integradas en el sistema operativo que reducen notablemente la frecuencia y el impacto de los problemas de seguridad de la aplicación. El sistema está diseñado de modo que puedas compilar tus apps con permisos predeterminados del sistema y de archivos, y evitar tener que tomar decisiones difíciles sobre seguridad.

2.5.1 SOFTWARE ANALIZADOR DE VULNERABILIDADES

Tang et al.,(2017) El trabajo se centra en el estudio de los problemas de seguridad traído por los componentes que se supone que son privados, pero se han establecido erróneamente como públicos por los desarrolladores de aplicaciones, suponiendo que el marco de Android puede proteger los componentes de privada de ser invocado por otras aplicaciones.

Sin embargo, recientemente se descubrió una nueva clase de vulnerabilidad llamada vulnerabilidad de próxima intención (NIV), que muestra que componentes privados podrían ser invocados por otras aplicaciones. Más específicamente, una aplicación maliciosa puede ofrecer una intención diseñada para iniciar un componente público de la aplicación vulnerable, en el que se guarda una intención de destino bajo una clave especial. A continuación, la intención de destino se recupera y se pasa a los métodos de comunicación entre componentes (ICC), como startActivity. El resultado es que el atacante puede invocar el componente privado designado de la aplicación vulnerable.

La causa fundamental de NIV es que los desarrolladores diseñan una forma insegura de redirigir los componentes durante la interacción del usuario con la aplicación. Por ejemplo, redirigir a un componente especificado después del usuario ha iniciado sesión correctamente. Como resultado, muchas aplicaciones, incluyendo proyectos de código abierto y SDK de terceros, todavía tienen NIVs explotables. Sin embargo, no existe un enfoque automatizado para detectar y verificar estas vulnerabilidades a gran escala.

Por lo tanto, no se sabe cuán severa y prevaleciente esta clase de vulnerabilidades en aplicaciones Android del mundo real. Para responder estas preguntas, en este artículo presentamos NIVAnalyzer, que detecta y verifica automáticamente las vulnerabilidades de la siguiente intención.

NIVAnalyzer se compone de dos módulos: módulo descubrimiento y módulo de explotación NIV. Con el fin de encontrar NIV eficientemente y eficazmente a gran escala, diseñamos una estrategia de análisis de flujo intención que rastrea con precisión la intención en código smali.

El módulo de descubrimiento de NIV realiza principalmente análisis de flujo de intención estática, que está diseñado para rastrear la instancia de destino y comprobar si cumple con todas las características de NIV. Mientras tanto, genera información relevante para guiar la explotación de la vulnerabilidad. El módulo de explotación NIV instala la aplicación vulnerable en el emulador de Android y luego construye

automáticamente un proyecto de prueba de Robotium que incluye casos de prueba para explotar el NIV y simular el proceso de inicio de sesión posible.

Con la información de registro recopilada desde el emulador de Android durante la explotación, somos capaces de ver si la vulnerabilidad se explota con éxito. Utilizamos NIVAnalyzer para analizar 20,000 aplicaciones y finalmente encontrado que 190 de ellas tienen NIV. Algunas de las aplicaciones vulnerables descubiertas son populares con más de 500 millones de descargas.

Una aplicación para Android se compone de cuatro tipos de componentes: Actividad, Servicio, receptor de radiodifusión (Broadcast) y el proveedor de contenido. Un intento es un objeto de mensajería que puede ser utilizada para solicitar una acción de otro componente. Es un medio de comunicación entre componentes (CPI) en aplicaciones de Android y tiene muchos casos de uso, como el inicio de una actividad o servicio y la entrega de una emisión.

Un intento es una descripción abstracta de la operación a realizar. Un objeto intención lleva la información utilizada por el sistema Android para determinar qué componente para enviar especificando explícitamente el nombre del componente (es decir, intención explícita) o declarando los atributos de la intención general (es decir, intención implícita). También se puede llevar información adicional en sus extras campo para el componente receptor para llevar a cabo correctamente una operación. API's de Android, tales como startActivity, Comienza el servicio y sendBroadcast, utilizar las intenciones como parámetros para llevar a cabo la comunicación entre componentes.

Un componente Android se puede establecer como privado o público. Un componente es público si su propiedad "exportado" se establece como verdadero o se declara al menos un filtro intención en el manifiesto del archivo. De lo contrario, el componente es privado. Componentes públicos pueden ser provocados por otras aplicaciones mientras que los componentes privados sólo pueden iniciarse por los componentes de la misma aplicación o aplicaciones que comparten el mismo ID de usuario.

Wang et al. Descubrieron la vulnerabilidad de la siguiente intención (NIV) en las aplicaciones de Android y demostraron la vulnerabilidad usando las aplicaciones de Dropbox y Facebook. El atacante puede invocar el componente privado de la aplicación vulnerable, que originalmente se pretendía invocar por componentes dentro de la aplicación vulnerable. La causa fundamental de NIV es que los desarrolladores diseñan una manera insegura de redirigir los componentes durante la interacción del usuario con la aplicación.

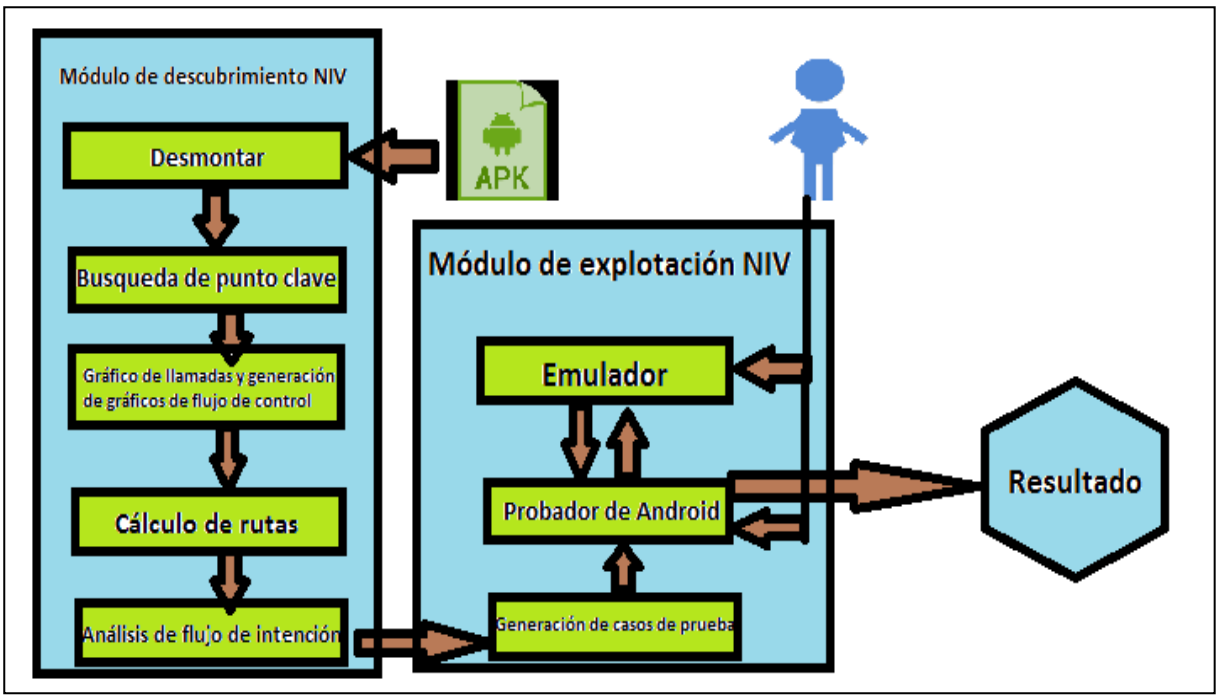


Figura 13. Arquitectura de NIVAnalyzer

Para ilustrar mejor el problema y simplificar aún más la discusión, aquí definimos algunas terminologías. Denotamos la intención extraída de otro intento bajo una tecla especial como la intención de destino. La intención de proxy es la que guarda la intención de destino. El registro que almacena la intención de destino se llama registro de destino y el registro que almacena la intención proxy se llama registro proxy.

La Figura 13 da una visión general de NIVAnalyzer. Se compone de dos módulos. El módulo de descubrimiento NIV que toma como entrada la aplicación y analiza si es potencialmente vulnerable. Si es así, se activa el módulo de explotación NVI para generar casos de prueba y activar la NIV de la aplicación de la víctima en el emulador.

1) Módulo Descubrimiento NVI

Este módulo utiliza el análisis estático para comprobar si la aplicación dada contiene vulnerabilidad de la siguiente intención. Primero se escanea el código smali desmontado para encontrar los puntos clave, que se utilizan para filtrar los métodos con código potencialmente vulnerable. Para cada método con puntos clave, calcula las posibles rutas de ejecución del método. Realizamos análisis de flujo estático en cada ruta de ejecución posible NVI.

Puntos clave de búsqueda. Definimos puntos clave como instrucciones que recuperan la intención de destino de otro intento. Para dar un ejemplo, un intento invoca el método getParcelableExtra (llave) para recuperar un objeto con llave como el parámetro. A continuación, la instrucción con el código de operación check-cast se invoca para convertir el objeto parcelable a una instancia de intención. Al menos un punto clave debe existir en el código con NVI. Por lo tanto podemos apuntar rápidamente el código potencialmente vulnerable escaneando el código smali para encontrar todos los puntos clave. El análisis continúa si existe punto clave en el código smali.

Cálculo de rutas de acceso. Diferentes ramas pueden controlar la intención destino de diferentes maneras. Puesto que queremos rastrear el flujo de la intención objetivo, nuestro análisis es sensible a la ruta y tiene en cuenta el efecto de cada rama. Para un punto clave, NIVAnalyzer construye primero el gráfico de flujo de

control (CFG) del método que lo incluye. Cada nodo de la CFG representa un bloque básico y aristas dirigidas que representan transiciones entre bloques. A continuación, establecemos el nodo con el punto clave como punto de inicio.

Análisis de flujo de Intención. El análisis de flujo de intención toma una posible ruta de ejecución como entrada. Se realiza primero análisis hacia adelante para comprobar si este camino pasaría la intención de destino a un método fregadero. Los métodos fregadero se definen como la comunicación entre componentes (ICC) métodos tales como startActivity, StartService, sendBroadcast, etc. Estos métodos utilizan las intenciones como el parámetro para especificar los componentes que se invocan posteriormente. Si la intención de destino puede llegar a un método fregadero, nuestro análisis sigue llevando a cabo el análisis hacia atrás para comprobar si la intención de proxy se devuelve desde el getIntent API invocado por un componente público.

Los nodos que no tienen aristas salientes se definen como puntos finales. A continuación, usamos la búsqueda de amplitud primero para calcular las rutas de ejecución posibles desde el punto inicial hasta todos los puntos finales. Cada arista solo se puede atravesar una vez en un trazado. La mayoría de las veces, el número de rutas calculadas son pocos. Pero se generaría una gran cantidad de rutas cuando hay muchas ramas en el método. Para manejar eficientemente este caso, establecemos el umbral máximo de las rutas de ejecución.

De nuestras estadísticas, encontramos que la mayoría de los métodos incluyen menos de 16 ramas entre el punto clave y el sumidero. Por lo tanto, establecemos el umbral como 216. Cuando el número de rutas es mayor que el umbral, detendremos el proceso de cálculo de rutas y realizaremos un análisis de flujo de intención en las rutas ya generadas.

Se realiza un análisis directo donde las instrucciones de Smali utilizan registros para almacenar argumentos. Denotamos el registro que almacena el destino rastreado intent como el registro de destino. Nuestro objetivo es comprobar si el registro destino se pasará a un método de sumidero.

El módulo estático de NIVAnalyzer tomará las acciones correspondientes cuando se encuentre con cada clase de instrucciones. Cuando se define que el registro en la lista de seguimiento se pasa a los métodos de receptor, se continúa con el análisis hacia atrás. De lo contrario, se llega a la conclusión de que la ruta actual no tiene NVI.

2) Módulo de Explotación NVI

El objetivo de este módulo es verificar la existencia de NIV en la aplicación potencialmente vulnerable y explotar la vulnerabilidad mediante el inicio de un componente privado designado. El módulo de explotación NIV instala la aplicación potencialmente vulnerable en el emulador de Android y, a continuación, crea una aplicación de ataque para explotar el NIV.

Con la información de registro recopilada del emulador de Android durante la explotación, podemos saber qué componente se invoca y si la vulnerabilidad se explota con éxito

Generación de casos de prueba. Para verificar las vulnerabilidades descubiertas en el módulo de detección NIV, necesitamos generar casos de prueba con el objetivo de explotar estas vulnerabilidades. Una prueba de caso libera una intención diseñada de proxy, que lleva una intención, con el objetivo de iniciar la actividad privada elegida de la aplicación vulnerable.

Se requiere información básica para crear la intención proxy. Además del componente público vulnerable, la actividad privada elegida y la clave especial para almacenar la intención de destino, también son necesarios algunos pares clave de valor bien diseñados en la intención proxy. Esto se debe a que el componente receptor a menudo comprueba ciertos atributos de entrada para comprobar su integridad o utilizarlo para decidir el flujo de control.

Algunas aplicaciones incluso emplean lógica empresarial compleja y estricta restricción en los valores de entrada. Por lo tanto, la explotación exitosa la vulnerabilidad es complicada. Nuestro enfoque para crear intenciones de proxy significativas se basa en la observación de que si existe un NIV en una aplicación, los desarrolladores de esta aplicación sin duda lo usarán para invocar componentes para cumplir la funcionalidad de la aplicación. Más específicamente, la propia aplicación crea una intención de proxy, en la que se almacena una intención de destino. Por lo tanto sólo tenemos que encontrarla y extraer la información que necesitamos para crear nuestra intención de proxy.

Explotación. En nuestro análisis manual, encontramos que la mayoría de las aplicaciones inician la actividad de inicio de sesión oficial antes de iniciar el componente vulnerable. Una vez que se realiza el inicio de sesión, se ejecuta el código vulnerable. Así que agregamos un módulo basado en Robotium para emulación de operaciones humanas para el inicio de sesión automático.

Robotium es un marco de automatización de pruebas Android que proporciona soporte completo para aplicaciones nativas e híbridas y pruebas de interfaz de usuario de caja negra fáciles de escribir. La clase de prueba del proyecto incluye todos los métodos test, un método setUp() que se ejecutará antes de cada método de prueba que comienza a ejecutarse y un método teardown() que automáticamente se ejecuta al final para liberar todos los recursos. Garantiza que la ejecución de cada caso de prueba es independiente.

Cada método de prueba desencadenará un nuevo caso de prueba y simulará el siguiente proceso de inicio de sesión. Con el fin de probar la aplicación potencialmente vulnerable con nuestro proyecto de prueba, Hemos instrumentado el sistema Android para pasar por alto la fase de verificación de firmas. Por lo tanto, no es necesario volver a firmar la aplicación potencialmente vulnerable.

NIVAnalyzer primero ejecuta la aplicación vulnerable potencial. A continuación, navegamos manualmente a la actividad de inicio de sesión y pasa la información de la cuenta a la consola de NIVAnalyzer. NIVAnalyzer inmediatamente usa UIAutomator para volcar la jerarquía de la interfaz de usuario actual. Los datos exportados se almacenan en un archivo XML que podemos utilizar para encontrar los widgets de la interfaz de usuario y simular el proceso de inicio de sesión. Por ejemplo, el tipo del campo de entrada para la contraseña es un Editar Texto y el atributo de contraseña se establece en verdadero. Otro Editar Texto adyacente a él es el campo de entrada para el nombre de usuario. Y el texto de la sugerencia suele ser "correo electrónico", "teléfono", "nombre de usuario", etc.
El botón de inicio de sesión tiene un atributo de enlaces que se establece en verdadero. Y el texto que aparece en el botón suele decir "iniciar sesión", "iniciar en", "iniciar sesión", etc. Hay algo especial que algunas aplicaciones tienen que hacer clic en un botón de inicio de sesión y luego se carga la página de inicio de sesión real. Nos ocupamos de esta situación encontrando en primer lugar y haciendo clic en el botón de inicio de sesión cuando no podemos encontrar el campo entrada para la contraseña. Usando la información anterior, podemos usar Robotium para introducir automáticamente el nombre de usuario y la contraseña en el lugar correcto en la actividad de inicio de sesión.

Con el fin de comprobar si la explotación es exitosa o no, NIVAnalyzer vuelca automáticamente los registros del emulador Android en tiempo real. A continuación, investiga el registro para ver si se invoca el componente privado elegido, coincidencia del nombre del componente, hora de inicio, etc. Cuando encuentra que el componente privado elegido se invoca correctamente, informa de que se encuentra un NIV y comienza a probar la siguiente aplicación potencialmente vulnerable.

2.5.2 EVALUACIÓN Y RESULTADOS DEL SOFTWARE NIVANALYZER

A continuación se dan a conocer los resultados de la evaluación de NIV Analyzer en aplicaciones del mundo real para una comprensión más clara de los efectos de las NIV's.

Con la ayuda del software NIVAnalyzer se realizó un análisis en un servidor con 2 CPU Intel Xeon E5-2650 y 128 GB de memoria física con Ubuntu 12.04. Elegimos al azar 20.000 aplicaciones de nuestra base de datos, que se rastrea de Google Play Store desde 2014. 503 de ellos no se pudieron desmontar, por lo que analizamos las aplicaciones restantes 19,497. Utilizamos 35 procesos para realizar el análisis en paralelo. Se tardó 10,8 horas en finalizar el análisis estático. Encontramos 203 aplicaciones para tener NIV en el de detección NIV, y en última instancia 138 aplicaciones fueron confirmadas vulnerables por el módulo de explotación NIV. 65 aplicaciones no pueden iniciar sesión automáticamente, para un CAPTCHA es necesario o se necesita un proceso de inicio de sesión complejo o el servidor de estas aplicaciones ha estado fuera de servicio. Por lo tanto probamos manualmente estas 65 aplicaciones y finalmente confirmamos 52 aplicaciones.

Analizamos manualmente 20 aplicaciones vulnerables confirmadas con el mayor número de descargas. La Figura 14 proporciona el desglose de los componentes públicos vulnerables por su tipo. El 85% de los componentes vulnerables son Actividades, el 15% son receptores de broadcast y el 0% son servicios. Podemos ver que NIV principalmente existe en Actividades. Entre los componentes privados que se pueden invocar explotando NIV, el 91% son Actividades y 9% son Servicios.

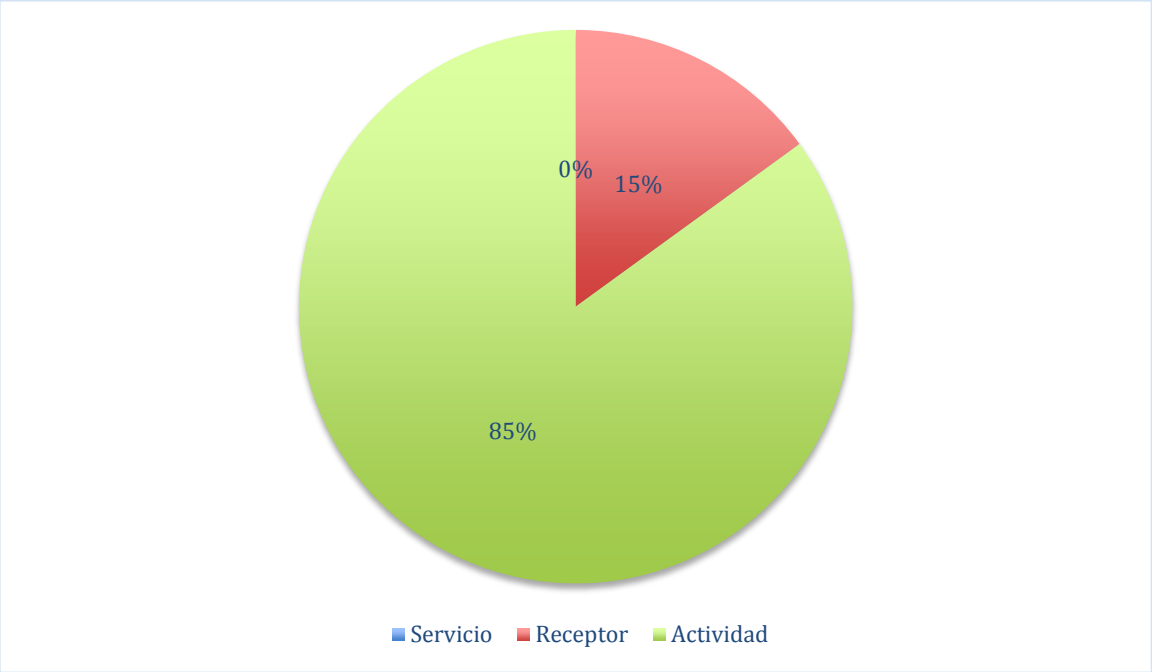


Figura 14. Distribución de los componentes vulnerables

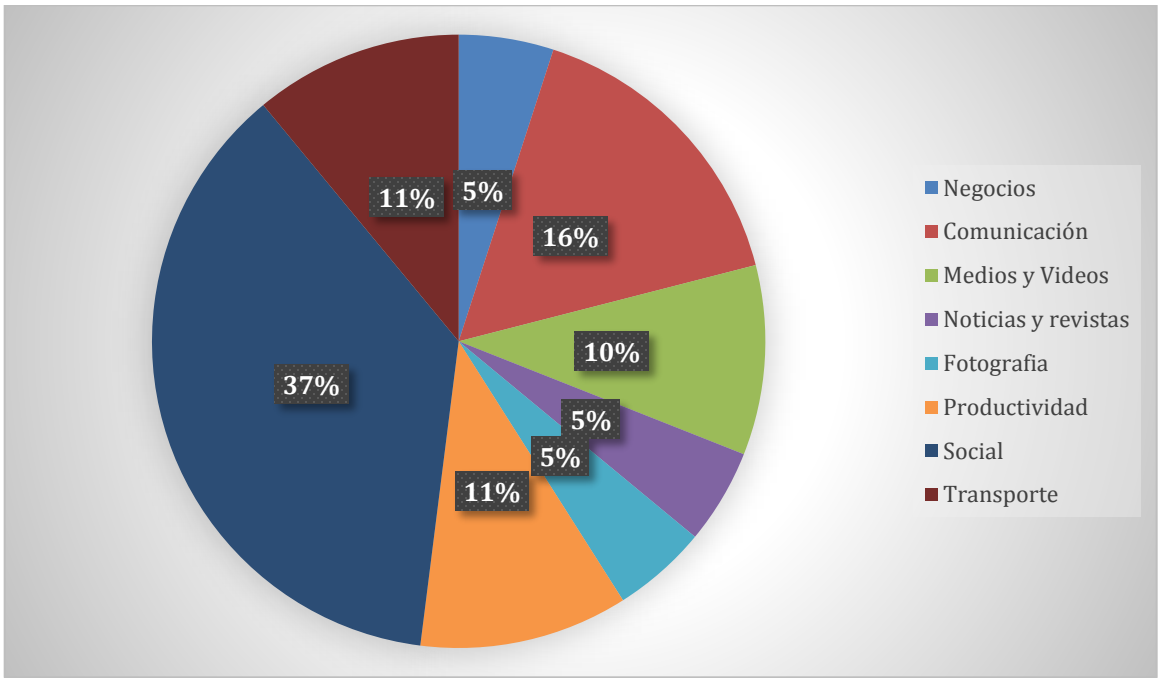


Figura 15. Categorías de las aplicaciones vulnerables

La Figura 15 muestra las categorías de aplicaciones vulnerables. 37% de las aplicaciones vulnerables pertenecen a la categoría de redes sociales. La Tabla 5 proporciona una fracción de estas aplicaciones vulnerables. Estas aplicaciones son todas populares en el mercado de Google Play. Viber y Twitter se han descargado 500 millones de veces en Google Play.

Componentes de invocación privada utilizando NVI

El resultado de NIVAnalyzer proporciona el nombre del componente vulnerable y la siguiente clave de intención, lo que facilita es explotar el NIV en algunas aplicaciones. Twitter es una famosa aplicación social con más de 500 millones de descargas en Google Play. Creamos una intención de exploit para iniciar el vulnerable LoginActivity, en la que construimos otra intención para iniciar la actividad privada bajo la clave "android.intent.extra.INTENT". LoginActivity requiere que el usuario inicie sesión con una cuenta existente o se registre una cuenta nueva.

Nuestros resultados muestran que algunos proyectos de código abierto y SDK de terceros también son vulnerables. Por ejemplo, K-9 mail 5 es un cliente de correo electrónico de código abierto para Android con más de 5 millones de descargas en Google Play.

1 play.google.com/store/apps/details?id=com.twitter.android
2 play.google.com/store/apps/details?id=com.psynet
3 play.google.com/store/apps/details?id=com.infraware.of fi ce.link
4 play.google.com/store/apps/details?id=com.evernote
5 play.google.com/store/apps/details?id=com.fsck.k9

Tabla 5. Algunas aplicaciones y componentes vulnerables descubiertos por NIVAnalizer

Nombre de la Aplicación	Componente	Tipo	Siguiente clave de intención	Descargas
MeetMe	LaunchActivity	Actividad	com.myyearbook.m.extra. RESTART_INTENT	10M+
Cámara IP	Rolling	Actividad	returnto	10M+
Oficina Polaris	FmLauncherActivity	Actividad	runIntent	10M+
Navegador DB	WebAccessActivity	Actividad	de.bahn.dbtickets.extra.IS_ BACK_TO_ ORDER	10M+
Charla de la abeja	BTSplashActivity	Actividad	_sys_ intent	10M+
Momento Cam	NotificationBroadcastReceiver	Receptor	REAL_INTENT	10M+
Evernote	LandingActivity	Actividad	EXTRA_LOGIN_PRESERVED_INTENT	10M+
Viber	ConversationActivity	Actividad	back _intent	10M+
Twitter	LoginActivity	Actividad	android.intent.extra.INTENT	10M+

Aplicaciones se extienden a partir de ella. Uno de sus receptores públicos BootReceiver es vulnerable. Una intención creada con **action"com.fsck.k9.service.BroadcastReceiver.schedule-Intent"** puede iniciar **BfootReceiver**. **BootReceiver** extrae la intención de destino en la clave"com.fsck.k9.service.BroadcastReceiver.pending Intent". Después de eso BootReceiver construye una nueva intención para iniciarse, en la que guarda la intención de destino extraída en la clave "com.fsck.k9.service.BroadcastReceiver.fireIntent".

Utiliza los servicios de alarma del sistema, que pueden activar intenciones en la hora programada o después del intervalo de tiempo especificado incluso si el componente actual ya no está activo. Por último, BootReceiver recupera la intención de destino bajo la clave“com.fsck.k9.service.BroadcastReceiver.fireIntent ” y la pasa al comenzar el servicio de la aplicación del sistema.

Utanbaby6 es una aplicación para Android dedicada a las madres jóvenes. La descripción oficial muestra que 20 millones de madres jóvenes la utilizan para comunicarse, compartir y aprender durante el embarazo y la paternidad en China. Confirmamos NIV en el público LoginActivity. Encontramos que 6 actividades privadas recuperan un valor de cadena de la intención de lanzamiento como una dirección URL, que luego se carga en el WebView. Al manipular la intención de lanzamiento, el atacante puede abrir páginas web arbitrarias mediante la aplicación vulnerable. Lo que es peor, algunas actividades adjuntan información confidencial en la solicitud. JingpinWebActivity y ShopWebActivity adjuntan un encabezado de autenticación a la solicitud.

Además, el atacante puede perturbar algunos flujos lógicos de la aplicación. DjBuyDialogActivity recupera un valor de cadena "picurl" que se guarda como una dirección URL y se usa para cargar la foto de un producto. Este valor de cadena puede ser manipulado por el atacante. El nombre y el precio se establecen de la misma manera. MallMyRebateActivity se usa para mostrar la devolución, que también se recupera de la intención de lanzamiento. Por último, muchas actividades pasan valores "nulo" mediante la intención de lanzamiento a métodos que no pueden controlar el valor "nulo". Esto hace que el cierre de la aplicación, tenga el mismo efecto que un ataque de denegación de servicio.

En un trabajo futuro, vamos a combinar las técnicas de ejecución simbólicos para construir la intención de explotar de una manera más inteligente. La eficiencia y la precisión del análisis de flujo de intención podrían mejorarse aún más. Y la automatización de la interfaz de usuario todavía tiene un poco de margen de mejora mediante el análisis y la categorización de más páginas de inicio de sesión.

Veliz et al.,(2016) Afirma que los problemas de seguridad a los que los dispositivos móviles están expuestos son similares a los que está expuesto una computadora, pero se ven agravados ya que cuentan con una mayor exposición al ser su comunidad de usuarios más amplia, que los utiliza tanto en el ámbito laboral como en el personal. Las amenazas que actualmente afectan a nuestros dispositivos móviles, se realizará una descripción de estas amenazas y se complementarán con noticias actuales que referencian a ataques realizados. Se hablará de las nuevas problemáticas que las nuevas tecnologías como el IoT traerán a la escena.

La problemática se potencia tanto debido al desconocimiento general sobre los problemas de seguridad a los que los dispositivos están expuestos como a la falta de información en las contramedidas que son posibles de adoptar.

Estas amenazas, suelen estar asociadas a problemas de seguridad en aplicaciones móviles descritos en el Top Ten de OWASP. OWASP Mobile Security Project es un proyecto que tiene la intención de brindar recursos necesarios para que las aplicaciones móviles sean más seguras. A través de este proyecto, el objetivo es clasificar los riesgos de seguridad móviles y proporcionar controles en el desarrollo para reducir su impacto y probabilidad de explotación.

La metodología OWASP Mobile resulta interesante para estudiar y aplicar, con la finalidad de disminuir los riesgos y vulnerabilidades y evitar posibles ataques o robos de información.

MALWARE

Malware es una clasificación general de software malintencionado en la que se incluyen los virus, rootkits, troyanos, etc., y es más comúnmente relacionado con la destrucción o robo de datos. La definición de malware ha cambiado durante los últimos años y se puede dividir en dos categorías: tradicional o moderno. El malware tradicional se refiere a la forma clásica de malware en la que el objetivo es infectar y propagarse al mayor número de dispositivos posible, al tiempo que maximiza el daño al sistema, sin ningún motivo específico detrás de un ataque. Los malware actuales son más sofisticados y estratégicamente planificados.

Un ejemplo de malware moderno son los ataques APT (Amenaza persistente avanzada), cada vez más conocidos. Un ataque APT es un ciber-ataque bien planeado con un objetivo específico, como por ejemplo una organización o un sistema de un gobierno. El ataque se lleva a cabo en múltiples pasos para cada una de las metas específicas y es intencionalmente sigiloso con el fin de evitar la detección.

Una de las principales diferencias del malware moderno en comparación con el tradicional, es que el malware moderno está dirigido y es sigiloso. El objetivo es elegido específicamente ya que tiene algo de valor para el atacante y evitar la detección es esencial con el fin de llevar a cabo un ataque prolongado.

Un ataque APT puede durar durante un largo período de tiempo y mientras más datos detectados puedan ser extraídos mayor se considera la efectividad del ataque.

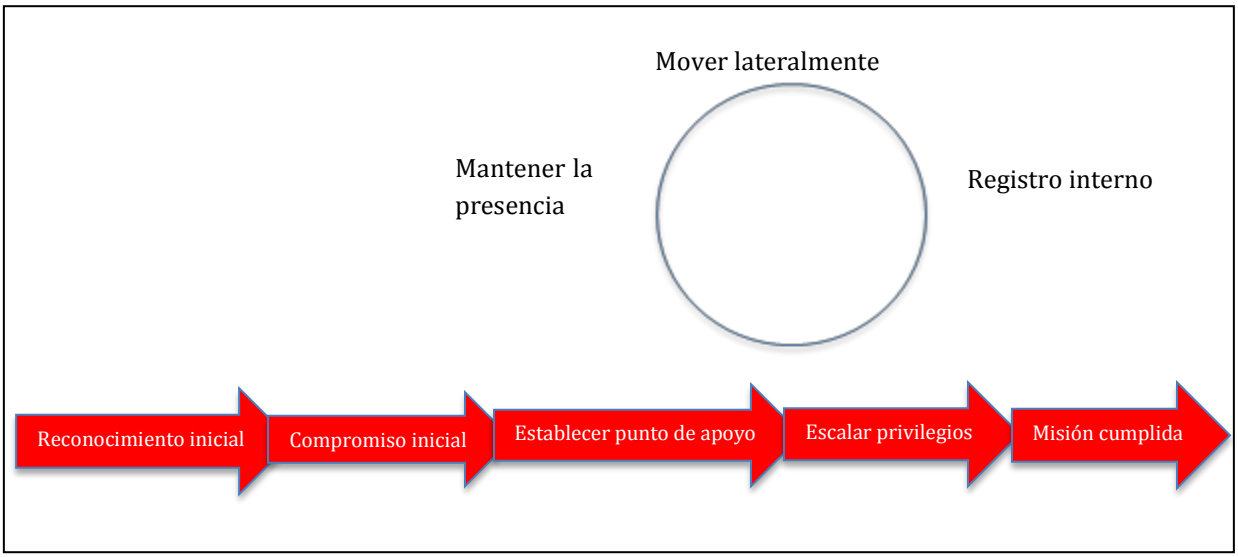


Figura 16. Ciclo de vida de un APT

Fuente: Veliz et al., (2016)

Las características del ciclo de vida APT pueden ser descriptas en la Figura 16, como un proceso secuencial que implica la investigación, la intrusión, la recolección de credenciales, la extracción de datos y la limpieza. El primer paso consiste en la investigación sobre el objetivo con el fin de encontrar puntos débiles, o puntos de entrada, en un perímetro de la organización y para determinar la información de interés que podría ser localizada. Un atacante busca las vulnerabilidades y luego las explota para establecer una penetración inicial utilizando técnicas como Phishing o ingeniería social. Una vez dentro, el malware intenta obtener las credenciales requeridas con el fin de obtener acceso a los datos específicos. Los datos se extraen finalmente. Cuando la extracción se completa entonces el malware se limpia a sí mismo para cubrir sus pasos y no dejar rastros. La intención es intentar que tome semanas o incluso meses antes de que alguien descubra que un ataque se ha llevado a cabo.

Un ejemplo de una APT contra Android tuvo lugar en el año 2013, dirigido contra un grupo de activistas tibetanos. El ataque consistió en el uso de una cuenta de correo electrónico comprometida con la que el atacante se las arregló para enviar un correo electrónico masivo acerca de una conferencia que incluyó un archivo .apk malicioso adjunto. Los datos robados fueron contactos, registros de llamadas, Mensajes SMS, geolocalización y otros datos del teléfono, como el número de teléfono, la versión del sistema operativo, modelo de teléfono y la versión SDK.

VULNERABILIDADES DE SOFTWARE

Las vulnerabilidades de software son un riesgo de seguridad constante para cualquier aplicación o plataforma. En el tiempo entre el descubrimiento de una vulnerabilidad hasta que sea arreglada y parcheada, una aplicación corre el riesgo de ser explotada. Los navegadores Web son un ejemplo típico y que se puede encontrar básicamente en cualquier PC o dispositivo móvil. Si los autores de malware descubren una falla, entonces pueden explotarla con el fin de inyectar código malicioso que puede afectar a millones de dispositivos hasta el momento en que el defecto sea parcheado en el sistema del usuario.

Hoy en día se manejan términos como el 0-day vulnerability (vulnerabilidad de día cero), que se refiere a una nueva vulnerabilidad para la cual aún no se crean parches o revisiones, y que se emplea para llevar a cabo un ataque. El nombre 0-day (día cero) se debe a que aún no existe ninguna revisión para mitigar el aprovechamiento de la vulnerabilidad. Estas a veces se usan junto a los troyanos, rootkits, virus, gusanos y otros tipos de malware, para ayudarlos a propagarse e infectar más equipos. También se puede encontrar escrito como "0 day", "zero day" y "zero-day".

Las vulnerabilidades de día cero, o zero-day son de las más peligrosas, ya que los atacantes pueden

2.5.3 DETECCIÓN DE VULNERABILIDADES EN APLICACIONES BASADAS EN GESTOS

Wang et al., (2016) se centra en las vulnerabilidades causadas por desarrolladores de aplicaciones que no entienden completamente la influencia del sistema Android en el bloqueo basado en gestos. Primero hacemos un análisis estático en las aplicaciones vulnerables con funciones de bloqueo basadas en gestos y, a continuación, observamos las características comunes en estas aplicaciones. Específicamente, encontramos que la actividad registrada en manifest.xml, tiene dos atributos mostrados en la Figura 17, que tienen una influencia importante en el bloqueo basado en gestos.

```
<actividad
android: name=". Actividad principal"
android: label=" @ cadena / nombre_api"
Android: exportado=" cierto"
Android: launchMode= "singleTask ">
</ Actividad>
```

Figura 17. Los atributos relacionados con la actividad

Con base en el análisis, desarrollamos una herramienta llamada LockBreaker para detectar automáticamente las vulnerabilidades de bloqueo basado en gestos después de analizar la influencia de los atributos de la actividad en él.

Para proteger la seguridad de la interfaz, el bloqueo basado en gestos, que sirve como la primera línea de defensa, ha sido ampliamente utilizado en muchas aplicaciones financieras, de pago y de redes sociales. Más específicamente, el bloqueo basado en gestos se implementa para evitar acceso no autorizado a la cuenta predeterminada en las aplicaciones. Por ejemplo, al limitar el número de intentos de inicio de sesión para ser más precisos 5 un bloqueo basado en gestos, la aplicación rechazará a un usuario para que inicie sesión utilizando la cuenta predeterminada si no puede introducir el gesto correcto dentro de 5 intentos.

Además, para defenderse de ataques de fuerza bruta, la cuenta predeterminada en una aplicación se borrará una vez que los intentos de inicio de sesión superen el valor preestablecido. Un bloqueo típico basado en gestos se muestra en la Figura 18.

Una cerradura típica basada en el gesto es como se muestra en la Figura 18.

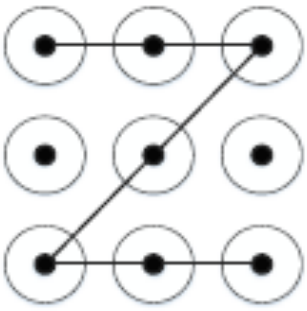


Figura 18. Cerradura basada en gestos de Android.

Sin embargo, encontramos que muchos bloqueos basados en gestos pueden ser anulados fácilmente debido a la existencia de vulnerabilidad en aplicaciones. Como resultado, la cuenta de usuario corre un grave riesgo de ser divulgada, lo que puede conducir a pérdidas financieras, personales o violaciones de privacidad. La mayoría de las vulnerabilidades se producen ya que desarrolladores no entienden completamente la influencia de atributos de la actividad en la seguridad del bloqueo basado en gestos.

Desarrollamos Lock-Breaker para explorar automáticamente las vulnerabilidades basadas en gestos. Utilizando Lock-Breaker, detectamos 13 vulnerabilidades en diferentes aplicaciones, que tienen cientos de millones de usuarios. Proponemos algunos consejos sobre cómo evitar y corregir las vulnerabilidades para mejorar la seguridad de bloqueo basado en gestos.

El mecanismo ICC (Comunicación de componentes internos) está diseñado para intercambiar datos entre componentes. La clave de ICC es la intención que se introduce como una carga útil del mensaje entre componentes. Hay dos tipos de intenciones: la intención explícita y la intención implícita. Ambos pueden iniciar un componente y llevar mientras tanto datos de comunicación. Las comunicaciones entre diferentes actividades también se basan en intención. La Figura 19 ilustra el modelo de comunicación entre las actividades.

Cada aplicación tiene un archivo AndroidManifest.xml en su directorio raíz. El archivo del manifiesto presenta información esencial relacionada sobre la aplicación al sistema Android.

Las configuraciones estáticas sobre la aplicación también se incluyen en este archivo. Solo se consideran válidos los componentes que se han registrado en el archivo del manifiesto. Una actividad válida debe registrarse como un nodo secundario en el archivo del manifiesto según corresponda. El nodo de actividad tiene muchos atributos que determinan los comportamientos de la actividad

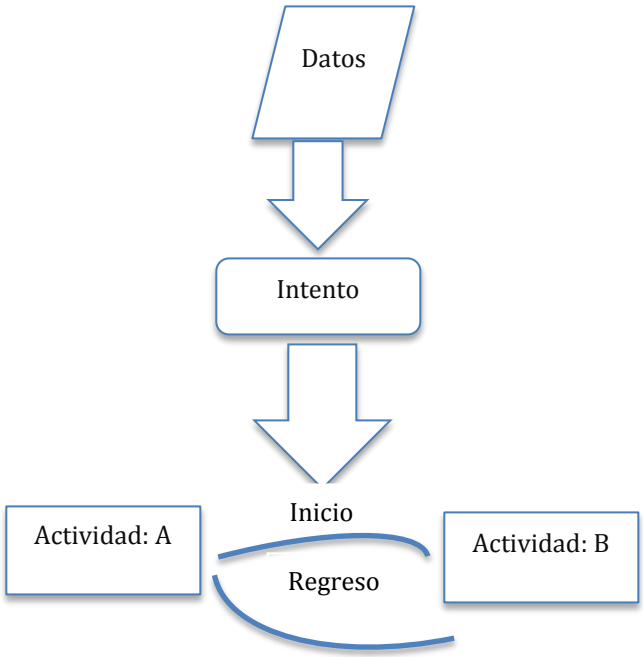


Figura 19. La comunicación entre actividades

Las actividades siempre se inician dentro de una aplicación para crear una ventana interactiva para los usuarios. Sin embargo, diferentes aplicaciones puede necesitar compartir ventanas para mayor comodidad. En este circunstancia, una actividad debe iniciarse desde un proceso externo (la actividad se denomina actividad exportada). Por ejemplo, PayPal proporciona actividades de pago para aplicaciones de compras.

El bloqueo basado en gestos está diseñado para mejorar la seguridad de cuenta de usuario desde una perspectiva local. De forma predeterminada, la mayoría de las aplicaciones guardan la cuenta y la contraseña del último usuario para mayor comodidad. Cuando un atacante obtiene el teléfono del usuario, la cuenta predeterminada estará en peligro ya que el atacante puede hacer algunas ofertas utilizando la cuenta recordada por la aplicación.

La privacidad y la seguridad de la propiedad del usuario se verán afectadas si la aplicación está relacionada con el pago, los fondos o las redes sociales. El bloqueo basado en gestos está diseñado para evitar el acceso no autorizado a la cuenta predeterminada en las aplicaciones. Cuando alguien intenta iniciar sesión con la cuenta predeterminada, la aplicación iniciará la actividad de gestos (Definimos la actividad que implementa el bloqueo basado en gestos como Actividad de gestos) y la solicitud de entrada de gestos.

Si sus intentos de inicio de sesión superan el valor predefinido, se identificará como un usuario no válido para la cuenta predeterminada. La cuenta preguardada en la aplicación se borrará para evitar que sea hackeada.

La implementación del bloqueo basado en gestos se basa en la pila de actividad. Es difícil obtener la contraseña de gesto de un proceso externo porque los datos de gestos se almacenan normalmente en forma cifrada y protegida por el mecanismo sandbox de Android. Un desarrollador puede usar una variable (la llamamos estado de bloqueo) para describir el estado del bloqueo basado en gestos. El tipo de estado de bloqueo puede ser Booleano ya que solo hay dos valores de esta variable.

El valor predeterminado de lockState es falso, lo que significa que la aplicación actual no se ha desbloqueado. El valor se establecerá como verdadero solo si se introduce correcto. Cada vez que se inicia una actividad de la aplicación, el programa debe comprobar el valor de lockState. Siempre que el valor sea falso, el sistema iniciará actividad de gestos para evitar que el usuario inicie el registro de la aplicación.

- Las amenazas de Actividades exportadas

A veces se debe establecer una actividad para que pueda proporcionar servicios para otras aplicaciones. Las actividades exportadas pueden tener una mala influencia en el bloqueo basado en gestos. Según el principal mecanismo de protección de bloqueo basado en gestos, todas las actividades que contienen datos confidenciales o empresas críticas deben protegerse mediante bloqueo basado en gestos.

Sin embargo, muchas implementaciones de bloqueo basado en gestos no obedecen estrictamente el principio de protección de que algunas actividades exportadas en una aplicación se pueden iniciar correctamente desde un proceso externo. Definimos estas actividades desprotegidas como amenazas a la seguridad. Una vez que la actividad contiene datos sensibles o negocios críticos, se define como una vulnerabilidad.

Más especialmente, si la actividad tiene una posición importante en el gráfico de transición de actividad, lo que significa que una ruta de acceso de esta actividad o a cualquier otra actividad de la aplicación está disponible, el bloqueo basado en gestos se omitirá como resultado.

Es llamada a una actividad exportada que no está protegida por el bloqueo basado en gestos EDActivity (Exportado y Actividad Peligrosa). De forma similar, llaman a una actividad ESActivity (Exportado pero Actividad segura) si no se puede iniciar desde un proceso de aplicación externo. Las amenazas de seguridad resultantes de las actividades exportadas se definen como TEA (Amenazas de actividad exportada).

El objetivo de esta investigación es detectar vulnerabilidades y amenazas sobre bloqueo basado en gestos en aplicaciones de Android. La Figura 20 proporciona una vista general de nuestro sistema llamado Interruptor de bloqueo. El Interruptor de bloqueo es una herramienta automática diseñada para detectar vulnerabilidades en el bloqueo basado en gestos. Consta de tres componentes principales:

a) **Analizador estático:** Analiza estáticamente la aplicación de prueba y busca todas las actividades exportadas en el archivo de manifiesto.

b) **Clasificador de componentes:** Reconoce la actividad de gestos desde la aplicación de prueba y clasifica las actividades exportadas como ESActivities y EDActivities.

c) **Comprobador de vulnerabilidades:** Identifica las amenazas y vulnerabilidades de EDActivities y genera informes de resultados.

Un componente ejecutable en una aplicación debe ser registrado como un nodo en el archivo de manifiesto. Un archivo de manifiesto contiene toda la información sobre un componente. Para obtener actividades exportadas de una aplicación, debemos analizar el archivo manifiesto.

a) Analizador estático

Un componente ejecutable en una aplicación debe ser registrado como un nodo en el archivo del manifiesto. Un archivo manifiesto contiene toda la información sobre un componente. Para obtener actividades exportadas de una aplicación, debemos analizar el archivo manifiesto.

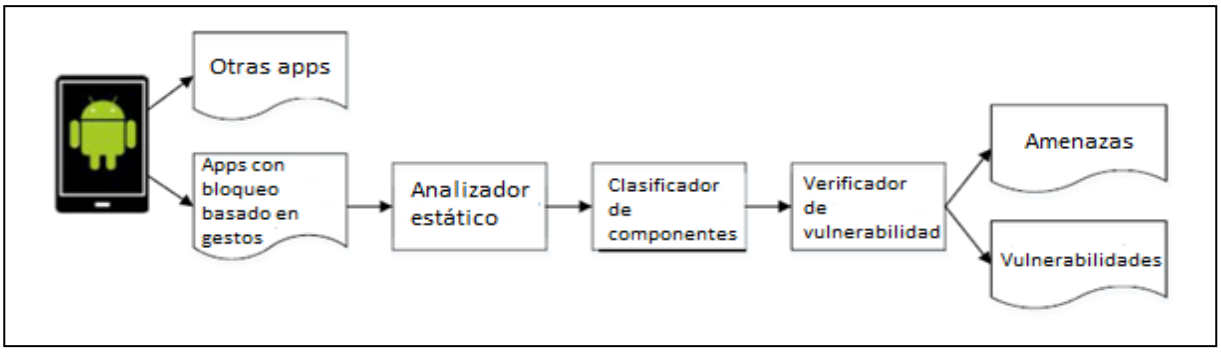


Figura 20. Descripción general del interruptor de bloqueo.

Hasta nuestro conocimiento, el archivo de manifiesto se compila en binario que forma el paquete de aplicación sufrido por el APK. El archivo será ilegible si solamente lo extraemos del paquete APK.

Apktool es una herramienta para la ingeniería inversa para los archivos APK en Android. Esta herramienta puede ayudarnos a obtener el archivo de manifiesto en una forma descompilada, pero también influirá en el grado de automatización del interruptor de bloqueo.

b) Componente clasificador

El Componente clasificador se encarga de dos trabajos: Averiguar las actividades de gestos en una aplicación y clasificar las actividades exportadas como ESActivities y EDActivities.

Cuando un usuario hace clic en un icono de aplicación en el que se ha abierto la función de bloqueo basada en gestos, la aplicación responde de la siguiente manera: iniciar la actividad asociada con la intención de lanzamiento; inicia algunas actividades contra salpicaduras que se utilizan para mostrar anuncios si es necesario (la actividad de salpicaduras significa que se destruirá en pocos segundos); inicia la actividad gesto y lo mostrará en la pantalla hasta que se introduce un gesto correcto.

Una vez que la actividad de gesto se reconoce desde la aplicación, El componente clasificador realiza el segundo trabajo. Como se muestra en la Figura 21, el sistema intenta iniciar una actividad exportada y comprueba si se ha iniciado correctamente comparando la actividad en primer plano con la actividad de gestos. Si una actividad exportada se iniciado correctamente, es un EDActivity. De lo contrario, es una ESActivity.

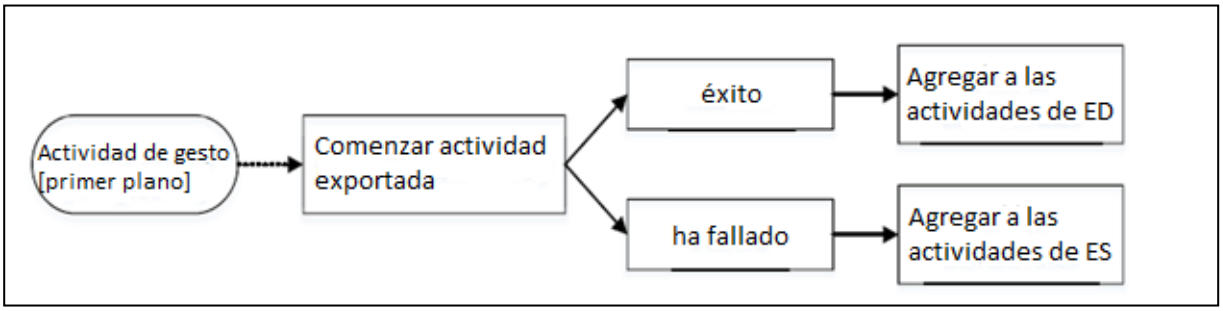


Figura 21. Clasificación de las actividades exportadas

El clasificador de componentes se pasará al Comprobador de vulnerabilidades para más pruebas. El comprobador de vulnerabilidades se encarga de juzgar si una EDActivity es una vulnerabilidad. La Figura 22 muestra el diseño del comprobador de vulnerabilidades. En primer lugar inicia un EDActivity como un intento de destruir el bloqueo basado en gestos. Para comprobar si la actividad de gesto se destruye mediante la EDActivity, el sistema intenta iniciar una ESActivity y consulta la actividad en primer plano.

En otras palabras, el EDActivity es una vulnerabilidad que destruye el mecanismo de protección de bloqueo basado en gestos. Si el ESActivity todavía no se puede iniciar, a continuación, concluirá que la protección del bloqueo basado en gestos sigue siendo existente. En esta circunstancia, se necesitan algunas intervenciones manuales para especificar las vulnerabilidades, como la fuga de privacidad, la exposición de datos sensibles y la vulnerabilidad lógica.

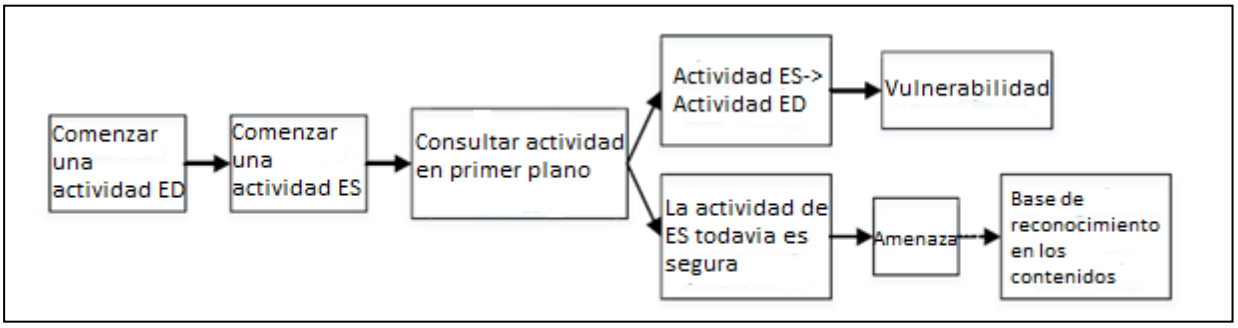


Figura 22. Procedimiento de identificación de Verificador de Vulnerabilidad

2.5.4 EVALUACIÓN

Se presentan los resultados de la evaluación de la ejecución de bloqueo del interruptor en un conjunto de 63 aplicaciones del mundo real. Las versiones de estas aplicaciones son todos tardar en el momento de finalizar nuestra prueba.

Seleccionamos 63 aplicaciones que tienen bloqueo basado en gestos como nuestras muestras de prueba. La mayoría de estas aplicaciones tienen un gran número de usuarios y generalmente se consideran relativamente seguras. Estas aplicaciones están relacionadas con fondos o con privacidad. Probamos estas aplicaciones usando el interruptor de bloqueo y encontramos varias vulnerabilidades.

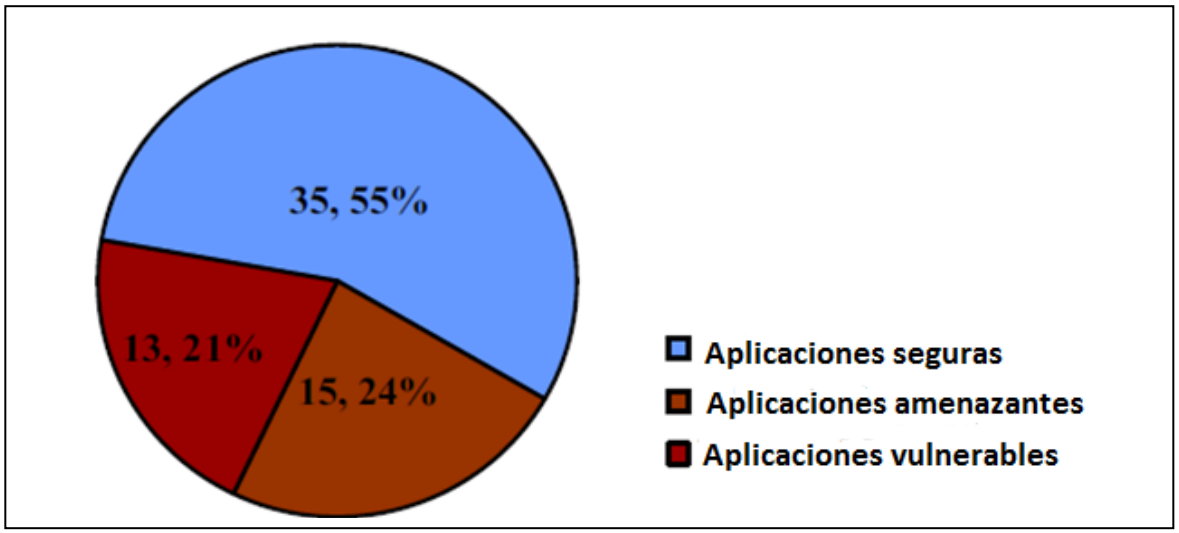


Figura 23. Amenazas y vulnerabilidades detectadas por el interruptor de bloqueo

La Figura 23 muestra el resultado de nuestra prueba. Entre las 63 aplicaciones, 28 de ellas se enfrentan a la amenaza de TEA. Además, el módulo de comprobador de vulnerabilidades encontró que 13 de las 28 aplicaciones tienen vulnerabilidades evidentes. Los detalles de las 13 aplicaciones vulnerables se muestran en la Tabla I. Excepto QQ y XiaoNiuFinance, vulnerabilidades en el resto de las aplicaciones vulnerables provocan el desvío del bloqueo basado en gestos.

Específicamente, la vulnerabilidad de QQ hace que el número de teléfono asociado con su cuenta QQ se puede cambiar arbitrariamente por el atacante. Como QQ también permite a un usuario cambiar su contraseña a través del teléfono, el atacante controlará completamente la cuenta del usuario. Además, XiaoNiuFinance filtra la información de los contactos almacenados en el teléfono.

Los resultados muestran que el interruptor de bloqueo ha detectado correctamente muchas vulnerabilidades no publicadas. En gran medida, llegamos de concluir que Lock-Breaker es eficaz en la detección vulnerabilidades en el bloqueo basado en gestos.

A continuación se muestran en la Tabla 6 la detección de las vulnerabilidades existentes basadas en gestos, así como el nombre del paquete de la aplicación la versión donde está siendo explotada así como la cuenta de usuario correspondiente.

Nombre del paquete	Versión	Cuenta de usuario
com.tencent.mobileqq	v6.2.3.2700	430,000,000
com.chinatelecom.bestpayclient	v5.0.9	15,100,000
com.xiaomi.jr	v2.0.4	6170,000
com.zhangdan.app	v8.2.2	2580,000
com.suning.mobile.epa	v5.2.0	520,000
co.welab.wolaidai	v4.5.3	400,000
com.xiangshang.xiangshang	v2.9.0	270,000
co.welab.wolaidai	v3.6.1	260,000
com.xiaoniu.finance	v2.2.0	190,000
com.qianbaomu.view	v1.7.4	180,000
cn.jac.fund	v3.7.0	57,000
com.rrjc.activity	v2.1.2	12,000
com.tcg.very.wallet	v1.1	2,000

Tabla 6. Vulnerabilidades existentes basadas en gestos.

Las vulnerabilidades sobre el bloqueo basado en gestos son ampliamente existentes en la historia. Dividimos estas vulnerabilidades en cuatro categorías según las causas: el gesto se almacena localmente en forma de texto claro; el gesto se almacena localmente en forma de texto cifrado pero es accesible y se puede descifrar; ignorancia del desarrollador sobre la influencia del sistema Android y otros.

Aprovechamos el interruptor de bloqueo para probar las vulnerabilidades existentes que pertenecen al tercer tipo ya que los otros tres están más allá de nuestra investigación. Seleccionamos cinco vulnerabilidades cuyos detalles se muestran en la Tabla 6 como datos de prueba. Estas vulnerabilidades son detectadas con éxito por el interruptor de bloqueo.

Tabla 7. Detalles existentes de las vulnerabilidades. Es una organización china que proporciona el número Wooyun para cada vulnerabilidad como identificador de la misma (CVE) con el respectivo nombre del paquete que está siendo explotado.

ID DE LA VULNERABILIDAD	NOMBRE DEL PAQUETE
WooYun-2015-158586	com.chinatelecom.bestpayclient
WooYun-2015-101170	com.wacai365
WooYun-2015-101182	com.mymoney
WooYun-2015-111692	com.umetrip.android.msky.app
WooYun-2015-120120	com.zrb

Tabla 7. Detalles existentes de las vulnerabilidades

1) Evitar la vulnerabilidad

El desarrollador deberá establecer las actividades exportadas lo menos posible para reducir las amenazas procedentes de las actividades exportadas. Para actividades que se deben exportar, se propone un modelo de protección para garantizar la seguridad del bloqueo basado en gestos. Como se puede ver en la Figura 24, cuando una actividad exportada se inicia desde un método externo por startActivity() o startActivityForResult(), proporcionamos dos métodos para evitar que se muestre en la pantalla.

Para evitar las vulnerabilidades de bloqueo basado en el gesto, un desarrollador debe entender la influencia del sistema Android. Le damos nuestro consejo a partir de dos aspectos.

La aplicación supervisa todos los intentos que están asociados con actividades exportados. Se inicia la actividad gesto como una autenticación cada vez que se va a iniciar una actividad exportado. Por lo tanto, la actividad se exporta efectivamente controlada por la actividad gesto. La aplicación comprueba el estado del bloqueo basado en gestos inmediatamente después de que se inicie una actividad exportada, y comienza la actividad gesto para cubrir la actividad exportada en la pantalla.

Como resultado, el bloqueo basado en gestos no se ve afectado por las actividades exportadas porque nunca se pueden ver visualmente. En segundo lugar, un desarrollador debe asegurarse de que las actividades cuyos modos de lanzamiento son tarea única o instancia única se puede iniciar antes de la actividad de gesto porque podrían destruir la actividad de gesto cuando se invoca la instancia desde la tarea.

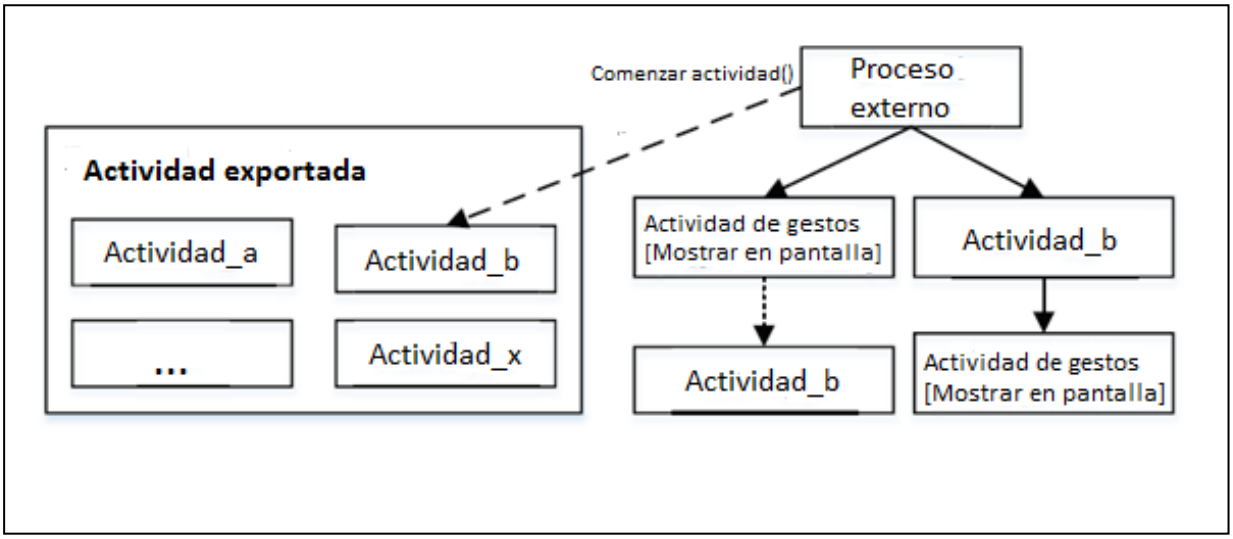


Figura 24. Defensa sobre TEA en bloqueo basado en gestos

2) Solucionar la vulnerabilidad

Para las vulnerabilidades existentes en el bloqueo basado en gestos, proponemos dos soluciones para ayudar a solucionarlas. Si el bloqueo basado en gestos contiene una vulnerabilidad, Lock-Breaker notificará la actividad vulnerable, a saber, la actividad exportada. Por lo tanto, la forma más fácil de corregir una vulnerabilidad basados en gestos es cambiar el valor de exportación= atributo de verdadero a falso. La actividad de gestos no se puede destruir, ya que no se puede iniciar la actividad vulnerable desde un proceso externo.

Para las vulnerabilidades existentes en la cerradura basada en el gesto, se proponen dos soluciones para ayudar a solucionarlos.

Se propone solucionar la vulnerabilidad si es necesario exportar la actividad vulnerable. Como se muestra en la Figura 25, un desarrollador de aplicaciones debe encontrar la ruta de transición de la actividad iniciada a partir de la actividad vulnerable que se notifica mediante el interruptor de bloqueo. En la ruta de acceso de transición de actividad, debe haber una actividad cuyo modo de lanzamiento sea una tarea única o instancia única. La vulnerabilidad se soluciona, siempre y cuando cambie su modo de lanzamiento es modo estándar o único top.

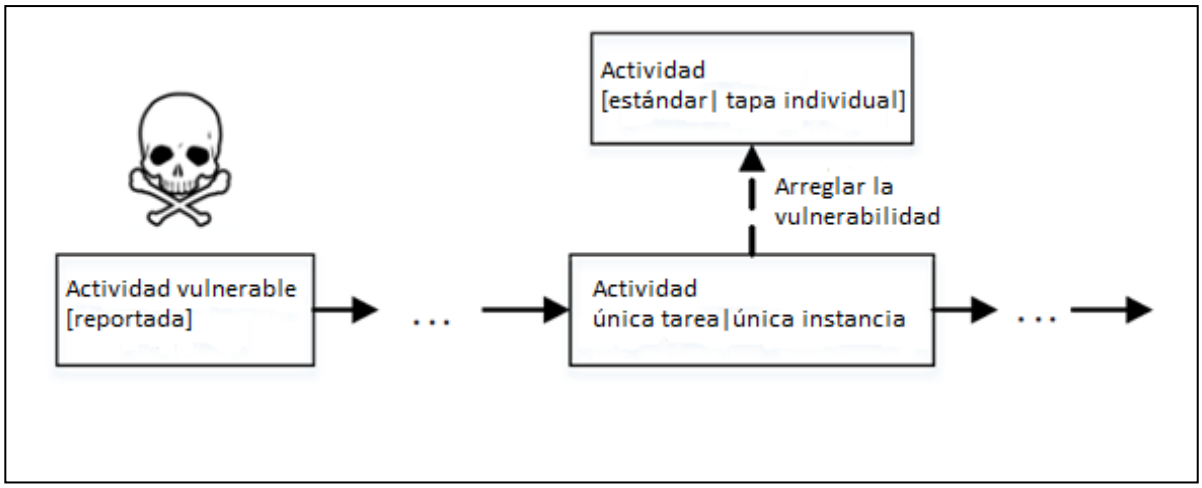


Figura 25. Corrige la vulnerabilidad en la ruta de transición de la actividad

2.6 VULNERABILIDADES EN APLICACIONES MÓVILES

Los teléfonos inteligentes son un medio también para la propagación de virus y gusanos móviles, existe multitud de vulnerabilidades en el uso de redes móviles infectadas con malware al igual que, para él envió de spam y ciberdelincuencia, los medios de propagación por los móviles son diversos, por ejemplo (Anaya, 2015):

- Acceso a internet, que tienen código malicioso, Drive by Download
- Conexión inalámbrica Bluetooth o Wi-Fi
- Mensajería tradicional SMS y MMS
- Redes de móviles que envían mensajes spam y no deseados
- Aplicaciones de comunicación instantánea como WhatsApp, Instagram, etc.
- Acceso al correo electrónico
- Puertas traseras, troyanos
- Descargas de internet, sobre todo el sistema operativo Android, debido a que el software instalado en el dispositivo, no pasa los controles de calidad que debe pasar por ejemplo en IOS.

Como se indica en el informe de ESET Latinoamérica Tendencias 2014: el desafío de la privacidad en Internet, las amenazas informáticas para estos dispositivos continúa aumentando, puesto que se observa el incremento de software malicioso en los dispositivos móviles, especialmente en los que funcionan con el sistema operativo Android.

La consolidación de Android como la plataforma móvil más utilizada (con aproximadamente el 80% del mercado), la ha convertido en un objetivo primordial para los cibercriminales, y como consecuencia nuevas familias de malware y variantes que componen cada una son continuamente detectadas.

Los primeros casos incluyen troyanos espía, SMS, que buscan convertir el dispositivo en zombi para formar parte de una botnet, o malware del tipo filecoder como Simplocker. A partir del año 2014 se reportan subcategorías de troyanos que sólo se identificaban en plataformas como Windows, tal es el caso de downloders, droppers, clickers o bancarios para obtener datos relacionados con entidades financieras (Welive security, 2018).

2.6.1 SOFTWARE MALICIOSO

Escobar et al., (2015) dice que el sistema operativo Android en dispositivos móviles es uno de los más usados en el Mercado con un porcentaje de 48.96 %. La vulnerabilidad, también entendida como fallos de seguridad, está relacionada con todo aquello que altera o provoca perdida de información mediante amenazas en un sistema. Los incidentes de seguridad en dispositivos con S.O. Android se han incrementado de manera alarmante debido a la preferencia de sus usuarios, esto los hace más vulnerables de ser atacados. Se presenta un alto riesgo de vulnerabilidad en la violación de información confidencial de las personas que administran estos dispositivos. De acuerdo con esto, en este artículo se pretende reflexionar acerca de los aspectos más relevantes en dicho problema de seguridad que afecta a las terminales con S.O. Android, y que se describen las amenazas más frecuentes que se presentan. Además, se dan posibles recomendaciones para lograr minimizar los riesgos de seguridad que se presentan en estas terminales.

Perú y Ecuador continuando liderando este ranking. Debajo queda Colombia (63%). Chile (17%) y Argentina (20%), Dando paso a Bolivia, Paraguay y México. (ESET, 2014).

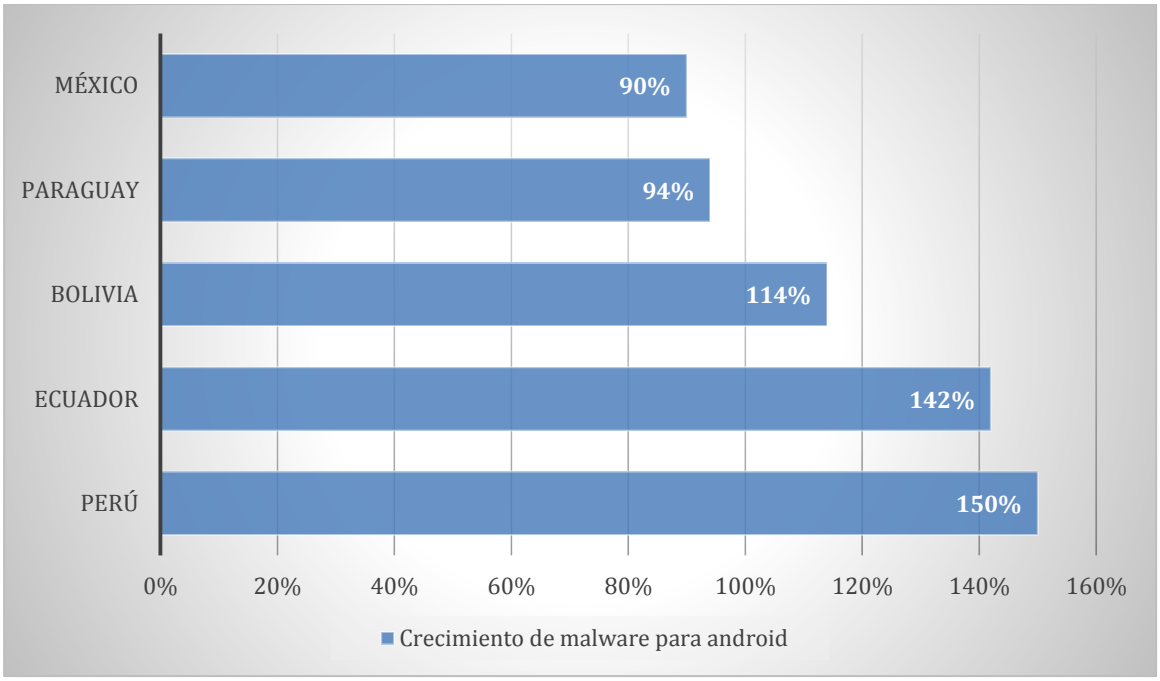


Figura 26. Crecimiento de Malware en América Latina (ESET, 2014).

Se entiende por vulnerabilidad o fallo de seguridad todo aquello que provoca que los sistemas informáticos funcionen de manera diferente a lo programado, afectando la seguridad y pudiendo llegar a provocar la pérdida y/o robo de información sensible. (Crespo y Ramos, 2012).

Estas vulnerabilidades pueden ser el resultado de controles técnicos inadecuados, pero también puede ser consecuencia de las malas prácticas de seguridad de los consumidores" (Pc Word, 2009).

A continuación se describe varias vulnerabilidades conocidas y que son las más comunes en las plataformas móviles.

Los dispositivos móviles no tienen contraseñas habilitadas. Muchos de los usuarios, al adquirir estos dispositivos móviles, no registran el acceso con contraseñas para poder salvaguardar los datos almacenados. Muchos dispositivos tienen la capacidad técnica para soportar contraseñas, números de identificación personal (PIN) o patrones de clave en la pantalla para la autenticación. Algunos dispositivos móviles también incluyen un lector biométrico para escanear una huella dactilar como autenticación. (Pc Word, 2009).

Las transmisiones inalámbricas no siempre están encriptadas. Las redes públicas o gratis de Wi-Fi, no solo están abiertas a los usuarios, sino también a ataques de piratas informáticos que intentan robar datos confidenciales.

La información generada por un dispositivo móvil no suelen estar cifrada durante su transmisión. Muchas aplicaciones no encriptan los datos que transmiten y reciben a través de la red, lo que facilita que puedan ser interceptados. Por ejemplo, si una aplicación está transmitiendo datos a través de una red WiFi sin encriptar y utilizando HTTP (en lugar de HTTP seguro), los datos pueden ser fácilmente interceptados. Cuando una transmisión inalámbrica no está cifrada, los datos pueden ser interceptados fácilmente. (Pc Word, 2009).

Los dispositivos móviles pueden contener malware. Malware son programas informáticos maliciosos con el fin de robar información privada (como datos personales o bancarios) o bien instalar programas y correr procesos sin tu consentimiento.

Los usuarios pueden descargar aplicaciones que contienen malware. Los consumidores descargan estos programas sin saberlo, ya que pueden estar disfrazados como un juego, parche de seguridad, utilidad o aplicación. Es difícil que los usuarios noten la diferencia entre una aplicación legítima y una que contenga un software malicioso. Cuando una transmisión inalámbrica no está cifrada, los datos pueden ser fácilmente interceptados por intrusos que pueden obtener acceso no autorizado a información sensible. (Pc Word, 2009).

Los dispositivos móviles a menudo no utilizan software de seguridad. Al adquirir un dispositivo móvil, las empresas distribuidoras no instalan un software para protegerse de aplicaciones como virus, troyanos, spyware y spam; además los usuarios, por desconocimiento del funcionamiento del móvil, no saben instalar dicho software, por ende no conocen la importancia de tener un aplicativo de seguridad.

Los canales de comunicación pueden estar mal asegurados. Uno de los problemas más comunes de seguridad es dejar abiertos los canales de comunicación. El Bluetooth, o en el modo 'descubrimiento' (que permite que el dispositivo sea visto por otros dispositivos compatibles con Bluetooth para que se puedan hacer conexiones), podría permitir que un atacante instale malware a través de esa conexión, o active subrepticiamente un micrófono o una cámara para espiar al usuario. (Pc Word, 2009).

Otro problema común es el uso de redes públicas o puntos de Internet inalámbrico Wi-Fi, lo que podría permitir que un atacante se conecte al dispositivo y vea la información sensible.

Descargar nuevas aplicaciones solamente a través de los canales oficiales de los fabricantes.

El software malicioso no es un problema exclusivo de los ordenadores, también afecto a los Smartphone y Tablets, por tanto, necesitan la misma protección que se utiliza en cualquier PC. Al instalar aplicaciones de cualquier otra fuente, como páginas web u otras tiendas de aplicaciones, estamos corriendo el riesgo de instalar aplicaciones maliciosas.

La mayor parte de los virus se "cuelan" en los dispositivos móviles a través de descargas de aplicaciones desde sitios poco fiables.

Para minimizar los riesgos de seguridad en los dispositivos móviles que usan Android, se recomienda usar Play Store -el canal oficial- para descargar aplicaciones. (Gestión, 2014)

El dispositivo móvil y las aplicaciones actualizadas.

Uno de los problemas principales de seguridad en los dispositivos móviles es tener desactualizadas sus apps. Un inconveniente (Cooney, 2012) de los parches de seguridad para las aplicaciones de terceros es que no siempre se desarrollan y se publican en el momento oportuno. Además, las aplicaciones móviles de terceros, incluyendo los navegadores web, no siempre notifican a los consumidores cuando hay actualizaciones disponibles. A diferencia de los navegadores web tradicionales, los navegadores móviles rara vez obtienen actualizaciones. El uso de software obsoleto aumenta el riesgo de que un atacante pueda explotar vulnerabilidades asociadas con estos dispositivos.

Protección ante el robo o la pérdida de un terminal móvil.

Los dispositivos móviles como los Smartphone y las Tablets correen un riesgo diario de robo o extravío; la solución ante este escenario no es dejar de usarlos, sino salvar la información contenida para que la pérdida del equipo no sea traumática.

Un dispositivo móvil puede contener información muy valiosa: datos de tarjetas de crédito, números de cuentas bancarias, contraseñas, datos de contacto, fotos, videos, correos electrónicos y una larga lista de información privada. Una información muy atractiva para cualquier ciberdelincuente. (Pc Word, 2009).

Los dispositivos móviles a menudo no utilizan firewall de seguridad.

Los dispositivos móviles como los teléfonos inteligentes y las tabletas corren un riesgo al momento de conectarse a una red WI-FI desconocida, donde automáticamente empieza a descargar o actualizar las apps instalados, dando la posibilidad de que se instale un software malicioso o malintencionado y pueda dañar o robar la información.

2.6.2 VULNERABILIDADES Y SEGURIDAD EN DISPOSITIVOS ANDROID

Meshram et al., (2014) comenta que según estudios de mercado de IDC, 1.200 millones de nuevos teléfonos inteligentes se utilizarían en 2014, de los cuales más de 950 millones de teléfonos inteligentes serían impulsados por el sistema operativo Android, lo que representaría casi el 79 por ciento de la cuota de mercado de teléfonos inteligentes en 2014, para convertirse en el líder invencible en la industria. iOS, que está en el segundo lugar, tendría una cuota de mercado del 14,9 por ciento con casi 180 millones de dispositivos inteligentes en uso.

Sistema Operativo	**Volúmenes de envío 2014**	**Cuota de mercado 2014**	**Volúmenes de envió 2018**	**Cuota de Mercado 2018**	**2014-2018 CAGR**
Android	950.5	78.9%	1,321.1	76.0%	10.7%
iOS	179.9	14.9%	249.6	14.4%	10.2%
Windows Phone	47.0	3.9%	121.8	7.0%	29.5%
BlackBerry	11.9	1.0%	5.3	O.3%	-22.6%
Otros	15.1	1.3%	40.7	2.3%	32.7%
Total	**1204.4**	**100.0%**	**1.738.5**	**100.0%**	**11.5%**

Tabla 8. Cuota de mercado mundial de Smartphone que se prevé de 2014 al 2018 (IDC, 2014).

En respuesta a las amenazas en los dispositivos móviles inteligentes, muchos proveedores de seguridad han estado ofreciendo productos de seguridad. El método más común de detectar aplicaciones malintencionadas utilizadas por los productos de seguridad es extraer la firma de una aplicación y luego compararla con las firmas de la aplicación malintencionada en la base de datos. Si son los mismos, la aplicación es maliciosa.

Este método puede detectar las aplicaciones maliciosas populares, mientras que podría hacer nada para el nuevo, impopular o aplicaciones de terceros. Esta es la razón por la detección de aplicaciones maliciosas no es alta. Otro retroceso del método de comparación de firmas es que cada vez que las aplicaciones maliciosas reemplazan sus firmas, la base de datos de firmas malintencionadas debe actualizarse en consecuencia.

Los proveedores de seguridad propusieron un nuevo modelo de seguridad, conocido como antivirus en la nube para dispositivos móviles. Su idea básica es extraer la firma de la aplicación en los dispositivos móviles, enviarla a la nube para detectar y devolver resultados a los dispositivos desde la nube.

En la detección normal, los dispositivos móviles tienen una gran cantidad de trabajo de procesamiento, que consume valiosos recursos de energía del sistema de los dispositivos móviles. Mientras que en el antivirus en la nube, tienen un gran número de tareas de procesamiento, por lo tanto, los dispositivos se liberan y se mejora la tasa de detección. Pero la tasa de detección de las últimas aplicaciones maliciosas todavía no es alta utilizando los productos antivirus en la nube existentes.

En el sistema Android, los métodos de detección de ataques desde el día cero se dividen principalmente en dos categorías, análisis dinámico y análisis estático. El análisis estático es analizar archivos .apk o archivos .dex del Sistema Android. William Enck propuso un método de análisis estático para analizar los permisos de una aplicación en Android. Primero analizan los permisos concedidos a la aplicación cuando se está ejecutando.

Se comparan los permisos declarados en el AndroidManifest.xml. Si son diferentes, la aplicación tal vez oculta permisos peligrosos. Así que la instalación está prohibida. Pero este método solo puede analizar si hay permisos ocultos en una aplicación. Si no hay permisos ocultos, el usuario determina si debe continuar instalando la aplicación por sí mismo.

A pesar de que una aplicación declara una serie de permisos extremadamente peligrosos (por ejemplo, CALL PHONE, SEND SMS), si la aplicación no oculta estos permisos, el método de detección mencionado no prohíbe la instalación de la Aplicación. Mientras la gran mayoría de los usuarios de Android no saben que estos permisos pueden causar graves consecuencias al continuar la instalación de estas aplicaciones maliciosas. Thomas Blasing propuso un método para detectar el comportamiento malicioso mediante el uso del emulador.

Utilizan un modo que puede generar flujos pseudoaleatorios de eventos de usuario como clics, toques o gestos, así como una serie de eventos a nivel de sistema. Se coloca un entorno de espacio limitado en el Kernel para registrar el comportamiento de la aplicación a nivel de sistema. Analizan el archivo de registro resultante para determinar si la aplicación es maliciosa. El método es para que los proveedores de seguridad detecten el comportamiento malintencionado de las nuevas aplicaciones. Pero no está adaptado a los dispositivos móviles ordinarios.

Antes de la introducción del sistema, tenemos que despejar los problemas de seguridad que enfrentan los dispositivos móviles, incluyendo los siguientes aspectos:

Aplicaciones maliciosas. Recientemente, el número de aplicaciones malintencionadas está creciendo considerablemente. Las aplicaciones maliciosas no sólo se encuentran en algún mercado de aplicaciones de terceros, también encontrado en Google Play Store de Android. Problemas muy graves son causados por estas aplicaciones maliciosas.

Algunas aplicaciones pueden revelar la ubicación del usuario, contactos y otra información personal; algunos pueden enviar mensajes cortos o hacer llamadas telefónicas en segundo plano con el objetivo de generar beneficio; algunas aplicaciones descargarán otras aplicaciones maliciosas de Internet.

Sitios web inseguros. Algunos sitios pueden contener un gran número de aplicaciones maliciosas. Además un usuario de un teléfono Android que utiliza el navegador web para navegar por Internet puede ser explotado si visita una página maliciosa y el atacante puede ejecutar cualquier código con los privilegios de la aplicación del navegador web.

Seguridad de datos de dispositivos móviles. Los dispositivos móviles inteligentes son diferentes de la PC debido a su portabilidad, y por lo tanto están en un mayor riesgo de pérdida. Cuando un usuario pierde su dispositivo, los datos de los dispositivos son difíciles de recuperar. Algunas otras razones como el mal uso de los usuarios, el daño de dispositivos también puede causar la pérdida de datos.

Seguridad de datos de red de los dispositivos móviles. Cuando un usuario se conecta a un punto de acceso configurado por el atacante, el tráfico de su dispositivo móvil puede ser reconocido muestra que, el atacante puede configurar un punto de acceso inalámbrico. Cuando un usuario utiliza este punto de acceso para navegar por Internet, la información personal y las credenciales del usuario (incluido nombre de usuario, contraseña, números de tarjeta de crédito, etc.) se revelarían al atacante. Incluso si el usuario utiliza una conexión SSL, el atacante tiene muchos métodos para obtener la información personal del usuario.

2.7 VULNERABILIDADES EN ANDROID RELACIONADOS CON LA NUBE

Los ataques de Android mencionan Wu et al., (2013) que son generalmente clasificados en tres tipos de amenazas: malware, grayware, y spyware personal. Los propósitos finales de estos ataques son la privacidad del usuario, el rastreo, denegación de servicio y la sobrefacturación.

DroidRanger está diseñado para detectar aplicaciones malintencionadas en los mercados populares de Android. Pero los métodos de detección existentes se centran en las aplicaciones de Android, que son adecuados para detectar vulnerabilidades en el mismo sistema Android.

Un nuevo paradigma de detección de vulnerabilidades es la nube Fuzzing para detectar vulnerabilidades en el sistema Android, donde se detectan todas las capas de arquitectura incluyendo el kernel de Linux, bibliotecas nativas, tiempo de ejecución de Android, marco de aplicación y se detectan las aplicaciones.

La nube Fuzzing proporciona un servicio de detección de vulnerabilidades para toda la versión del sistema Android mediante la agrupación en clústeres de un paquete de nodos fuzzing. Para probar un módulo, los casos de prueba en la máquina virtual correspondiente se ejecutan por un dispositivo Android real o un emulador de Android. Toda la ejecución del caso se supervisa en tiempo real, y si se bloquea, puede haber una vulnerabilidad potencial.

La Figura 27 muestra la arquitectura de la nube Fuzzing, donde los nodos fuzzing juegan como difusores para los módulos del sistema Android. La computación en la nube administra los recursos informáticos (por ejemplo, CPU, memoria, disco duro, red, etc.), configura el entorno, empaqueta el hardware y el software en máquinas virtuales separadas, es decir, difusores. La nube Fuzzing es una plataforma recursos bajo demanda, que puede satisfacer de forma elástica los requisitos de procesamiento y almacenamiento de las pruebas de pelusa.

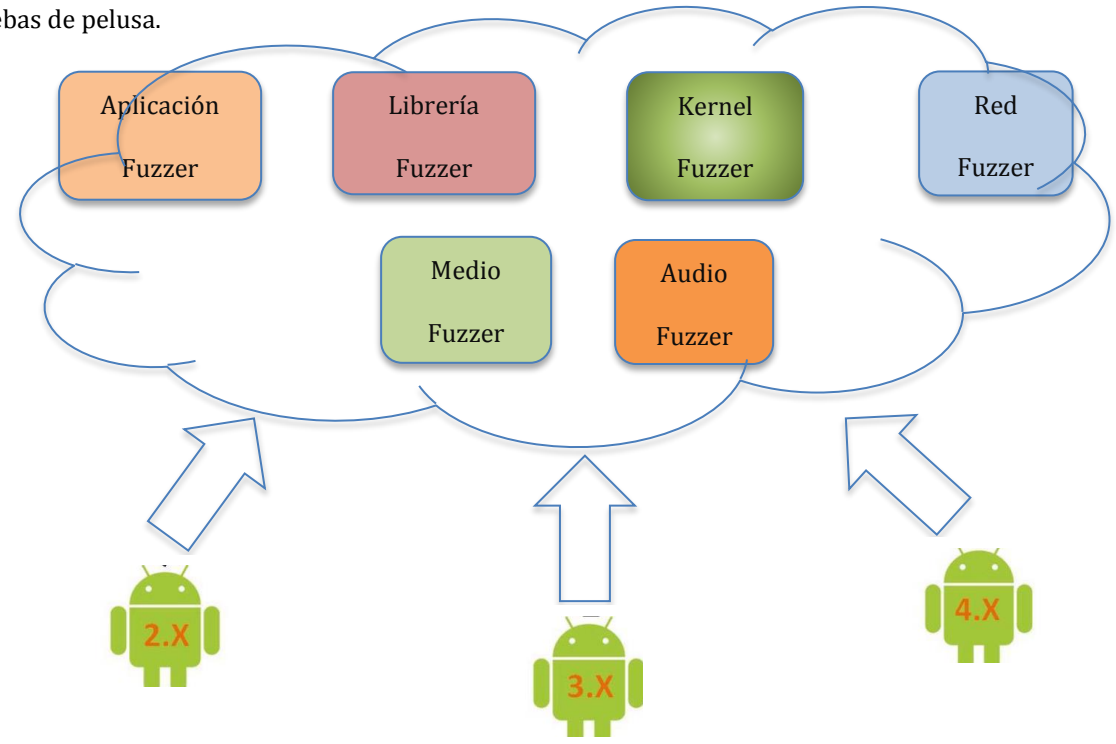


Figura 27. Visión general de alto nivel de la nube Fuzzing

2.7.1 LA DESCOMPOSICIÓN DE ANDROID

La pila de software de Android se puede subdividir en cuatro capas incluyendo el Kernel, bibliotecas nativas, tiempo de ejecución de Android, la capa de marco de trabajo y las aplicaciones. Para detectar vulnerabilidades en el sistema Android, cada capa debe descomponerse en módulos separados con más detalle y probarse con casos de prueba fuzzing en el fuzzer, como se muestra en la figura 27.

La capa básica es el Kernel de Linux, que es una capa de abstracción entre el hardware y la pila de software. La nube Fuzzing divide el Kernel en el módulo del sistema de archivos, módulo de memoria, módulo de audio, módulo de visualización, módulo WiFi, etc. y construye el fuzzer separado para cada módulo. Capa de bibliotecas nativas admite el administrador de superficies para controlar el acceso a la pantalla para el administrador de ventanas desde la capa de marco de trabajo, proporciona marcos para el servicio de audio y vídeo e implementa dispositivos que consultan a través de SQLite.

La nube fuzzing ofrece fuzzers para detectar vulnerabilidades en los componentes de webkit, FreeType, SSL, etc. el marco de trabajo de aplicaciones es la base del desarrollo de aplicaciones en Android. La nube fuzzing crea fuzzers para el gestor de actividades, el gestor de ventanas, los proveedores de contenido, el sistema de vista, el gestor de notificaciones, el gestor de paquetes, el gestor de telefonía, el gestor de recursos y el administrador de ubicación.

La nube fuzzing se refiere principalmente a la aplicación preinstalada como marcador, SMS/ MMS, mensajería instantánea, navegador, calendario, etc. en la capa de aplicaciones de Android, y compila fuzzers para todas estas aplicaciones.

El prototipo de nube de difusión se implementa en los servidores con plataforma de virtualización Xen. La Figura 28 muestra la implementación detallada del algoritmo 1 de la nube de difusión, donde CPU y los recursos de disco se asignan para cada difusor por módulos separados.

1: procedure FUZZINGCLOUD(*res, mods*)	----------*construir* fuzzers
2: *fc* ← *set*()	
3: for all *m* ∈ *mods* do	--------- inicializacion
4: *m.cpu* ← *res.cpu* _ CPU assignment	--------- asignacion de CPU
5: *m.disk* ← *res.disk* _ storage assignment	---------- asignación de almacenamiento
6: *m.fuzzerinit*() _ initialize fuzzer	--------- inicializacion del fuzzer
7: *fc.add*(*m*) _ add to cloud	---------- agregar a la nube
8: end for	
9: *Thread*(*adjust*(*fc*)).*start*()	---------- ajuste dinámico
10: return *fc*	---------- conjunto final de fuzzers
11: end procedure	

Figura 28. Algoritmo 1. Implementación de la nube de difusión

Manifestaron Zhang et al., (2015) que los métodos de Fuente de código orientado no pueden manejar los factores que son extraídos por el compilador, mientras que los métodos orientados al código binario procesan el software binario real, que puede ser ejecutados por la CPU directamente, los métodos orientados de código binario pueden lograr un resultado con más exactitud cuándo se aplica para detector vulnerabilidades en software Android. Hasta ahora los métodos orientados de código binario son más eficientes e incluyen el análisis de mancha.

La ejecución simbólica interpreta el programa mediante la asignación de datos simbólicos en lugar de datos reales a los insumos del programa. La ejecución simbólica tradicional tiene que enfrentarse a un problema "ruta de explosión", que impide que este método se pueda aplicar al software del mundo real.

Método basado en la ejecución simbólica. Vulnerabilidades de corrupción de memoria a menudo se originó a partir de la memoria de acceso ilegal a las localidades de memoria, como las vulnerabilidades de desbordamiento de pila, por lo que el límite del objeto de memoria es un factor crítico para detectar estas vulnerabilidades. Existe una variedad de objetos de memoria en el lenguaje de programación de alto nivel, como la matriz, clase y así sucesivamente, podríamos revisar el código fuente.

Apilar límite de seguridad de marco. Vulnerabilidad de desbordamiento de bufer basado en pila es un tipo popular de vulnerabilidades como hasta ahora. Pila es un bloque de memoria que realiza de forma automática el sistema operativo para almacenar variables, parámetros y direcciones de funciones para volver. Desbordamiento de búfer basado en pila se produce cuando el programa escribe más datos en una memoria intermedia situada en la pila de lo que realmente es asignado para ese búfer.

Pero sobrescribir la ubicación del puntero de marco y la dirección de retorno en la pila puede conducir el software en un estado desconocido o, lo que es más grave, permitir la ejecución de código malicioso arbitrario.

Límite de seguridad del montón del bloque. La memoria de pila de un proceso se utiliza para almacenar datos que son demasiado grandes de memoria de pila para manejar, memoria Heap es administrado por API especificas en la biblioteca Bionic C que es la plataforma de Android, como mailoc(), realloc(), y asi sucesivamente.

El gestor de memoria de pila en la biblioteca Bionic C es virtualmente una memoria asignada Dong Lea.

La adhesión ilegal de los límites de seguridad de la memoria que hemos definido anteriormente siempre se va a plantear con la vulnerabilidad de corrupción de memoria. Para detectar estas adhesiones de memoria ilegales, se han definido algunas reglas de seguridad de la siguiente manera, que describen como el programa debe acceder a la memoria legalmente y de forma segura.

Reglas de seguridad de la adhesión de memoria de pila. Como se mencionó anteriormente, muchos ataques de desbordamiento de búfer basado en pila explotable tratan de sobrescribir las direcciones de retorno de funciones y lugares sensibles de seguridad en la pila para controlar el programa. Así podemos definir reglas de seguridad de la memoria de pila de adhesión de la siguiente manera:

Regla1:

$$\forall\{write(addr, value) \wedge addr \in STACK \wedge value.taint = True\},$$
$$\{ \mid , . \wedge . \}\ i\ i\ i\ i \in I\ addr = Frame\ RET\ addr = Frame\ FP = \emptyset$$

Regla2:

$$\forall\{write(addr, value) \wedge read\ (addr, value)\},$$
$$\{addr \mid addr \in STACK \wedge addr < SP\} = \emptyset$$

Marco se refiere a que la pila de la cadena de invocación de la función bajo APCS, SP se refiere al puntero de la pila actual, se refiere a la colección de etiquetas que hemos mantenido para marcos de pila sombra.

Regla 1 significa una operación de escritura en la memoria cuyo valor está controlado por la salida, no puede sobrescribir el nivel de la pila donde almacena el puntero de marco o la dirección de retorno.

Regla 2 limita el funcionamiento de la memoria en pila que impide que el acceso a la zona sin inicializar en la memoria de pila.

Reglas de seguridad de la adhesión de memoria heap. Como se mencionó anteriormente, muchos ataques de desobordamiento de bufer basado en heap explotables intentan atacar el asignador de memoria a través de sobrescribir el trozo de la cabecera que puede dar la oportunidad de escribir datos arbitrarios a la dirección arbitraria.

La ejecución simbólica selectiva trata de explotar todas las rutas de ejecución en el espacio de ejecución del programa. En base a esta característica, se ha propuesto un nuevo método para detectar las vulnerabilidades de corrupción de memoria latentes en el software binario nativo de Android.

El método que aprovecha de ejecución simbólica selectivo para explotar todas las posibles rutas de ejecución con recursos finitos de tiempo y espacio, la integración de las normas de seguridad de adhesión que hemos definido para la pila de memoria, comprobamos todos los caminos que han sido desenterradas para determinar si existe una adhesión de memoria ilegal, es decir vulnerabilidades de corrupción de memoria.

S2E es una plataforma de análisis binario dinámica que utiliza ejecución simbólica selectiva para analizar pilas de software enteros en tiempo de ejecución. Hasta el momento, S2E está disponible para muchas arquitecturas del conjunto de instrumentos, tales como x86, ARM y así sucesivamente. El S2E reutiliza partes de la máquina virtual QEMU, el motor KLEE simbólico de ejecución y la cadena de herramientas LLVM. Tenemos medida S2E primordial para apoyar al sistema operativo Android. La Figura 29 muestra la arquitectura básica de S2EDroid.

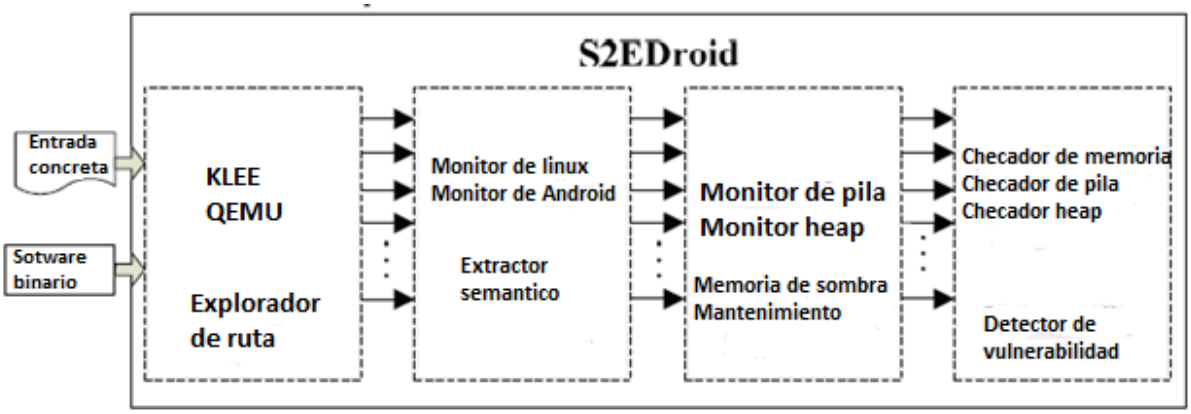


Figura 29. Arquitectura de S2EDroid

S2EDroid se puede dividir en 4 módulos, es decir Explore Path, extractor semántico, mantenedoras, memoria sombra y detector de vulnerabilidad Explorer Path se utiliza para desenterrar todas las posibles rutas de ejecución basado en la ejecución simbólica selectiva; el extractor semántico trata de extraer información semántica a nivel de sistema operativo, como el proceso actual y así sucesivamente, a través de la interpretación de Android y tabla de símbolos de Linux.

Hemos comparado nuestro S2EDroid con el famoso melocotón motor de prueba de pelusa, que es ampliamente utilizado para encontrar errores de software binario en diferentes plataformas. Elegimos la vulnerabilidad raíz GingerBreak para comparar el rendimiento entre S2EDroid y melocotón. La Tabla 9 muestra los resultados de nuestra comparación. De los resultados podemos ver claramente que nuestro método puede detectar la vulnerabilidad más eficiente que el marco de melocotón en el costo de un poco de sobrecarga de memoria.

Tabla 9. Comparación entre S2EDroid y el Melocotón

Método	Memoria	Hora	Detectado
S2EDroid	1.7 GB	25 minutos	Si
Melocotón	1.53 GB	1,7 horas	Si

Comparación. En particular, nuestra comparación abarca dos preguntas. ¿Puede S2EDroid detectar con precisión las vulnerabilidades de corrupción de memoria mediante el uso de nuestro algoritmo de detección de la vulnerabilidad de corrupción de memoria? ¿Qué cobertura de código se puede esperar de explorar la trayectoria de S2EDroid que utiliza la ejecución simbólica selectiva?

Para la pregunta 1, seleccionamos una vulnerabilidad raíz de Android, que tiene un identificador CVE: CVE-2011-18123 en CVE de base de datos de vulnerabilidad para evaluar la capacidad de detección de la vulnerabilidad de S2EDroid, firmado para comprobar en el método DirectVolume, handlePartitionAdded, lo que provoca daños en la memoria, esta vulnerabilidad es también llamado como GingerBreak. S2EDroid inyecta entrada simbólica en proceso interceptando el mensaje en el socket que envio al void.

Para la pregunta 2, seleccionamos un programa de demostración que tiene un número finito de ruta y conocidos para evaluar la cobertura de código de S2EDroid. El código de demostración se muestra a continuación:

```
void path_coverage(int length) {
    int x;
    char x_buf[10];
    s2e_enable_forking();
    s2e_make_symbolic(&x,sizeof(x), "sym");
    if(x > 0) {
    if(x > 50)
         s2e_kill_state(1, "if branch 'x >50+++'.xcanbe" +x_buf);
      else {
      if(x>20)
         s2e_kill_state(2, "if branch 'x >20'. x can be " +x_buf);
      else
         s2e_kill_state(3, "else branch 'x <=20' .x can be " +x_buf);
      }
    }else
       s2e_kill_state(4, "else branch 'x <=0'. x can be" +x_buf);
     s2e_disable_forking();
    }
```

A partir del resultado anterior, podemos ver que las horquillas S2EDroid Unidos durante los 4 caminos de ejecución, respectivamente. Por lo que podemos concluir de los resultados que logra la cobertura S2EDroid bloque básico de 100% y, además, proporciona el valor de muestra de la entrada. NIVAnalyzer: a Tool for Automatically Detecting and Verifying Next-Intent Vulnerabilities in Android Apps.

En lugar de definir las vulnerabilidades de una especificación, (que deben escribirse para cada especificación), elegimos definir patrones de vulnerabilidad para describir vulnerabilidades basadas en la intención de un tipo de componente Android. Un patrón de vulnerabilidad, denotado V, es una suspensión ioSTS especializada compuesta de dos ubicaciones finales distintas Vul, NVul que tienen como objetivo reconocer el estado de vulnerabilidad sobre las ejecuciones de componentes.

Intuitivamente, las ejecuciones de un patrón de vulnerabilidad, a partir de la ubicación inicial y terminada por Vul, expresan la presencia de la vulnerabilidad. Por deducción, las corridas terminadas por NVul express comportamientos funcionales que muestran la ausencia de la vulnerabilidad. Vul también es de salida completa para reconocer un estado lo que sea las acciones observadas durante las pruebas.

2.8 ANÁLISIS Y EXPLOTACIÓN DE VULNERABILIDADES

(González, 2017) Menciona que 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.

2.8.1 VULNERABILIDADES COMUNES EN SOFTWARE

A continuación se enlistan algunas de las vulnerabilidades más comunes encontradas en el sistema operativo.

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.

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.

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, si un atacante es capaz de escribir datos en dicha localidad, se puede llegar a ejecutar el código arbitrario plantado por el atacante.

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. 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.

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), 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 a sus usuarios.

2.8.2 Vulnerabilidades importantes en Android

Se establece 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 Figura 30, solo se mencionan una por año desde 2013, esto para establecer una visión general, más que un reporte detallado.

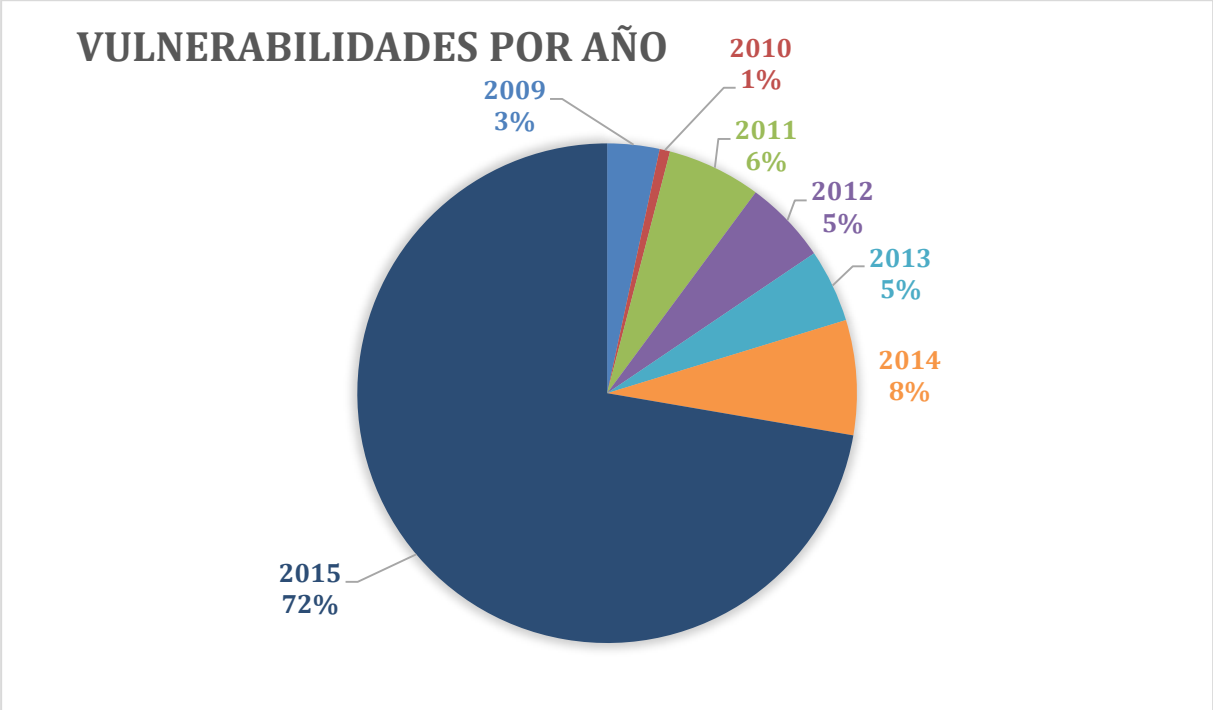


Figura 30. Vulnerabilidades por año.

2.8.3 Vulnerabilidades con mayor impacto

2.8.3.1 Apk Duplicate file (Julio de 2013)

Antes de que una aplicación sea descomprimida e instalada en un dispositivo, las firmas criptográficas de sus contenidos son verificadas, buscando que coincidan con las establecidas en un archivo de manifiesto incluido en la misma aplicación. 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 aplicación 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 aplicación, solo se verificaba la firma del primero de esos archivos y la última versión era la que se extraía y utilizaba para la app, 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, tal como se puede ver en la Figura 31:

Figura 31. Cadena de texto Bluebox

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.

2.8.3.2 THE FUTEX VULNERABILITY (JUNIO DE 2014)

Esta es una vulnerabilidad descubierta por Nicholas Allegra 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.

2.8.3.3 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 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, incluyendo Desbordamientos de entero (de los cuales ya hemos hablado) y Sobre-lectura de Búferf.

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**.

Los vectores de ataque 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.

2.9 Informe sobre el estado de seguridad 2019h1

El objetivo de este informe es sintetizar la información sobre ciberseguridad de los últimos meses (desde la seguridad en móviles hasta el ciberriesgo, desde las noticias más relevantes hasta las más técnicas y las vulnerabilidades más habituales), tomando un punto de vista que abarque la mayoría de los aspectos de esta disciplina, para así ayudar al lector a comprender los riesgos del panorama actual.

El lector dispondrá con este informe de una herramienta para comprender el estado de la seguridad desde diferentes perspectivas, y podrá conocer su estado actual y proyectar posibles tendencias a corto plazo.

La información recogida se basa en buena parte en la recopilación y síntesis de datos internos, contrastados con información pública de fuentes que consideramos de calidad. Vulnerabilidades en Android 2019-H1. A continuación se muestra en la Figura 32 la evolución de las vulnerabilidades por año en Android.

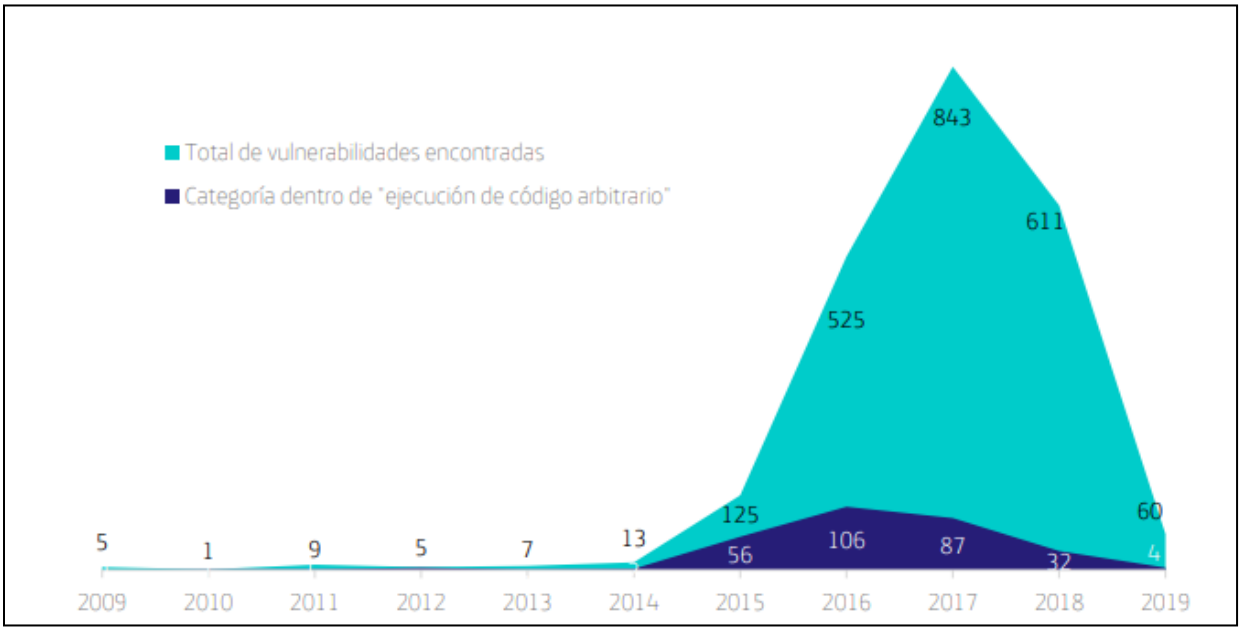


Figura 32. Evolución de las vulnerabilidades de Android por año.

Tiempo medio de retirada de aplicaciones maliciosas en Google Play Hemos vuelto a estudiar el tiempo medio de retirada de aplicaciones maliciosas en Google Play. Lo primero que nos llama la atención es la gran cantidad de aplicaciones retiradas (casi tres veces más) que el periodo anterior.

Esto puede representar que, o bien están realizando una "limpieza" más profunda del mercado, o bien que todavía sigue siendo relativamente fácil colar aplicaciones de baja calidad en él.

Hemos analizado el tiempo que tarda Google Play (el Market oficial de aplicaciones para Android) en retirar aplicaciones maliciosas o como es denominado por Google: PUA, Potentially Unwanted Application. Esto es, desde que la aplicación es subida por su autor o autores hasta que finalmente es retirada.

No solo se tiene en cuenta que la aplicación haya sido retirada, dado que esta operación suele ser común y no siempre por ser esta calificada como maliciosa, sino que la aplicación debe ser detectada como malware por al menos un motor antivirus.

El periodo de estudio comprende desde el 1 de enero de 2019 hasta el 30 de junio de 2019. El grueso de aplicaciones seleccionada es de 44.782 aplicaciones retiradas en algún momento del periodo mencionado.

De ellas, hemos analizado un subconjunto representativo de más de 5000 aplicaciones, en que hemos calificado 115 aplicaciones como maliciosas. Podemos establecer como aproximadamente un 2% de las aplicaciones retiradas por considerarse malware. Finalmente, los datos comparativos de tiempo (medido en días) de retirada de la tienda oficial de Android con respecto al semestre anterior arrojan los resultados de la gráfica. Esto quiere decir que Google Play retira las aplicaciones consideradas malware en una media de algo más de 51 días, aunque algunas han podido llegar a mantenerse en él hasta 138. Además, estos datos no suponen una variación relevante con respecto al segundo semestre de 2018.

2.9.1 Plazo de retirada de Play Store

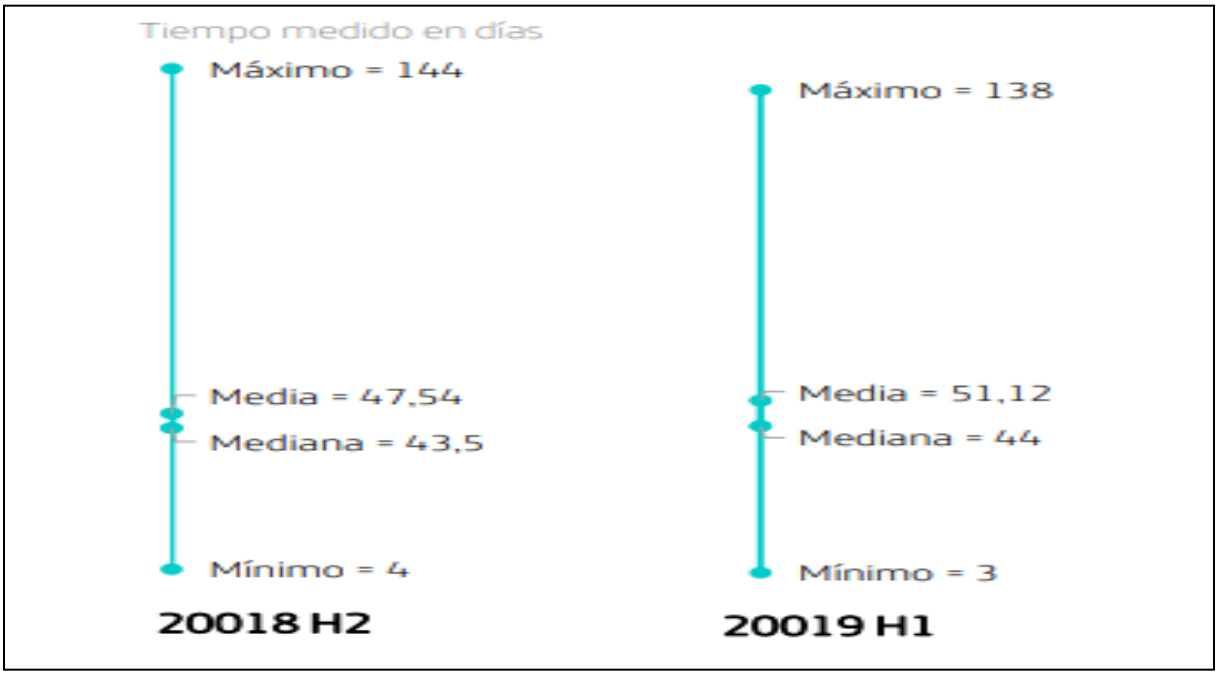


Figura 33. Plazo de retirada medido en días de las aplicaciones en Play Store

En la Figura 33 se muestra el plazo de tiempo en días de las aplicaciones en la tienda oficial de Android.

2.9.2 VULNERABILIDADES DESTACADAS

Tabla 10. Vulnerabilidades más destacadas (CVE Details, 2020).

CVE ID	OBJETIVO	DESCRIPCIÓN	PUNTUACIÓN
CVE-2019-12735	Editor de textos Vim y NeoVim	Vim y NeoVim utiliza cierto tipo de metadatos en archivos de texto plano para configurar ciertas propiedades del editor. Existe un fallo que permite a ciertas opciones de este mecanismo (denominado 'modeline') evadir la sandbox y ejecutar comandos del sistema de forma arbitraria. Como puede derivarse de la descripción, tan solo es necesario agregar una línea comentada a un archivo de texto plano con contenido malicioso y esperar a que un usuario de este editor de texto abra el archivo para que dicho contenido se ejecute en el sistema.	**9.3**
CVE-2019-11683	Kernel Linux	Un error en el proceso de paquetes UDP podría causar un estado de denegación de servicio en el sistema a través de paquetes UDP especialmente manipulados. Solo afecta a la rama 5 del kernel Linux.	**10.0**
CVE-2019-11477	Kernel Linux	Ingenieros de Netflix han encontrado un fallo en la implementación de SACK del kernel Linux. SACK permite seleccionar de forma unitaria para su retrasmisión aquellos paquetes TCP que no han sido recibidos. Esta función ahorra tráfico, dado que sin SACK, si se reclama un paquete no recibido, la otra parte enviará el paquete no recibido y todos los paquetes posteriores; incluso si estos si fueron recibidos por el cliente. Se da la particular circunstancia de que el fallo ha permanecido 10 años sin ser descubierto, o al menos, sin noticia de su explotación durante este periodo de tiempo tan prolongado	**N/D**

CVE-2018-12130 (ZombieLoad)	Múltiples CPU de Intel	Nuevo fallo de seguridad que afecta a las CPU de Intel Core y Xeon que permitiría el acceso a datos sensibles de otros procesos. Este tipo de fallos explotan la ejecución especulativa de los procesadores, es decir, la anticipación en la ejecución de instrucciones que podrían ser o no descartadas. Una forma de optimización que permite "adelantar" trabajo de CPU antes de que el flujo de ejecución alcance ese punto.	**6.5**

2.9.3 TOP 10 CWE MÁS REPRESENTATIVOS

CWE (Common Weakness Enumeration) es una clasificación que agrupa todas las debilidades identificadas en productos informáticos. Similar al esfuerzo realizado con CVE para etiquetar las vulnerabilidades concretas, halladas por producto, CWE se centra en definir los tipos de forma abstracta. Esto permite realizar un mapeo directo entre CVE y CWE. Esta lista comprende a los 10 CWE que más se han asignado por número de CVE. Esto nos permite observar qué tipo o clase de debilidades han sido más frecuentes en este periodo de estudio.

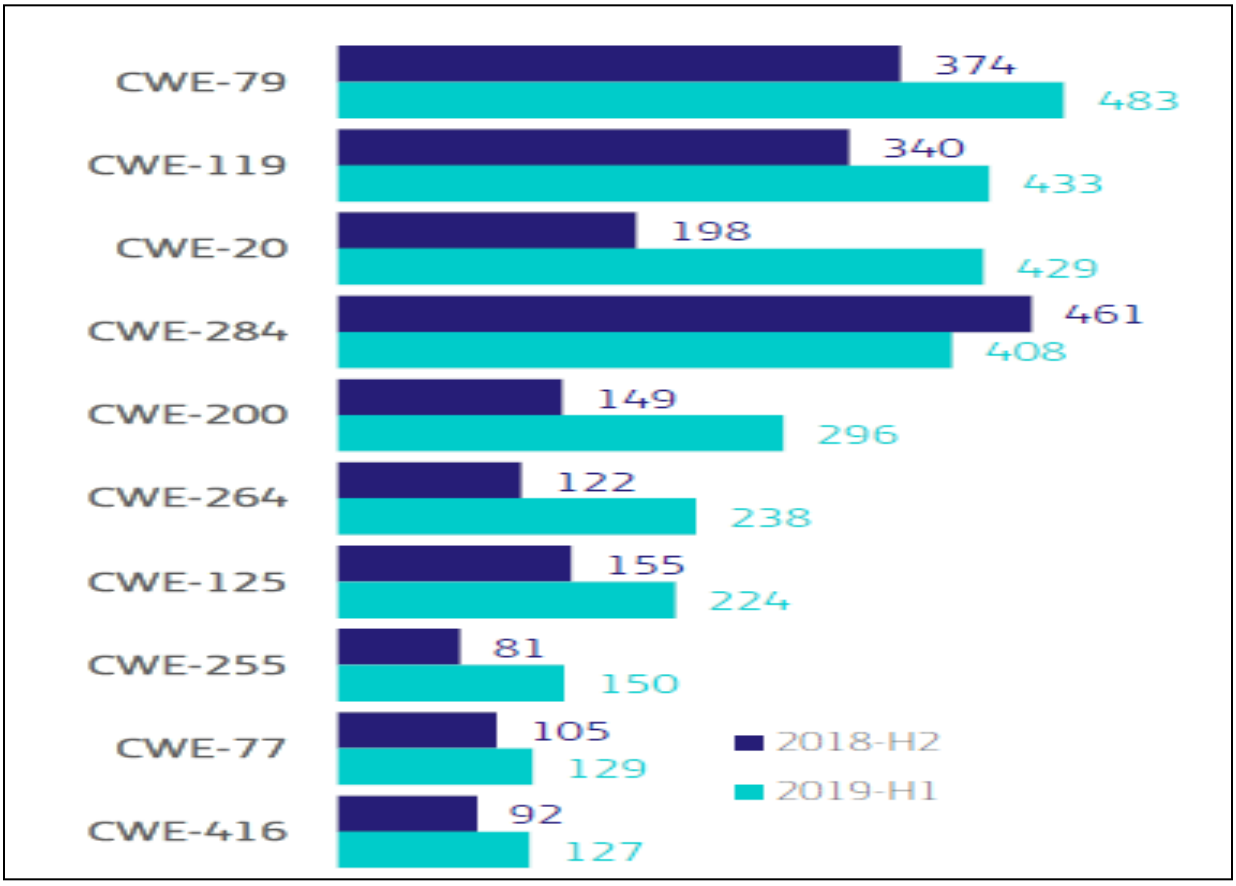


Figura 34. Top 10 CWE más representativos (CWE, 2020)

La Figura 34 nos muestra las vulnerabilidades más representativas durante 2018 y 2019 según CWE.

En la Tabla 11, se muestra una descripción más a detalle del Top 10 CWE de vulnerabilidades más representativas durante el año 2019.

Tabla 11. Descripción de las vulnerabilidades.

CWE	TÍTULO	DESCRIPCIÓN	CANTIDAD
CWE-79	Improper Neutralization of Input During Web Page Generation	Básicamente, recoge los tres tipos conocidos de vectores para realizar un Cross-site scripting: Reflejado, almacenado y basado en DOM	483
CWE-119	Improper Restriction of Operations within the Bounds of a Memory Buffer	De forma general, recoge aquellos errores de programación donde no se está controlando la capacidad de un buffer de memoria, tanto en operaciones de escritura como de lectura	433
CWE-20	Improper Input Validation	Categoría general para errores que consisten en un control deficiente o inexistente en entradas de datos procedentes de usuario.	429
CWE-284	Improper Access control	La aplicación no restringe el acceso a los recursos de forma adecuada. Se trata de un capítulo genérico donde se recoge aquellos defectos relacionados con la falta de prohibición o control adecuado cuando un tercero accede a recursos para los cuales no posee los permisos adecuados.	408
CWE-200	Information Exposure	Recoge, de forma general, el compromiso de información sensible debido a la ausencia o deficiencia de controles que impidan la fuga de información.	296
CWE-264	Permissions, Privileges and Access Controls	Se trata de una categoría general donde entra toda deficiencia relacionada con los permisos atribuidos a los usuarios o procesos, los privilegios que se les atribuye y el control de acceso a los recursos (relacionada, en este sentido, con CWE-284).	238
CWE-125	Out-of-bounds Read	Muy relacionada con CWE-119, recoge operaciones de lectura a memoria rebasando los límites de control de un búfer en concreto	224

CWE-255	Credentials Managenements	Comprende todas aquellas vulnerabilidades que afectan a la forma en la que se gestionan las credenciales de usuarios. Por ejemplo, su almacenamiento, transporte y creación.	150
CWE-77	Improper Neutralization of Special Elements user in a Command	Se refiere a la ausencia o deficiencia de filtrado de elementos en cadenas que podrían permitir la ejecución arbitraria de comandos en el sistema	129
CWE-416	Use After Free	Fallo en la gestión de memoria de un proceso que permite llamar a un objeto o referenciar una zona de memoria en el montículo que ha sido liberada previamente.	127

CAPÍTULO III. DISCUSIÓN

En este capítulo se discuten las vulnerabilidades principales del sistema operativo Android encontradas en la literatura y se describen recomendaciones generales para los usuarios de este sistema operativo.

3.1 CATALOGO DE VULNERABILIDADES DE ANDROID EN SEGURIDAD

(Özkan, 2019a) clasifica las vulnerabilidades utilizando principalmente la coincidencia de palabras clave y secundariamente con los números de identificador de vulnerabilidad (CVE).

A menos que se indique lo contrario, los puntajes **CVSS** que publica Ozkan se incluyen en las fuentes de la base de datos NVD, la cual es actualizada diariamente, para más detalles consultar **nvd.nist.gov**.

Por otro lado, (Özkan , 2019b) proporciona una interfaz web única para las definiciones de lenguaje de evaluación y vulnerabilidad abierta (OVAL). En la Tabla 1 podemos ver las vulnerabilidades del año 2009 a la fecha (Özkan, 2019ª):

A continuación se enlistan algunas de las vulnerabilidades más representativas con el impacto correspondiente:

CVE-2009-2348 Vulnerabilidad correspondiente a Bypass (Omitir una restricción o similar):

Impacto de confidencialidad seria que existe una divulgación total de información, lo que resulta en la revelación de todos los archivos del sistema.

Impacto de integridad Existe un compromiso total de la integridad del sistema. Hay una pérdida completa de la protección del sistema, lo que da como resultado que todo el sistema se vea comprometido.

Impacto de disponibilidad hay un cierre total del recurso afectado. El atacante puede dejar el recurso completamente disponible.

CVE -2010-1807 Vulnerabilidad de código de ejecución de denegación de servicio (DoS):

Impacto de confidencialidad existe una divulgación total de información, lo que resulta en la revelación de todos los archivos del sistema

Impacto de integridad existe un compromiso total de la integridad del sistema. Hay una pérdida completa de la protección del sistema, lo que da como resultado que todo el sistema se vea comprometido.

Impacto de disponibilidad hay un cierre total del recurso afectado. El atacante puede dejar el recurso completamente disponible.

CVE-2011-2344 Vulnerabilidad de ganar privilegios

Impacto de confidencialidad existe una divulgación total de información, lo que resulta en la revelación de todos los archivos del sistema.

Impacto de integridad existe un compromiso total de la integridad del sistema. Hay una pérdida completa de la protección del sistema, lo que da como resultado que todo el sistema se vea comprometido.

Impacto de disponibilidad hay un cierre total del recurso afectado. El atacante puede dejar el recurso completamente disponible.

Vulnerabilidad de ejecución de desbordamiento de código

Impacto de confidencialidad existe una divulgación total de información, lo que resulta en la revelación de todos los archivos del sistema.

Impacto de integridad existe un compromiso total de la integridad del sistema. Hay una pérdida completa de la protección del sistema, lo que da como resultado que todo el sistema se vea comprometido

Impacto de disponibilidad hay un cierre total del recurso afectado. El atacante puede dejar el recurso completamente disponible

Vulnerabilidad Privilegios de ganancia por desbordamiento Corrupción de memoria

Impacto de confidencialidad divulgación total de información, lo que resulta en la revelación de todos los archivos del sistema.

Impacto de integridad existe un compromiso total de la integridad del sistema. Hay una pérdida completa de la protección del sistema, lo que da como resultado que todo el sistema se vea comprometido.

Impacto de disponibilidad hay un cierre total del recurso afectado. El atacante puede dejar el recurso completamente disponible).

Vulnerabilidad ejecutar scripts de sitios cruzados de código (XSS)

Impacto de confidencialidad una divulgación total de información, lo que resulta en la revelación de todos los archivos del sistema.

Impacto de integridad existe un compromiso total de la integridad del sistema. Hay una pérdida completa de la protección del sistema, lo que da como resultado que todo el sistema se vea comprometido.

Impacto de disponibilidad hay un cierre total del recurso afectado. El atacante puede dejar el recurso completamente disponible.

La seguridad en un dispositivo y para un usuario son pieza fundamental así que deben tener cuidado con los riesgos a los que de igual forma se encuentra expuesto como el robo y fuga de información, Phishing, la ingeniería social y el Ransomware, las cuales como tal no son vulnerabilidades propias de Android pero son riesgos latentes que el usuario puede experimentar en algún momento y que sin duda son problemas que afectan de manera considerable al usuario.

3.1.1 Plataforma CVE Details

Las vulnerabilidades que resaltaron y que impactaron con más fuerza al sistema operativo Android del año 2009-2019 según CVE Details fueron:

a) Ejecución de código

La vulnerabilidad con identificador CVE-2019-2095 362 con fecha de publicación 2019-06-07 y fecha de actualización 2019-06-11, la cual se explota de manera remota afectando la integridad, disponibilidad y configuración completamente, categorizándola con una complejidad alta y un score de

7.6 a pesar del score se considera una vulnerabilidad complicada. En callGenIDChangeListeners y funciones relacionadas de SkPixelRef.cpp, hay un posible uso después de libre debido a una condición de carrera. Esto podría llevar a la ejecución remota de código sin necesidad de privilegios de ejecución adicionales. La interacción del usuario es necesaria para su explotación. La versión afectada de Android es la 9.

La vulnerabilidad con identificador CVE-2019-2126 415 con fecha de publicación 2019-08-20 y fecha de actualización 2019-08-22 teniendo un grado de complejidad medio afectando completamente la integridad, disponibilidad y configuración teniendo un score de 9.3. En ParseContentEncodingEntry de mkvparser.cc, esta vulnerabilidad podría llevar a cabo la ejecución remota de código sin necesidad de privilegios de ejecución adicionales. La interacción del usuario es necesaria para su explotación. Las versiones afectadas de Android son: 7.0, 7.1.1, 7.1.2, 8.0, 8.1 y 9.

CVE-2019-2044 787 teniendo fecha de publicación en 2019-05-08 y fecha de actualización 2019-05-09 con una complejidad media de la vulnerabilidad, con un score de 9.3 explotando de manera remota la disponibilidad e integridad por completo. En el paquete MakeMp>G4VideoCodecSpecificData hay una posible escritura fuera de los límites debido a una comprobación incorrecta de los límites. Esto podría llevar a la ejecución remota de código en el servidor de medios sin necesidad de privilegios de ejecución adicionales. La interacción del usuario es necesaria para su explotación. Las versiones afectadas de Android son: 7.0, 7.1.1, 7.1.2, 8.0, 8.1 y 9.

CVE-2018-9446 787 publicada en 2018-11-06 y actualizada en 2018-12-12, con un score de 10.0, explota la integridad y disponibilidad referente a la corrupción de memoria. Esta vulnerabilidad tiene una posible escritura fuera de los límites debido a daños en la memoria. Esto podría llevar a la ejecución remota de código sin necesidad de privilegios de ejecución adicionales. No es necesaria la interacción del usuario para su explotación. Las versiones afectadas de Android son: 6.0, 6.0.1, 7.0, 7.1.1, 7.1.2, 8.0 y 8.1.

CVE-2018-9572 publicada en 2018-12-07 con una complejidad media la cual se le dio un score de 9.3 completando y corrompiendo algunos rubros importantes del sistema como disponibilidad, integridad y configuración del mismo. Esta vulnerabilidad principalmente se le puede ver un fallo de una posible escritura fuera de los límites debido a la falta de la comprobación de límites. Esto podría llevar a la ejecución remota de código sin necesidad de privilegios de ejecución adicionales. La interacción del usuario es necesaria para su explotación. La versión afectada de Android es la 9.

b) Desbordamiento de búfer

La vulnerabilidad con CVE-2019-2046 190 la cual se considera complicada debido a que se le da un score de 10.0, siendo publicada en 2019-05-08 ejecutándose de manera remota corrompiendo la integridad y disponibilidad del sistema por completo. El sistema se ve afectado y es posible que la memoria esté dañada debido a un desbordamiento de enteros. Esto podría llevar a la ejecución remota de código en la configuración automática del proxy sin necesidad de privilegios de ejecución adicionales. No es necesaria la interacción del usuario para su explotación. Las versiones de Android afectadas son: 7.0, 7.1.1, 7.1.2, 8.0, 8.1 y 9.

CVE-2019-2007 190 la cual tiene un score de 10.0 siendo una vulnerabilidad de cuidado, de igual forma se explota de manera remota corrompiendo el sistema, fue publicada en 2019-06-19. Una posible escritura fuera de los límites debido a un desbordamiento de enteros. Esto podría llevar a una escala local de privilegios en el servidor de audio sin necesidad de privilegios de ejecución adicionales.

La interacción con el usuario no es necesaria para su explotación, viéndose afectadas las versiones de Android siguientes: 8.1 y 9.

CVE-2018-6271 119 vulnerabilidad que afecto a la NVIDIA Tegra OpenMax teniendo una complejidad media con un score de 9.3 afectando la disponibilidad, configuración e integridad del sistema, publicada en 2019-02-13.

CVE-2018-11905 119 afectando al sistema con un score de 10.0 siendo critica la vulnerabilidad, completando y afectando el rubro de disponibilidad e integridad del mismo. En todas las versiones Android (Android for MSM, Firefox OS for MSM, QRD Android) de CAF utilizando el núcleo de Linux, posible desbordamiento de búfer en la función WLAN debido a la falta de validación de entrada en los valores recibidos del firmware.

c) DoS

CVE-2018-6271 119 con un score de 9.3, de manera remota siendo publicada en 2019-02-13 con una complejidad media para exponer el sistema, la cual tiene un error en el controlador NVIDIA Tegra OpenMax entrega datos adicionales con el búfer y no valida correctamente los datos adicionales lo que puede provocar la denegación del servicio.

CVE-2018-6268 416 vulnerabilidad publicada en 2019-02-13 y fecha de actualización en 2019-04-02 en estado crítico con un score de 9.3 accediendo de manera remota y explotando la integridad, disponibilidad y configuración de manera completa, esta vulnerabilidad presenta errores en el controlador NVIDIA Tegra el cual puede escalar los privilegios o denegar el servicio.

CVE-2018-6267 20 Vulnerabilidad referente al controlador NVIDIA Tegra OpenMax, la cual fue publicada en 2019-02-13 y con fecha de actualización en 2019-10-02, siendo una vulnerabilidad critica con un score de 9.3 siendo una complejidad media a la hora de la explotación de la misma.

CVE-2016-7990 190 Vulnerabilidad que principalmente afecto a los dispositivos Samsung Galaxy del S4 al S7, existe una condición de desbordamiento de enteros en lobomacp.so al analizar mensajes SMS y WAP los cuales provocan un daño en la pila que puede dar lugar a la denegación del servicio y una ejecución de código de manera remota afectando de manera crítica al sistema con un score de 10.0, fue publicada en el año 2016-10-31.

CVE-2016-2464 20 Esta vulnerabilidad permite a los usuarios la ejecución de código arbitrario que provocan la denegación del servicio (corrupción de memoria) a través de un archivo mkv, fue publicada en 2016-06-12 teniendo un score de 9.3.

d) Ganar privilegios

CVE-2016-6706 264 La vulnerabilidad cuenta con un score de 9.3 lo cual la hace critica, dicha vulnerabilidad afecta en el servicio del sistema de versión Android 6.x ejecuta código arbitrario en el contexto de un proceso de privilegios, este problema se considera alto porque podría usarse para obtener el acceso local a capacidades elevadas (usuario root), publicada en 2016-11-25.

CVE-2016-6706 264 Esta vulnerabilidad se centra en la versión 7.0 de Android lo cual podría permitir que una aplicación maliciosa local ejecute código arbitrario en el contexto de obtener los privilegios necesarios para corromper todo el sistema operativo, viéndose afectada, la integridad, disponibilidad y confidencialidad del mismo, publicada en 2016-11-25 con 9.3 de puntuación siendo critica.

CVE-2016-6673 264 La vulnerabilidad que afecta directamente al controlador de cámara NVIDIA, el cual permite a los atacantes obtener privilegios a través de una aplicación diseñada, fue publicada en 2016-10-10 con un score de 9.3.

CVE-2018-9577 787 Esto podría llevar a la ejecución remota de código sin necesidad de privilegios de ejecución adicionales. La interacción del usuario es necesaria para su explotación. Se vio afectada la versión 9.0 de Android, fue publicada en 2018-12-07 con una puntuación es 9.3.

e) Ganar información

CVE-2019-9461 200 La vulnerabilidad afecta principalmente el Kernel de Android mediante el enrutamiento VPN donde hay una posible revelación de información, no es necesaria la interacción del usuario para su explotación, fue publicada en 2019-09-06 con un score de 7.8.

CVE-2018-9358 200 Esta vulnerabilidad compromete una posible lectura de datos no inicializados debido a una comprobación de límites que falta, Esto podría llevar a la revelación de información remota en el proceso Bluetooth sin necesidad de privilegios de ejecución adicionales afectando al sistema, con una puntuación de 7.8.

CVE-2017-13205 200 Esta vulnerabilidad de divulgación de información en el marco de medios de Android (libmpeg2) afectando a las versiones (7,0, 7.1.1, 7.1.2, 8.0 y 8.1) teniendo un score de 8.5 con fecha de publicación en 2018-01-12.

f) Bypass

CVE-2019-2018 264 Fue publicada en 2019-06-19, viéndose comprometido el servicio DevicePolicyManagerService.java, existe una posible omisión de la protección de restablecimiento de contraseña debido a una causa inusual de **root** siendo crítica con una puntuación de 9.3, afectando a las versiones 8.1 y 9 de Android.

CVE-2017-13176 20 Hay una validación de entrada incorrecta del campo host. Esto podría conducir a una elevación remota de privilegios que podría permitir omitir los requisitos de interacción del usuario sin necesidad de privilegios de ejecución adicionales. La interacción del usuario es necesaria para su explotación. Las Versiones de Android comprometidas son: 5.1.1, 6.0, 6.0.1, 7.0, 7.1.1, 7.1.2, 8.0, 8.1, publicada en 2018-01-12 con un score de 9.3 siendo crítica.

CVE-2017-0593 732 Fue publicada en 2017-05-12 Una vulnerabilidad de elevación de privilegios en el Framework de las API's podría permitir que una aplicación maliciosa local obtenga acceso a permisos personalizados. Este problema se considera alto porque es un bypass general para las protecciones del sistema operativo que aíslan los datos de aplicaciones de otras aplicaciones. Las Versiones de Android comprometidas son: (6.0, 6.0.1, 7.0, 7.1.1, 7.1.2) con un score de 9.3 siendo critica.

g) Corrupción de memoria

CVE-2019-2097 704 Esta vulnerabilidad podría llevar a la ejecución remota de código desde una configuración proxy maliciosa, sin necesidad de privilegios de ejecución adicionales. Las versiones de Android que se ven afectadas son: Android 7.0, Android 7.1.1, Android 7.1.2, Android 8.0, Android 8.1 y Android 9, dicha vulnerabilidad fue publicada en 2019-06-07, siendo una vulnerabilidad critica con un score de 10.0.

CVE-2019-2006 416 La vulnerabilidad podría llevar a una escala local de privilegios en el servidor de audio sin necesidad de privilegios de ejecución adicionales, fue publicada en 2019-06-19, afectando al sistema de manera crítica con un score de 10, en la versión 9 de Android.

CVE-2018-9446 787 Podría verse afectada una posible escritura fuera de los límites debido a daños en la memoria. Esto podría llevar a la ejecución remota de código sin necesidad de privilegios de ejecución adicionales, corrompiendo el sistema, dicha vulnerabilidad se pone como critica con un score de 10.0 siendo publicada en 2018-11-06.

CVE-2017-9678 119 Esta vulnerabilidad afecta a todos los productos Qualcomm con versiones de Android de CAF utilizando el Kernel de Linux, en un controlador de vídeo, la corrupción de memoria puede ocurrir potencialmente debido a la falta de límites que comprueban un memcpy(), publicada en 2017-08-18 siendo critica con 9.3 de score y afectando al sistema operativo Android de manera potencial.

3.2 SOFTWARE ANALIZADOR DE VULNERABILIDADES

Tang et al., (2017) El trabajo se ha centrado en el estudio de los problemas de seguridad traído por los componentes que se supone que son privados, pero se han establecido erróneamente como públicos por los desarrolladores de aplicaciones, suponiendo que el marco de Android puede proteger los componentes de privada de ser invocado por otras aplicaciones.

La vulnerabilidad de la siguiente intención, la cual es usada para secuestrar cualquier aplicación instalada ya que es difícil detectar el ataque, la cual puede ser usada para pedir cualquier permiso en donde no es necesario permisos a nivel de "**root"** ya que afecta casi a todas las versiones de Android de manera que se pueden escalar privilegios de manera significativa haciéndose pasar por una aplicación legitima, no requieren ningún permiso especial para poder ejecutarse y explotar dicha vulnerabilidad.

La vulnerabilidad de la siguiente intención en las aplicaciones Android es un problema grave. Al aprovechar esta vulnerabilidad, los atacantes pueden iniciar directamente componentes privados desde otras aplicaciones. Aunque se han descubierto algunas vulnerabilidades de próxima intención con el análisis manual, no está claro cuán severa y prevaleciente es en las aplicaciones del mundo real.

Una aplicación maliciosa puede ofrecer una intención diseñada para iniciar un componente público de la aplicación vulnerable, en el que se guarda una intención de destino bajo una clave especial. A continuación, la intención de destino se recupera y se pasa a los métodos de comunicación entre componentes (ICC), como startActivity. El resultado es que el atacante puede invocar el componente privado designado de la aplicación vulnerable.

Comúnmente una aplicación puede solicitar permiso para acceder a datos del dispositivo, como los contactos de un usuario, los mensajes de texto, el micrófono, el dispositivo de almacenamiento (tarjeta SD o almacenamiento local), la cámara y la conexión Bluetooth.

Si son invocados los componentes privados desde alguna otra aplicación, se pueden ver afectados los datos que se resguardan en la aplicación, así como se pueden acceder a servicios del sistema. Viéndose afectados los componentes privados debido al mal desarrollo en la aplicación se puede explotar la vulnerabilidad de la siguiente intención sin necesidad de pedir los permisos necesarios al usuario para poder acceder a los diferentes servicios del sistema, basta con invocarlos y poder interactuar en ella de la manera que se desee, la información que guarda la aplicación, así como los datos personales se ven totalmente comprometidos a

tal grado de que se puede hacer un mal uso de ellos, pegándole principalmente a la confidencialidad y disponibilidad de los datos.

Por lo tanto, el software NIVAnalyzer, detecta y verifica automáticamente las vulnerabilidades de la siguiente intención.

3.3 ANÁLISIS Y EXPLOTACIÓN DE VULNERABILIDADES

(González, 2017) Enlista algunas de las vulnerabilidades más comunes encontradas en el sistema operativo Android así como el impacto que generan:

Desbordamiento de búfer

Es un problema de seguridad de la memoria en donde el software o programa no considera o no verifica sus límites de almacenamiento.

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.

El impacto de explotar la vulnerabilidad de desbordamiento de búfer es ejecución remota de código corrompiendo la integridad y disponibilidad del sistema por completo. El sistema se ve afectado y es posible que la memoria esté dañada debido a un desbordamiento de enteros. No es necesaria la interacción del usuario para su explotación. En la mayoría de los casos se logra tomar el control del dispositivo en el cual el atacante puede ejecutar si así lo desea una denegación de servicios (DoS) para que el usuario legítimo no pueda hacer uso de los servicios.

Un software o programa con un diseño correcto debería estipular un tamaño máximo para los datos de entrada y garantizar que no superen ese valor para evitar que se explote la vulnerabilidad.

El desbordamiento de búfer ha afectado históricamente al sistema operativo Android, del año 2009 al 2019 un total de 503 vulnerabilidades documentadas, una cifra que ha marcado esta vulnerabilidad como una de las más severas para el sistema formando un 19.6% en total, a pesar del mejor desarrollo en el sistema para mejorar la seguridad prevalece en el mismo (CVE Details, 2020).

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.

El impacto de esta vulnerabilidad puede tener una complejidad de acceso baja, la autenticación no es necesaria para explotarla, no hay del todo impacto en la integridad, afecta de manera parcial a la confidencialidad del sistema esto pudiera conllevar a la divulgación de información local con los privilegios de ejecución necesarios, no es necesaria la interacción del usuario para su explotación, de igual forma no hay de todo impacto en la disponibilidad del sistema.

No sólo hay que tener cuidado con el desbordamiento de la pila o de los buffers. El desbordamiento de enteros es potencialmente igual de peligroso ya que se involucra una operación con la memoria esto puede hacer que se desborde el búfer lo cual implicaría control del dispositivo así como interrupciones de los servicios.

Incluso se puede ver afectado el sistema operativo por corrupción de memoria, vulnerabilidad que registra150 vulnerabilidades documentadas de 2009 al 2019 la cual ha impactado a Android en un 5.9%, siendo baja pero de cuidado si se explota correctamente (CVE Details, 2020).

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, si un atacante es capaz de escribir datos en dicha localidad, se puede llegar a ejecutar el código arbitrario plantado por el atacante.

El impacto de la vulnerabilidad seria la ejecución remota de código malicioso, el cual comprometería el sistema operativo por completo afectando diversos módulos del mismo, se vería vulnerable la integridad y disponibilidad del mismo.

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. 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.

Se puede ver comprometido el sistema teniendo control del mismo, pudiendo hacer cambios en configuración del sistema, corrompiendo la confidencialidad e integridad y divulgación de información sensible.

Se puede interceptar y ganar información del usuario, esta vulnerabilidad de ganar información ha registrado 313 del año 2009 al 2019 formando un 12.2% siendo una vulnerabilidad que ha persistido en el tiempo debido a que el usuario siempre está en riesgo de ser una víctima potencial (CVE Details, 2020).

3.3.1 VULNERABILIDADES HISTÓRICAS DE ANDROID

APK Duplicate file (Julio de 2013)

La aplicación se analiza para saber si revisando que las firmas criptográficas de sus contenidos correspondan por un archivo manifiesto en caso de que no coincidan la instalación de la aplicación termina, ya que en la tienda oficial existen diversas aplicaciones enmascaradas como fiables cuando en realidad tienen el objetivo de explotar diversos módulos y afectar directamente al sistema operativo.

Existen aplicaciones disfrazadas como juegos, aplicaciones de televisión o simuladores de controles remoto. El software malintencionado se ocultaba en forma de anuncios publicitarios (*adware*) y es capaz de controlar la funcionalidad de desbloqueo de la pantalla de un dispositivo y ejecutarse en segundo plano.

The Futex Vulnerability (Junio de 2014)

Mecanismo en el Kernel de Linux que permite realizar interbloqueos en programas que necesiten ejecutar instrucciones en paralelo sobre regiones compartidas de memoria.

El impacto y objetivo principal de explotar esta vulnerabilidad es conceder privilegios de **root** a dispositivos Android con el Kernel vulnerable, manipulando el sistema de forma que el atacante puede hacer los cambios que él quiera afectando de manera directa en la configuración y funcionamiento del mismo.

Stagefright (Julio de 2015)

La vulnerabilidad se centra en formatos de audio en video los cuales pueden contener código malicioso para ejecutarse con los mismos privilegios del servicio vulnerable en cuestión, los cuales son bastante elevados justo antes de **root,** corrompiendo el sistema y dejando mucha fuga de información que puede ser sensible para el usuario, teniendo, teniendo un control del sistema de mejor forma haciendo cambios que afecten de forma directa al sistema operativo.

3.4 ANÁLISIS DE SEGURIDAD EN LAS VULNERABILIDADES ANDROID

MALWARE

El objetivo es infectar y propagarse al mayor número de dispositivos posible, al tiempo que maximiza el daño al sistema, sin ningún motivo específico detrás de un ataque. Los malware actuales son más sofisticados y estratégicamente planificados. Ya que se considera como un software malintencionado en la que se incluyen los virus, rootkits, troyanos, etc., y es más comúnmente relacionado con la destrucción o robo de datos.

VULNERABILIDADES DE SOFTWARE

Hoy en día se manejan términos como el 0-day vulnerability (vulnerabilidad de día cero), que se refiere a una nueva vulnerabilidad para la cual aún no se crean parches o revisiones, y que se emplea para llevar a cabo un ataque. El nombre 0-day (día cero) se debe a que aún no existe ninguna revisión para mitigar el aprovechamiento de la vulnerabilidad.

Las vulnerabilidades de día cero, o zero-day son de las más peligrosas, ya que los atacantes pueden aprovecharse de fallos para los que todavía no hay parche de seguridad.

Una nueva vulnerabilidad crítica de día cero ha sido descubierta en el kernel de Linux a principios del año 2016. **Éste podría permitir a los atacantes obtener acceso root y controlar los sistemas de dispositivos Android o Linux afectados**.

3.5 INFORME SOBRE EL ESTADO DE LA SEGURIDAD 2019H1

En esta sección se discuten las vulnerabilidades destacadas y los Top 10 CWE de acuerdo al informe sobre el estado de seguridad 2019H1 (ElevenPaths, 2019).

3.5.1 VULNERABILIDADES DESTACADAS

A continuación se analiza el impacto de las vulnerabilidades más representativas.

CVE-2019-12735 Vulnerabilidad de código de ejecución (Editor de textos Vim y NeoVim)

Impacto: Tan solo es necesario agregar una línea comentada a un archivo de texto plano con contenido malicioso y esperar a que un usuario de este editor de texto abra el archivo para que dicho contenido se ejecute en el sistema. Divulgación total de información, lo que resulta en la revelación de todos los archivos del sistema. Pérdida completa de la protección del sistema, lo que da como resultado que todo el sistema se vea comprometido. Hay un cierre total del recurso afectado. El atacante puede dejar el recurso completamente disponible.

CVE-2019-11683 Vulnerabilidad Denegación de servicio Corrupción de memoria (Kernel de linux)

Impacto: En el Kernel de Linux 5.x anterior a 5.0.13 permite a los atacantes remotos causar una denegación de servicio (corrupción de la memoria fuera de los límites) o posiblemente tener otro impacto no especificado a través de paquetes UDP con una carga útil de 0, El atacante puede dejar el recurso completamente disponible. Hay una pérdida completa de la protección del sistema, lo que da como resultado que todo el sistema se vea comprometido. Divulgación total de información, lo que resulta en la revelación de todos los archivos del sistema.

CVE-2019-11477 Vulnerabilidad Denegación de servicio desbordamiento (Kernel de Linux)

Estaba sujeto a un desbordamiento de enteros en el kernel de Linux cuando se manejan los Agradecimientos selectivos de TCP (SACK). Un atacante remoto podría usar esto para causar una denegación de servicio. El atacante puede dejar el recurso completamente disponible. No existen condiciones de acceso especializadas o circunstancias atenuantes. Se requiere muy poco conocimiento o habilidad para explotar.

3.5.2 TOP 10 CWE

A continuación se describe el impacto de las 10 CWE más representativas durante 2018 y 2019

CWE-79 Neutralización inadecuada de la entrada durante la generación de la página web ('Scripting entre sitios')

Impacto: Una vez que se inyecta el script malicioso, el atacante puede realizar una variedad de actividades maliciosas. El atacante podría transferir información privada, como cookies que pueden incluir información de sesión, desde la máquina de la víctima al atacante. El atacante podría enviar solicitudes maliciosas a un sitio web en nombre de la víctima, lo que podría ser especialmente peligroso para el sitio si la víctima tiene privilegios de administrador para administrar ese sitio. Los ataques de phishing podrían usarse para emular sitios web confiables y engañar a la víctima para que ingrese una contraseña, permitiendo que el atacante comprometa la cuenta de la víctima en ese sitio web. Finalmente, el script podría explotar una vulnerabilidad en el navegador web, posiblemente tomando el control de la máquina de la víctima, a veces referido como "piratería de disco".

En muchos casos, el ataque se puede lanzar sin que la víctima se dé cuenta. Incluso con usuarios cuidadosos, los atacantes utilizan con frecuencia una variedad de métodos para codificar la parte maliciosa del ataque, como la codificación de URL o Unicode, por lo que la solicitud parece menos sospechosa.

CWE-119 Restricción incorrecta de operaciones dentro de los límites de un búfer de memoria

Impacto técnico: ejecutar código o comandos no autorizados; Modificar memoria.

Impacto en confidencialidad, integridad y disponibilidad: Si la memoria accesible por el atacante se puede controlar de manera efectiva, puede ser posible ejecutar código arbitrario, como con un desbordamiento de búfer estándar. Si el atacante puede sobrescribir la memoria de un puntero (generalmente 32 o 64 bits), puede redirigir un puntero de función a su propio código malicioso. Incluso cuando el atacante solo puede modificar un solo byte, la ejecución de código arbitrario puede ser posible. A veces esto se debe a que el mismo problema puede explotarse repetidamente con el mismo efecto. Otras veces es porque el atacante puede sobrescribir datos específicos de la aplicación críticos para la seguridad, como un indicador que indica si el usuario es un administrador.

Impacto técnico: memoria de lectura; DoS: bloqueo, salida o reinicio; DoS: consumo de recursos (CPU); DoS: consumo de recursos (memoria).

CWE-20 Validación de entrada incorrecta

Impacto técnico: DoS: bloqueo, salida o reinicio; DoS: consumo de recursos (CPU); DoS: consumo de recursos (memoria).

Impacto en disponibilidad: Un atacante podría proporcionar valores inesperados y provocar un bloqueo del programa o un consumo excesivo de recursos, como la memoria y la CPU. Impacto técnico: memoria de lectura; Leer archivos o directorios.

Impacto de confidencialidad: Un atacante podría leer datos confidenciales si puede controlar las referencias de recursos. Impacto técnico: Modificar memoria; Ejecutar código o comandos no autorizados

Integridad, confidencial y disponibilidad: Un atacante podría usar entradas maliciosas para modificar datos o posiblemente alterar el flujo de control de maneras inesperadas, incluida la ejecución arbitraria de comandos.

CWE-284 Control de acceso incorrecto

Impacto técnico: varía según el contexto, cuando el mecanismo de control de acceso falla , se puede comprometer la seguridad del software para obtener los privilegios del sistema, así como leer información confidencial del usuario, ejecutar comandos de manera arbitraria y evadir la detección dejando el sistema totalmente comprometido.

CWE-200: Exposición de información confidencial a un actor no autorizado

Confidencialidad. Impacto técnico: Leer los datos de la aplicación, divulgación y fuga de información sensible del usuario el cual puede contar con datos financieros, información personal, etc. la cual se ve comprometida completamente para poder ser utilizada con los fines que el atacante desee.

CWE-264 permisos, privilegios y controles de acceso

Impacto: Tener el control total del sistema operativo en modo “root” de manera que puede administrar lo que se desee dentro del mismo, información. Fuga o pérdida de información confidencial. El atacante puede dejar el recurso completamente disponible.

CWE-125 Lectura fuera de límites

Impacto técnico: memoria de lectura y mecanismo de protección de derivación.

Impacto: Al leer la memoria fuera de los límites, un atacante podría obtener valores secretos, como direcciones de memoria, que pueden omitir mecanismos de protección como ASLR para mejorar la confiabilidad y la probabilidad de explotar una debilidad separada para lograr la ejecución del código en lugar de simplemente denegación de servicio.

CWE-255 Errores de gestión de credenciales

Impacto: Fuga o pérdida de información confidencial. El atacante puede dejar el recurso completamente disponible.

CWE-77 Neutralización incorrecta de elementos especiales utilizados en un comando ('Inyección de comandos')

Impacto técnico: ejecutar código o comandos no autorizados

Impacto: Si un usuario malintencionado inyecta un carácter (como un punto y coma) que delimita el final de un comando y el comienzo de otro, puede ser posible insertar un comando completamente nuevo y no relacionado que no estaba destinado a ejecutarse.

CWE-416 Uso después de gratis

Impacto técnico: Modificar memoria

Impacto de integridad: El uso de memoria previamente liberada puede corromper datos válidos, si el área de memoria en cuestión se ha asignado y utilizado correctamente en otro lugar. Impacto técnico: DoS: bloqueo, salida o reinicio.

Impacto de Disponibilidad: Si la consolidación de fragmentos se produce después del uso de datos previamente liberados, el proceso puede bloquearse cuando se utilizan datos no válidos como información de fragmentos.

Impacto de Integridad, confidencialidad y disponibilidad: Si se ingresan datos maliciosos antes de que pueda llevarse a cabo la consolidación de fragmentos, es posible aprovechar una primitiva de escribir qué para ejecutar código arbitrario. Impacto técnico: ejecutar código o comandos no autorizados.

El impacto de dichas vulnerabilidades nos lleva a saber que el principal objetivo es suplantar la identidad del usuario en modo root para el robo o secuestro de la información, así como explotar diversos módulos que son vulnerables del sistema operativo debido a los errores que se cometieron durante el diseño y programación del mismo.

3.6 IMPACTO DE LAS VULNERABILIDADES

Se analiza el impacto que tienen las vulnerabilidades encontradas, el daño o consecuencia que ocasionaron al dispositivo móvil.

3.6.1 KERNEL

De acuerdo a las 2 vulnerabilidades del Kernel que se encuentran en el estado del de arte:

Vulnerabilidad de ganar privilegios

Impacto: Conseguir privilegios de **root** en el Kernel, es decir ser el usuario privilegiado, donde el usuario está expuesto completamente, pierde el control del sistema operativo de manera que el atacante puede hacer cualquier cuestión de administración y configuración del mismo sin que el usuario legitimo lo autorice.

El año con más vulnerabilidades de ganar privilegios fue en **2016** con un total de 250 donde alcanzó su pico más alto, ha ido disminuyendo considerablemente a tal grado de tener solo 1 vulnerabilidad documentada para el año **2019**, en este apartado de seguridad que es primordial tanto para un usuario como para el sistema operativo, muestra una mejoría considerable lo cual han ido solucionando dando un mayor soporte al usuario y seguridad sobre todo al sistema operativo Android en general (CVE Details, 2020).

Vulnerabilidad DoS

Impacto: Tiene un alto impacto de confidencialidad, integridad y disponibilidad, lo que hace posible que los atacantes potenciales puedan acceder a todos los recursos, modificar cualquier archivo y negar el acceso a los recursos después de explotar con éxito la vulnerabilidad, desactivar el antivirus, instalar y crear malware en el dispositivo e incluso la publicación de aplicación maliciosas en Google PlayStore para cuando la aplicación la descargue e instale la misma inyecte el malware.

Las vulnerabilidades de DoS son de las que más afectan al sistema operativo Android, del año 2009 al 2019 se documentaron 329 vulnerabilidades de las cuales hace un 12.8%, el año 2016 resalta por ser el año con más vulnerabilidades del 2009 a la fecha con un total de 106, lo cual ha ido disminuyendo pero sin poder erradicarla, para el año 2019 se documentaron 35 vulnerabilidades lo cual la reduce mas no desaparece (CVE Details, 2020).

3.6.2 APLICACIONES BASADAS EN GESTOS

El impacto primordial es que la seguridad de las aplicaciones no es la correcta ya que los desarrolladores de las mismas se basan principalmente en las funciones olvidando esa parte fundamental de las mismas. La seguridad de las aplicaciones es un punto muy relevante ya que alojan información importante y sensible del usuario principalmente la cual es vinculada a cuentas privadas.

Pérdida de Autenticación y Gestión de Sesiones implementadas incorrectamente permiten comprometer los datos de usuarios y contraseñas, los **token** de sesión y así suplantar la identidad, con todas las consecuencias que esto puede acarrear.

Exposición de Datos Sensibles, particularmente en las aplicaciones que no protegen los datos de respuesta pueden ser una interesante fuente de información.

Al poder eludir los mecanismos de defensa de la aplicación el hacker puede ver toda la información sensible que pudiera tener el usuario vinculado con su cuenta con la que inicia dicha aplicación. El impacto sería muy grave ya que toda la información estaría disponible para poder hacer los cambios que el desee.

Más específicamente, la cerradura basada en el gesto se implementa para evitar el acceso no autorizado a la cuenta predeterminada en las aplicaciones. Limitando el número de intentos de inicio de sesión para ser 5 en una cerradura basada en gestos, la aplicación va a rechazar a un usuario a iniciar sesión con la cuenta predeterminada si él o ella no pueden introducir el gesto correcto dentro de 5 veces. Por otra parte, para defenderse de los ataques de fuerza bruta, la cuenta predeterminada en una aplicación se borrará una vez que los intentos de entrada exceden un valor preestablecido.

Muchos bloqueos basados en gestos pueden ser anulados fácilmente debido a la existencia de la vulnerabilidad en las aplicaciones. Como resultado, la cuenta de usuario está en grave riesgo de ser divulgada, lo que puede conducir a la pérdida de privacidad o violaciones financieras personales.

De acuerdo a las vulnerabilidades, se desarrolló una herramienta llamada LockBreaker (Interruptor de bloqueo) que función primordial es detectar automáticamente las vulnerabilidades de bloqueo basadas en gestos de una manera eficiente.

Se probaron 63 aplicaciones muy populares del mercado de Android usando el interruptor de bloqueo. Los resultados arrojaron que 28 de las 63 se enfrentan a amenazas de seguridad, y 13 de las 28 aplicaciones tienen vulnerabilidades obvias.

El objetivo principal es ganar información sensible del usuario así como ganar privilegios en el sistema de manera que pueda acceder al mismo las veces que el atacante desee, haciendo cambios significativos que puedan cambiar la configuración del sistema operativo haciendo que el usuario no pueda utilizar sus servicios de forma habitual como lo hacía , quedando inhabilitados los servicios. Cabe mencionar que cuando el usuario configura a su manera un dispositivo móvil lo hace de acuerdo a sus necesidades pero en muchas ocasiones no se tiene la cultura informática adecuada y el mismo puede abrir puertas para que los atacantes pueden explotar las vulnerabilidades existentes en el dispositivo de una forma sencilla, hay que ser más responsables y tener una buena cultura informática al respecto.

3.6.3 PLATAFORMAS MÓVILES

Se describen las vulnerabilidades más comunes en las plataformas móviles así como el impacto generado por las mismas:

Los dispositivos móviles no tienen contraseñas habilitadas.

Impacto: Puede ocasionar el robo de información fácilmente, la manipulación de la misma, la modificación, fuga de datos, así como datos personales los cuales pueden ser usados para realizar operaciones malintencionadas.

Conexión a red Wi-Fi gratis

Impacto: Los usuarios que hacen uso de ellas se encuentran totalmente expuestos al robo de información confidencial por hackers, ya que si la transmisión no se encuentra cifrada es muy fácil el interceptar los datos y queda muy expuesto a perder la información más sensible con la que cuente el usuario.

Dispositivos infectados con malware

Impacto: El principal objetivo es el robo de información privada del usuario como datos personales, o bancarios, secuestros de información así como instalación de programas sin el consentimiento del usuario,

Canales de comunicación abiertos

Impacto: Que los hackers puedan hacer conexiones al dispositivo de manera que pueda instalar malware, que active el micrófono o la cámara para espiar al usuario.

Según Umasankar, (2017) define en la siguiente tabla las vulnerabilidades más representativas donde se analizara el impacto correspondiente de dichas vulnerabilidades.

Ganar privilegios

Impacto: Al escalar los privilegios del sistema operativo Android, el objetivo es ser usuario root de manera que se pueda tener control absoluto del sistema, pudiendo hacer las modificaciones necesarias de acuerdo al daño que se desee hacer directamente al sistema operativo de Android. Obtendría acceso a los datos del usuario pudiendo hacer lo que desee con ellos.

Ejecución remota de código

Impacto: El atacante puede ejecutar cualquier comando o código en el dispositivo para explotar algo específico del sistema operativo, dicha vulnerabilidad de acuerdo a las investigaciones tiene el identificador siguiente CVE-2016-3820.

Denegación de Servicio

Impacto: El impacto de esta vulnerabilidad tiene el objetivo de hacer que el servicio no esté disponible para el usuario legítimo de manera que no lo pueda utilizar.

Desbordamiento de buffer

Impacto: La vulnerabilidad se relaciona principalmente con escalar los privilegios de tal manera que el hacker pueda escribir enormes cantidades de datos en el buffer para hacerlos fallar en algún momento que se ocupe el sistema, de igual forma la aplicación maliciosa puede omitir los privilegios o permisos.

Ganar información

Impacto: Robo de información personal, sensible y confidencial del usuario, tener cuentas bancarias, tarjetas de crédito para hacer un mal uso de las mismas por ejemplo.

Corrupción de memoria

Impacto: Fuga de información del usuario, el manipular la memoria de un programa vulnerable para forzar a la aplicación a comportarse de manera diferente a la que fue programada así como desbordamiento de buffer.

Mencionaron Zhang et al., (2015) las vulnerabilidades de corrupción de memoria a menudo se originan a partir del acceso ilegal a los objetos de memoria, como las vulnerabilidades de desbordamiento de pila, por lo que el límite del objeto de memoria es un factor crítico para detectar estas vulnerabilidades. Existe una variedad de objetos de memoria en el lenguaje de programación de alto nivel, como la matriz, clase y así sucesivamente, podríamos revisar el código fuente.

El Impacto que generan las vulnerabilidades de corrupción de memoria se centran en la fuga de datos privados con los que cuenta el usuario. Los ataques por corrupción de memoria permiten manipular la memoria del programa vulnerable para forzar a la aplicación a comportarse de manera diferente a la que fue programada. De igual forma los ataques más comunes son el desbordamiento de buffer en la pila, cadenas de formato y desbordamiento de buffer en el montículo (del término en inglés “heap”).

CVE Details (2020), menciona el número de vulnerabilidades de corrupción de memoria del año 2011 al 2019:

- Año 2011 registro 1 vulnerabilidad
- Año 2012 registro 0 vulnerabilidades
- Año 2013 registro 2 vulnerabilidades
- Año 2014 registro 0 vulnerabilidades
- Año 2015 registro 46 vulnerabilidades
- Año 2016 registro 38 vulnerabilidades
- Año 2017 registro 32 vulnerabilidades
- Año 2018 registro 12 vulnerabilidades
- Año 2019 19 vulnerabilidades

La vulnerabilidad de corrupción de memoria en los últimos años arroja un total de 150 vulnerabilidades de las cuales afecto a un 5.9% de los usuarios Android.

Linares et al.,(2017) Realizaron un estudio en general consiste en 660 vulnerabilidades detectadas, extraídas de los boletines de seguridad de Android del sitio web oficial, a partir de agosto de 2015 a noviembre de 2016 se encontraron 16 boletines publicados por Google donde se detectaron 564 vulnerabilidades, se realizó una taxonomía de las vulnerabilidades detectadas

El impacto más importante fue en las capas o subsistemas del sistema operativo Android que se vieron afectados, como se observa en la Figura 5, la capa que se vio más afectada fue el kernel de Linux con 41%, las librerías nativas con 37%, las aplicaciones con 15%, el marco de aplicaciones 7%, el 3% capa de abstracción de hardware y el 2% Android Runtime.

Las vulnerabilidades afectan con mayor frecuencia son a la memoria, con 103 casos (20%).

Vulnerabilidades relacionadas con permisos, privilegios y control de acceso con 58 casos (11%).

Meshram et al., (2014) La cantidad de aplicaciones móviles maliciosas en teléfonos inteligentes basados en Android ha aumentado rápidamente. Además, estas aplicaciones maliciosas son capaces de descargar módulos de servidores que se ejecutan por usuarios malintencionados, lo que significa que los eventos inesperados pueden activarse dentro de los teléfonos inteligentes. Por lo tanto, el atacante puede controlar, obtener información personal y datos almacenados dentro del teléfono inteligente ilegalmente.

Dan respuesta a las amenazas en dispositivos móviles inteligentes, una gran cantidad de proveedores de seguridad han estado ofreciendo productos de seguridad. El método más común para la detección de aplicaciones maliciosas utilizadas por los productos de seguridad es extraer la firma de una aplicación y luego se compara con las firmas en la base de firmas de aplicación maliciosa. Si son lo mismo, por ende la aplicación es maliciosa.

Los problemas de seguridad que enfrentan los dispositivos móviles

Los problemas de seguridad que enfrentan los dispositivos móviles, son los siguientes aspectos:

1. Aplicaciones maliciosas. Recientemente, el número de aplicaciones maliciosas está creciendo rápidamente. Las aplicaciones maliciosas no sólo se encuentran en algún mercado de aplicaciones de terceros, también encontrado en la tienda oficial de Android (Google Play Store). Problemas muy graves son causados por estas aplicaciones maliciosas. Algunas aplicaciones pueden revelar

la ubicación del usuario, contactos y otra información personal; algunos pueden enviar mensajes cortos o hacer llamadas telefónicas en segundo plano con el objetivo de generar beneficio; algunas aplicaciones descargarán otras aplicaciones maliciosas a través de internet.

2. Sitios web inseguros. Algunos sitios pueden contener gran número de aplicaciones maliciosas. Además, muestra que un usuario de un teléfono Android que utiliza el navegador web para navegar por Internet puede ser explotado si visita una página maliciosa y el atacante puede ejecutar cualquier código con los privilegios de la aplicación del navegador web.

3. Seguridad de datos en dispositivos móviles. Los dispositivos móviles inteligentes son diferentes de las PC debido a su portabilidad, y por lo tanto están en un mayor riesgo de pérdida. Cuando un usuario pierde su dispositivo, los datos de los dispositivos son difíciles de recuperar. Algunas otras razones como el mal uso de los usuarios, el daño de dispositivos también puede causar la pérdida de datos.

4. Seguridad de datos de red de los dispositivos móviles. Cuando un usuario se conecta a un punto de acceso configurado por el atacante, el tráfico de su dispositivo móvil puede ser olfateado. Muestra que, el atacante puede configurar un hotspot. Cuando un usuario utiliza este punto de acceso para navegar por Internet, la información personal y las credenciales del usuario (incluido nombre de usuario, contraseña, números de tarjeta de crédito, etc.) se revelan al atacante. Incluso si el usuario utiliza una conexión SSL, el atacante tiene muchos métodos para obtener la información personal del usuario.

3.6.4 LA NUBE

Con el servicio de almacenamiento y procesamiento de la computación en la nube, teléfonos inteligentes han sido ampliamente utilizados en la vida diaria, el trabajo, el aprendizaje de, la comunicación y la amorfidad. En el mercado mundial Smartphone, Android es el sistema operativo más exitoso, compartido el 75% del mercado durante el tercer trimestre de 2012, con 500 millones de dispositivos activados en total y 1,3 millones de activaciones por día.

En la actualidad una de las tendencias del mercado de los sistemas de información es la proliferación de los servicios operando en la nube, los cuales son servicios que permiten la asignación dinámica de recursos en función de necesidades de los clientes y que aportan una reducción de costes en infraestructuras considerable

La computación en la nube es un nuevo modo de facilitar recursos de computación, no una nueva tecnología. Ahora los servicios de computación, desde el almacenamiento y procesamiento de datos hasta el software, como la gestión del correo electrónico, están disponibles de forma instantánea, sin compromiso y bajo demanda.

Los principales riesgos que se pueden presentar en la nube son los siguientes en donde se ve el impacto de dichos riesgos:

- Perdida o fuga de información. En la nube, aumenta el riesgo de que los datos se vean comprometidos ya que el número de interacciones entre ellos se multiplica debido a la propia arquitectura de la misma. Esto deriva en daños económicos y, si se trata de fugas, problemas legales, infracciones de normas, etc.

- Suplantación de identidad. En un entorno en la nube, si un atacante obtiene las credenciales de autenticación de un usuario del entorno puede acceder a actividades y transacciones, manipular datos, devolver información falsificada o redirigir a los clientes a sitios maliciosos.

- Abuso y mal uso de Cloud computing. Cualquiera con una tarjeta de crédito válida puede acceder al servicio, con la consecuente proliferación de spammers, creadores de código malicioso y otros criminales que utilizan la nube como centro de operaciones.

- Ataques de denegación de servicio. Pueden dejar sin acceso completamente a los usuarios legítimos.

- Vulnerabilidades de los sistemas. Las vulnerabilidades del sistema son bugs explotables en programas que los atacantes pueden usar para infiltrarse en un sistema para robar datos, tomar el control o interrumpir el servicio. Este tipo de vulnerabilidad es típica de cualquier sistema informático, pero en la nube cobran una mayor importancia.

3.7 GUÍA DE RECOMENDACIONES TÉCNICAS SOBRE LA SEGURIDAD EN ANDROID

Después de analizar las vulnerabilidades documentadas en el estado del arte, en esta sección se listan unas recomendaciones generales para proteger los dispositivos móviles.

- Mantener actualizado el sistema operativo: no se debe hacer caso omiso de las notificaciones, ya que el sistema manda parches continuamente de seguridad con la finalidad de corregir los defectos que posee el sistema operativo así como mejoras al sistema en todos los aspectos, lo cual llega a reducir fallos en el mismo.

- Seguridad: Mantener la confidencialidad y seguridad de su dispositivo como no compartir contraseñas, cuentas bancarias, tarjetas de crédito, etc., con la finalidad de no dejar expuesta información sensible del usuario para evitar suplantación de identidad o mal uso de datos.

- Se recomienda utilizar el software DroidRanger que está diseñado para detectar aplicaciones maliciosas en los mercados populares de Android para saber y tener un control más exacto y fidedigno de las aplicaciones que se instalan en los dispositivos inteligentes.

- Utilizar el software NIVAnalyzer, detecta y verifica automáticamente las vulnerabilidades de la siguiente intención con el fin de hacer un escaneo más certero de estas vulnerabilidades que afectan gravemente en el rubro de las aplicaciones instaladas en el sistema operativo Android teniendo un mejor control de las mismas.

- Tener un software de seguridad que permita detectar cualquier tipo de amenaza o intrusión en el equipo (Antivirus). El uso de antivirus disminuye posibles riesgos de aplicaciones, documentos, fotos, etc. que se encuentren infectados, realiza un escaneo en donde las localiza y hace un barrido de las mismas para quitarlas del sistema operativo.

- No revelar las credenciales con las que se loguea al sistema operativo para evitar entrar a intrusos los cuales realicen la suplantación de identidad, para no exponer la información, control y administración del dispositivo.

- Instalar aplicaciones de repositorios oficiales, como por ejemplo (Play Store). Hay que evitar instalar aplicaciones no comercializadas desactivando la opción ""Fuentes desconocidas" en el menú de Configuración – Aplicaciones. Comprobar los comentarios y las calificaciones de otros usuarios para asegurarse que la aplicación no representa ningún peligro. Leer la política de privacidad de la aplicación (si la tiene), con el fin de conocer la información a la que tiene acceso y si va a compartir sus datos con terceros.

- Verificar la reputación de la aplicación antes de instalarla, ya que a veces las vulnerabilidades en el software instalado pueden llevar a la explotación de las mismas y qué se realice el robo de información de manera remota.

- Conectarse a una red inalámbrica segura ya que si el dispositivo se conecta a una red inalámbrica pública están puertos abiertos por los cuales el atacante puede robar la información que desee del usuario,

- Habilitar el dispositivo con una clave segura. Con lo cual se impedirán los accesos no autorizados. De igual forma, se recomienda configurar el dispositivo para que bloquee automáticamente el acceso tras un determinado período de tiempo sin interacción alguna.

- No darle permisos a las aplicaciones donde puedan acceder a información sensible en el sistema, ya que pueden acceder a los contactos, cuentas bancarias, cámara, gestor de archivos, etc., de manera que la información e identidad quedan expuestos para ser robados.

- Encriptar archivos. Es muy útil el cifrar los archivos relevantes. El cifrado impide que el atacante tenga acceso a la información y a las imágenes de los dispositivos o servicios de copia de seguridad en línea, al cifrar se aseguran nuestros datos lo cual hace que el atacante no pueda tener control ni acceso en nuestros archivos e información.

- Tener desactivado o deshabilitado los dispositivos de comunicación como Bluetooth para no permitir el acceso a intrusos, ya que si los tenemos habilitados se quedan abierto los puertos que utilizan estos componentes y por ahí el intruso entre a realizar ataques al dispositivo.

- Proteger el dispositivo en caso de pérdida o robo. Proteger el dispositivo mediante un PIN, un patrón de desbloqueo o una contraseña. Esto dificultará el acceso a la información y permitirá incluso llevar a cabo algunas acciones antes de que el delincuente acceda a información personal. Instalar alguna aplicación de control remoto en los dispositivos. Los principales fabricantes de móviles o grandes empresas como Google disponen de sus propias aplicaciones.

- Utilizar un firewall de seguridad. Esta aplicación monitorea todas las comunicaciones entrantes y salientes en función de un conjunto de reglas preestablecidas para prevenir que intrusos no ingresen al sistema de manera ilegal con el fin de hacer daño al usuario.

- Crear copias de seguridad de los datos para no perder la información relevante de nuestro dispositivo y mantenerla disponible por cualquier altercado.

- No modificar configuraciones básicas del dispositivo que traen de fábrica por el proveedor ya que viene diseñado para proteger el dispositivo, ya que se debilitara el mismo y quedaría expuesto a posibles afectaciones del dispositivo.

- No realizar operaciones bancarias y compras online a través de conexiones Wi-Fi públicas ya que existen puertos abiertos en las redes públicas por los cuales pueden robar los datos. Es mejor reservar dichas transacciones para cuando se esté conectado a una red debidamente protegida.

CONCLUSIONES

En la Sección 1.1 se comentó que el sistema operativo Android es el más usado a nivel mundial con un 74.43% del mercado total, esto debido a que es un sistema intuitivo, fácil de usar, cómodo, flexible y robusto, no obstante debido a que el sistema es de código abierto quien sepa el lenguaje java puede modificarlo, por ende es el sistema más atacado por los hackers que intentan obtener información sensible del usuario, ganar privilegios, desactivar servicios o incluso corromper todo el sistema.

Es evidente por la información del Capítulo 2, que la seguridad es una de las características más importantes a ser implementada en un sistema operativo, y a su vez el más débil sino se tienen buenas medidas para evitar penetraciones de hackers. Debido a esto, es importante sensibilizar a los usuarios sobre el uso seguro de estos dispositivos con el fin de tratar de mantener la integridad del sistema, considerando que las vulnerabilidades no están exentas en todos los sistemas operativos.

Resaltando lo que se documentó en la Sección 2.5, varias de las vulnerabilidades de Android suelen ser por la falta de conocimientos y por el poco cuidado de los usuarios al instalar aplicaciones esto va enfocado a tener una cultura informática del usuario el cual juega también un papel muy importante en este rubro. Esto deja claro que una buena cultura informática es necesaria para un uso eficiente de las tecnologías de la información.

Otro punto muy importante es mantener actualizado el sistema operativo, ya que son liberados continuamente parches de seguridad que arreglan fallas detectadas, no con esto se garantiza que no se puedan explotar las diversas vulnerabilidades existentes pero se disminuye el riesgo de verse comprometidos y afectados por los atacantes.

En la Sección 2.5.1 se describe una forma de consolidar las vulnerabilidades utilizando la herramienta NIVAnalyzer, la cual realiza un análisis eficiente y automático de las vulnerabilidades del sistema operativo, es una buena medida de prevención para contrarrestar el daño de las mismas antes de que puedan explotarse del todo en el sistema operativo Android.

De acuerdo a la información presentada en la Sección 2.3.1, referente a la plataforma CVE Details se observa en la gráfica de la Figura 8, a partir del año 2015 hay un incremento en el número de vulnerabilidades, alcanzando un valor máximo (843) en el año 2017. Sin embargo, a partir del mismo año 2017 se observa un decrecimiento. Esto puede deberse a que debido al extenso uso del sistema operativo en dispositivos móviles, se está fortaleciendo su seguridad.

Se recomienda consultar con regularidad los repositorios donde están documentadas las vulnerabilidades del sistema operativo Android, por ejemplo: la plataforma CVE Details (CVE Details, 2020), la base de datos nacional de vulnerabilidades (Nist, 2020), el catálogo de vulnerabilidades de Serkan Özkan (IT Security Database, 2020), la fundación de seguridad OWASP (OWASP, 2020) y la plataforma de vulnerabilidades comunes (CWE, 2020), las cuales nos proporcionan estadísticas confiables del estado actual de la seguridad de este sistema operativo en sus diversos rubros.

Como se describe en la Sección 3.7, guía de recomendaciones técnicas sobre la seguridad en Android, todo sistema operativo es tan seguro como sea configurado, debido a que una mala configuración expone el sistema a las vulnerabilidades. Nuevamente, esto va relacionado con la cultura de seguridad de la información del usuario.

Finalmente, analizando el contenido de esta tesina, se han documentado las vulnerabilidades del sistema operativo Android disponibles en el estado del arte, con lo cual se identificaron algunas recomendaciones que permiten proteger los dispositivos inteligentes donde dicho sistema operativo esté instalado. Por lo tanto, se cumple el objetivo general planteado para esta tesina.

ANEXO A. GLOSARIO DE TÉRMINOS

Vulnerabilidad: es una debilidad o fallo en un sistema de información que pone en riesgo la seguridad de la información pudiendo permitir que un atacante pueda comprometer la integridad, disponibilidad o confidencialidad de la misma, por lo que es necesario encontrarlas y eliminarlas lo antes posible.

Sistema de Archivos: Es una estructura de directorios que se puede usar para organizar, almacenar y administrar los archivos.

Android Runtime: cada app ejecuta sus propios procesos con sus propias instancias del tiempo de ejecución de Android

Caja blanca: Es un tipo de pruebas de software que se realiza sobre las funciones internas de un módulo. Entre las técnicas más usadas se encuentran; la cobertura de caminos (pruebas que hagan que se recorran todos los posibles caminos de ejecución), pruebas sobre las expresiones lógico-aritméticas, pruebas de camino de datos (definición de uso de variables), comprobación de bucles (se verifican los bucles para 0,1).

Caja negra: Es aquel elemento que es estudiado desde el punto de vista de las entradas que recibe y las salidas o respuestas que produce, sin tener en cuenta su funcionamiento interno. De una caja negra nos interesara su forma de interactuar con el medio que lo rodea (en ocasiones otros elementos que también podrían ser cajas negras) entendiendo que es lo que hace, pero sin dar importancia a como lo hace.

Sys-setuid: Función de Linux que sirve para establecer la identidad del usuario. Establece el ID de usuario efectivo del proceso de llamada. Si el proceso de llamada es privilegiado (más precisamente: si el proceso tiene la CAP_SETUID en su espacio de nombre de usuario), el UID real y guardado set-user-ID también se establecen.

Kallsyms: Juega un papel muy importante en la depuración del código fuente. En el proceso de compilación, el kernel de Linux guarda todos los símbolos del kernel (todas las funciones y módulos del kernel ya cargados) y la dirección del símbolo y el tipo de símbolo en el archivo / proc / kallsyms. Formato específico: dirección simbólica, tipo de símbolo, nombre simbólico

Root: El usuario que posee mayor nivel de privilegios. De hecho, es el único que tiene privilegios sobre todo el sistema en su globalidad, así como el responsable de las tareas administrativas.

Código Smali: Utilizando smali/baksmali (ensamblador/desensamblador) se puede obtener una representación en un lenguaje de bajo nivel con el que se puede trabajar más fácilmente, al cual llamaremos código Smali,

Exploit: Programa o código que se aprovecha de un agujero de seguridad (vulnerabilidad) en una aplicación o sistema, de forma que un atacante podría usarla en su beneficio.

Divulgación de información: Este término se utiliza con frecuencia en avisos de vulnerabilidad para describir una consecuencia o impacto técnico de cualquier vulnerabilidad que tenga una pérdida de confidencialidad.

Fuga de información: Es un término de uso frecuente, sin embargo, el término "fuga" tiene múltiples usos dentro de la seguridad. En algunos casos se trata de la exposición accidental de información de una debilidad diferente, pero en otros casos (como "fuga de memoria"), se trata de un seguimiento inadecuado de los recursos, lo que puede llevar al agotamiento.

Instancia: Se llama instancia a todo objeto que derive de algún otro. De esta forma, todos los objetos son instancias de algún otro. La instancia es el objeto ya creado.

Framework: Un Framework es una estructura previa que se puede aprovechar para desarrollar un proyecto. El Framework es una especie de plantilla, un esquema conceptual, que simplifica la elaboración de una tarea, ya que solo es necesario complementarlo de acuerdo a lo que se quiere realizar.

Códec multimedia: Se trata de un código que realiza su función cuando el sistema operativo o software lo demande. Este código convierte las señales digitales de audio y video en un formato que se pueda reproducir. Se puede usar en una cámara de fotos o en un Smartphone. Un códec codifica y comprime los datos de un archivo vídeo o audio para que sea más rápida su transferencia y ocupe menos espacio. Cuando reproducimos o editamos dicho archivo, se descomprime.

launcher de Android: Los launchers o lanzadores son uno de los componentes más importantes que se pueden encontrar en Android, ya que sirven para lanzar las aplicaciones que tienes instaladas. Es el entorno de escritorio de Android, el diseño que se va a encontrar cuando se ingrese en la pantalla principal del dispositivo móvil.

Manifest de Android: Todos los proyectos de aplicaciones deben tener un archivo AndroidManifest.xml (con ese mismo nombre) en la raíz de la fuente del proyecto. El archivo de manifiesto describe información esencial de la aplicación para las herramientas de creación de Android, el sistema operativo Android y Google Play. El archivo manifestó es la parte más importante en una aplicación Android si éste no se encuentra, la aplicación no va a funcionar.

Confidencialidad: Esta propiedad de seguridad de los datos hace referencia a “Garantizar que la información sea accedida solamente por las personas indicadas”, que no se revele información a quien no esté autorizado, la propiedad de confidencialidad debe existir en los datos en cualquiera de sus 3 estados: en almacenamiento, proceso y en tránsito.

Integridad: La integridad de la información se refiere a garantizar que la información no ha sido modificada, en cualquiera de sus 3 estados: en almacenamiento, proceso o en tránsito.

Disponibilidad: La disponibilidad de la información hace referencia a mantener activo el acceso a la información necesaria a aquellas personas que deben tener acceso a la misma en el momento que sea necesario.

Phishing: Es el delito de engañar a las personas para que compartan información confidencial como contraseñas y números de tarjetas de crédito. Las víctimas reciben un mensaje de correo electrónico o un mensaje de texto que imita (o “suplanta su identidad”) a una persona u organización de confianza, como un compañero de trabajo, un banco o una oficina gubernamental.

REFERENCIAS

- Android (2018). Android, el sistema operativo con más vulnerabilidades en 2017, Disponible en: http://www.eltiempo.com/tecnosfera/novedades-tecnologia/android-fue-el-sistema-operativo-con-mas-vulnerabilidades-en-2017-167314, [Consultado en Agosto 2018]
- Anaya O. (2015). Resolución de averías lógicas en equipos microinformáticos
- Azone (2018). Disponible en: https://androidzone.org/, [Consultado en Agosto 2018]
- Basterra,Bertea,Borello,Castillo,Venturi (2017) Android OS Documentation Release 0.1
- CVE Details (2020). The ultimate security vulnerability datasource, Disponible en: www.cvedetails.com, [Consultado en Febrero 2020]
- CWE (2020). Enumeracion de debilidades comunes, Disponible en: cwe.mitre.org, [Consultado en Septiembre 2020]
- Developers (2020). Android Developers, Disponible en : https://developers.android.com/, [Consultado en Septiembre 2020]
- ElevenPaths (2019) Informe sobre el estado de la seguridad 2019H1
- Escobar, J., Quinto, L. (2015). Vulnerability in Mobile Devices with Android Operating System. *Cuaderno Activa, 7, 55-65.*
- *ESET (2014). Reporte de seguridad en Latinoamérica, Disponible en: https://www.welivesecurity.com/wp-content/uploads/2014/06/informe_esr14.pdf*
- González (2017) Análisis y explotación de vulnerabilidades en Android
- Hei X., Du X., Lin S 2013 Two vulnerabilities in Android OS Kernel
- IDC (2014). Worldwide Mobile Phone Tracker, Disponible en: http://www.idckorea.com/
- IT Security Database (2020).Vulnerabilidad, parche y compilacion de datos fuente, Disponible en: www.itsecdb.com, [Consultado en Febrero 2020]
- Las versiones de Android y niveles de API, Disponible en:https://swcb37.files.wordpress.com/2014/01/las-versiones-de-android-y-niveles-de-api.pdf, [Consultado en Agosto 2018]
- La vulnerabilidad del kernel de Linux CONFIG_KEYS se extiende a Android, Disponible en: https://www.xatakandroid.com/seguridad/la-vulnerabilidad-del-kernel-de-linux-config-keys-se-extiende-a-android-comprueba-si-estas-afectado, [Consultado en Agosto 2018]
- Linares M., Bavota G., Escobar C 2017 An Empirical Study on Android Android-related Vulnerabilities
- Meshram P. D., Thool R. C (2014) A Survey Paper on Vulnerabilities in Android OS and Security of Android Devices
- Nist (2020). Base de datos nacional de vulnerabilidades, Disponible en www.nvd.nist.gov, [Consultado en Octubre 2020]
- OWASP (2020). Open Web Application Security Project®, Disponible en https://owasp.org/, [Consultado en Febrero 2020]
- ¿Qué riesgos de seguridad hay en Android además del malware?, Disponible en: https://www.welivesecurity.com/la-es/2014/07/16/riesgos-seguridad-android-ademas-malware/, [Consultado en Septiembre 2018]
- Tanenbaum A. (2009). Sistemas Operativos Modernos 3era Edición
- Tang J., Cui X., Zhao Z., Guo S., Xu X., Hu C., Ban T., Mao B (2017) NIVAnalyzer: a Tool for Automatically Detecting and Verifying Next-Intent Vulnerabilities in Android Apps
- Umasankar. Analysis of latest vulnerabilities in Android, 2017
- Veliz, Exequiel, Orlando, Dámian (2016) Estudio y analisis de seguridad en dispositivos moviles. BYOD y su impacto en las organizaciones.

- Wang Y., Zhang Y., Wang K., Yan J (2016) Security Analysis and Vulnerability Detection of Gesture-based Lock in Android Applications
- Wu J., Wu Y., Yang M., Wu Z., Wang Y (2013) Vulnerability Detection of Android System in Fuzzing Cloud
- Xataka (2018). Historia y evolución de Android: cómo un sistema operativo para cámaras digitales acabó conquistando los móviles, Disponible en: https://www.xatakandroid.com/sistema-operativo/historia-y-evolucion-de-android-como-un-sistema-operativo-para-camaras-digitales-acabo-conquistando-los-moviles, [Consultado en Agosto 2018]
- Zhang B., Wu B., Feng C., Tang C (2015) Memory Corruption Vulnerabilities Detection for Android Binary Software