
# ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO

## DESARROLLO DE UNA APLICACIÓN QUE PERMITA EL ESCANEO DE LAS VULNERABILIDADES EN LOS DISPOSITIVOS MÓVILES ANDROID PARA MITIGAR LOS PROBLEMAS DE SEGURIDAD

**AUTOR: HENRY MAURICIO VILLA YÁNEZ**

**Proyecto de investigación, presentado ante el Instituto de Postgrado y Educación Continua de la ESPOCH, como requisito parcial para la obtención del grado de MAGISTER EN SEGURIDAD TELEMÁTICA**

**RIOBAMBA - ECUADOR**

**MARZO 2016**

**ESCUELA SUPERIOR POLITÉCNICA DE CHIMBORAZO**

**CERTIFICACIÓN**

El Tribunal del PROYECTO DE INVESTIGACIÓN CERTIFICA QUE:

El trabajo de investigación titulado "DESARROLLO DE UNA APLICACIÓN QUE PERMITA EL ESCANEO DE LAS VULNERABILIDADES EN LOS DISPOSITIVOS MÓVILES ANDROID PARA MITIGAR LOS PROBLEMAS DE SEGURIDAD", de responsabilidad del Ing. Henry Mauricio Villa Yánez, ha sido prolijamente revisada y se autoriza su presentación.

**Tribunal:**

| | |
|---|---|
| Ing. Oswaldo Geovanny Martínez Guahima | |
| **PRESIDENTE** | FIRMA |
| Ing. Jorge Ernesto Huilca Palacios | |
| **DIRECTOR** | FIRMA |
| Ing. Gloria de Lourdes Arcos Medina | |
| **MIEMBRO** | FIRMA |
| Ing. Julio Roberto Santillán Castillo | |
| **MIEMBRO** | FIRMA |
| | |
| **DOCUMENTALISTA SISBIB ESPOCH** | FIRMA |

**Riobamba, Marzo 2016**

## DERECHOS INTELECTUALES

Yo, Henry Mauricio Villa Yánez, declaro que soy responsable de las ideas y resultados expuestos en la presente Trabajo de Investigación, y que el patrimonio intelectual generado por la misma pertenece exclusivamente a la Escuela Superior Politécnica de Chimborazo.

060392349-1

## DEDICATORIA

La fe, el esfuerzo y optimismo dedicado a lo largo de los años de estudio, son el fruto de quienes creyeron en nosotros, apoyándonos en todo sentido extendiendo la mano a través de la educación.

El presente trabajo de investigación está dedicado a mis queridos padres, quienes siempre han estado en los momentos más difíciles apoyándonos y dándonos ánimos cuando más lo necesitamos.

Henry

## A G R A D E C I M I E N T O

Expreso mi más sincero agradecimiento, primero a Dios por darme la oportunidad de concluir nuestros estudios, a todos y cada uno de mis familiares que con su apoyo diario me ha permitido el cumplimiento otro objetivo en la vida, a todos nuestros maestros que cada día con sus enseñanzas y experiencias nos han encaminado hacia un futuro lleno de oportunidades.

Henry

## ÍNDICE DE CONTENIDO

## LISTA DE TABLAS

## ÍNDICE DE GRÁFICOS

## LISTA DE ANEXOS

# RESUMEN

El trabajo de investigación Desarrollo de una aplicación que permita el escaneo de las vulnerabilidades en los dispositivos móviles Android para mitigar los problemas de seguridad realizado en la ciudad de Riobamba, beneficia principalmente a los desarrolladores de software para la realización de una programación segura bajo Android. El presente trabajo de investigación utiliza el método científico porque utiliza varias etapas para obtener un conocimiento válido, lo que ha permitido la fiabilidad de la investigación y resultados. Se compararon los indicadores considerados en las variables y se aplicó la estadística descriptiva y la inferencial para la demostración de la hipótesis. Las herramientas que se utilizaron fueron como ambiente de desarrollo integrado Netbeans y Eclipse Juno, JDK (Java Development Kit) de Java y el SDK (Software Development Kit) de Android. Se desarrolló el generador y el framework de ejecución de los casos de prueba que permite la detección de las vulnerabilidades en las aplicaciones escaneadas, se elaboró la Guía de Mejores Prácticas para la Realización de una Programación Segura en Android. Se implementó y comparó los resultados obtenidos entre los prototipos I (sin considerar las recomendaciones de la guía) y II (considerando las recomendaciones de la guía) de 3 aplicaciones, que obtuvieron una valoración de 28 y 34 puntos de acuerdo a las escalas de Likert respectivamente. Se concluye que la aplicación desarrollada (APSEBI) escanea las vulnerabilidades basadas en intents, a través de la guía elaborada mejora la seguridad de las aplicaciones Android en un 21,4%. Se recomienda a los desarrolladores de software seguir actualizando el generador de casos de prueba en base a nuevos patrones de vulnerabilidad, así como también la guía elaborada.

Palabras clave: <VULNERABILIDADES ANDROID>, <SISTEMA OPERATIVO ANDROID>, <SEGURIDAD BASADA EN INTENTS>, <PROGRAMACIÓN SEGURA>, <RIOBAMBA [Ciudad]>, <PATRONES DE VULNERABILIDAD ANDROID>

## ABSTRACT

The research development of an application that allows scanning vulnerabilities in Android mobile devices to mitigate security issues held in the city of Riobamba, mainly benefits developers of software for the realization of a secure programming on Android. This research uses the scientific method because it uses several steps to obtain valid knowledge, allowing the reliability of the research and results. Comparing: the indicators considered in the variables and applied, descriptive and inferential statistics to demonstrate the hypothesis. The tool used were as integrated development environment, Netbeans and Eclipse Juno, JDK (Java Development Kit) Java and SDK (Software Development Kit) for Android. Developed, the generator and the framework for implementing the test cases that allows the detection of vulnerabilities in scanned applications, and it was developed the Guide to Best Practices for Making a Secure Programming in Android. It was implemented and compared the results obtained between the prototypes I (without considering the recommendations of the Guide) and II (considering the recommendations of the Guide) from 3 applications, which received a rating of 28 and 34 points according to Likert scales respectively. It is concluded that the developed application (APSEBI) scans based vulnerabilities intents, through the prepared Guide improves the security of Android applications by 21.4%. It is recommended, software developers continue to update the generator of test cases based on new patterns of vulnerability, as well as the prepared guide.



Keywords: <VULNERABILITIES ANDROID>, <ANDROID OPERATING SYSTEM> , <SECURITY BASED INTENTS> , <SECURE PROGRAMMING> , <RIOBAMBA [City]> , <PATTERNS OF VULNERABILITY ANDROID>

# CAPÍTULO I

## INTRODUCCIÓN

En este presente trabajo de investigación "Desarrollo de una aplicación que permita el escaneo de las vulnerabilidades en los dispositivos móviles Android para mitigar los problemas de seguridad", se plantea el desarrollo de una aplicación que permita el escaneo de las vulnerabilidades basadas en intents en dispositivos Android, además de la creación de una Guía de mejores prácticas para la realización de una programación segura bajo el sistema operativo Android, con el objetivo de mejorar la seguridad en las aplicaciones escaneadas y beneficiar principalmente a los desarrolladores de software para la creación de aplicaciones más seguras para Android.

El trabajo de investigación que se plantea es de tipo experimental ya que se fundamenta en pruebas realizadas en escenarios de laboratorio (prototipos), además por la naturaleza de la investigación se utilizará el método científico para obtener un conocimiento válido desde el punto de vista científico.

Se utilizará dentro de la investigación como técnicas: la búsqueda de información, pruebas, observación, análisis, estadística descriptiva. Como instrumentos se utilizará el software Netbeans, Eclipse y el SDK de Java que permitirán el desarrollo de las aplicaciones planteadas.

Para la validación de la hipótesis planteada se utilizará la estadística descriptiva y la inferencial.

La estructura del presente trabajo de investigación, mediante el uso de capítulos, se muestra a continuación:

- En el **CAPÍTULO I INTRODUCCIÓN**. Se detallará el problema de la investigación, la justificación de la investigación, los objetivos a alcanzar con el desarrollo de la misma y el planteamiento de la hipótesis que se desea demostrar. Determinando un enfoque u orientación general de la investigación, además precisar la delimitación del problema, circunscribiéndolo a aspectos definidos y dejando claramente establecidos aquellos que escapan a su alcance, para la elaboración del sistema teórico.

- En el **CAPÍTULO II MARCO DE REFERENCIA**. Se profundizará el estado del arte de la investigación planteada

- En el **CAPÍTULO III DISEÑO DE LA INVESTIGACIÓN**. se detallará el tipo de investigación el diseño, los métodos y técnicas utilizadas, los instrumentos, con su respectiva validación, que se van a usar dentro del trabajo de investigación.

- En el **CAPÍTULO IV RESULTADOS Y DISCUSIÓN**. Se desarrollan las pruebas en los escenarios establecidos, se analizan, comparan los resultados obtenidos y se demuestra la hipótesis definida.

- En las **CONCLUSIONES y RECOMENDACIONES**. Se enuncian los resultados obtenidos y las sugerencias luego de realizar la investigación.

- En la **BIBLIOGRAFÍA**. Se enlistan las referencias utilizadas para el desarrollo de la investigación.

- En los **ANEXOS**. Se adjunta la información adicional utilizada.

## 1.1. Problema de investigación

### 1.1.1. Planteamiento del problema

En la actualidad los dispositivos móviles (smartphones, tabletas), desempeñan un papel importante en la vida cotidiana, debido a las diversas funcionalidades que prestan, como, por ejemplo: aplicaciones de mensajería, banca virtual, video llamadas, ofimática entre otras.

Hoy en día Android es el sistema operativo más usado en todo el mundo en dispositivos móviles (**STATISTA**, 2015), convirtiéndose directamente en un blanco para los atacantes.

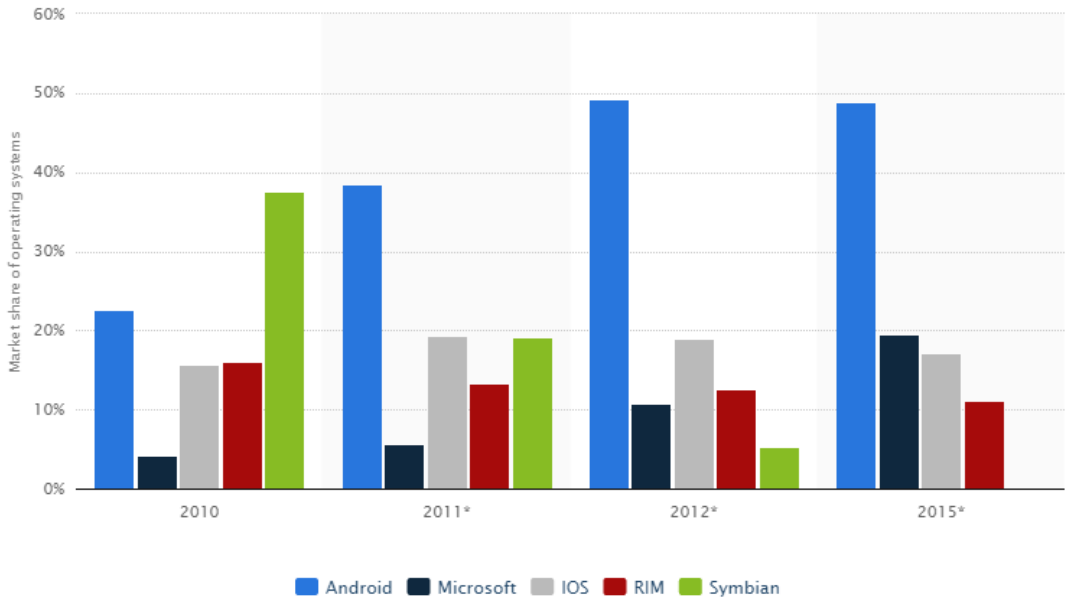



**Gráfico 1-1** Sistema operativos de smartphones populares 2010-2015

**Fuente:** http://www.statista.com/statistics/266970/market-share-forecast-of-smartphone-operating-systems-from-2010-to-2015/

Android es un sistema operativo open source, por lo tanto, cualquier persona con conocimientos de programación puede desarrollar una aplicación y embeber código malicioso dentro de ellas, tal es el caso de virus, spyware, worms, entre otros. Al ejecutar estas aplicaciones como resultado se tiene fallas en el sistema, pérdida y acceso a datos sensibles de los usuarios. (**DHAYA, DEPT. OF CSE, & POONGODI**, 2014).

Además, Android brinda la flexibilidad de personalizar los teléfonos inteligentes por cada uno de sus propietarios, mediante la instalación de un sinnúmero de aplicaciones que están disponibles en el Play Store, y en otro de los casos el usuario instala una APK (Application Package File) que encuentra en la web, en cualquier de los dos casos se corre el riesgo de ser víctima de ataques. (**CHIN, FELT, GREENWOOD, & WAGNER**, 2011).

Debido a las facilidades que presta este sistema operativo, también incrementa los riesgos de creación de vulnerabilidades que perjudiquen la seguridad de la información que maneja. En Android además existen diferentes tipos de vulnerabilidades considerándose a estos puntos débiles del software que permiten que un atacante comprometa la integridad, disponibilidad o confidencialidad del dispositivo. Algunas de las vulnerabilidades más severas permiten que los atacantes ejecuten código arbitrario, denominadas vulnerabilidades de seguridad, en un sistema comprometido. (**MICROSOFT**, 2015).

En el sistema operativo Android las vulnerabilidades detectadas y documentadas son relativamente pocas, en sí Android sufre más ataques de virus o malware que están embebidas en las aplicaciones instaladas en el dispositivo móvil, por tal motivo es muy importante el aseguramiento de las mismas.

Android tratando de asegurar la información, ha considerado algunos mecanismos para este fin, como, por ejemplo:

- Android permite el aislamiento de las aplicaciones al momento de su ejecución.
- Se puede establecer los permisos en la ejecución de cada aplicación en Android. (**SALVA & ZAFIMIHARISOA**, 2015)

A continuación, se muestra la manera en la cual interactúan las aplicaciones dentro del sistema operativo Android.

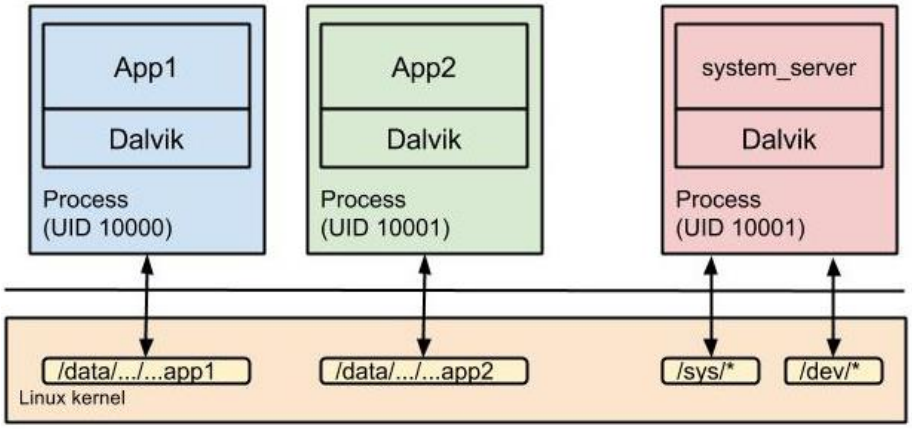



**Gráfico 2-1** Funcionamiento aplicaciones Android

**Fuente:** http://androideity.com/2011/10/17/trabajando-con-intents-en-android-overview/

Como podemos notar en el gráfico las aplicaciones que se ejecutan de manera aislada logran comunicarse utilizando un sistema de mensajería de Android denominado ***intent;*** que permite que una aplicación tenga acceso a contactos, GPS, Wifi, y demás recursos del dispositivo y de otras aplicaciones.

Se puede monitorear este sistema de mensajería (intents) usado en las comunicaciones entre aplicaciones (ataque de hombre en el medio), de esta forma se puede obtener la información que viaja entre aplicaciones, además que se puede insertar datos corruptos en esta comunicación. Por tal motivo es muy importante el aseguramiento de las aplicaciones por este medio.

Actualmente se ha realizado varias investigaciones previas a cerca del tema en cuestión, entre ellos se puede describir:

El trabajo de investigación "*Detecting software vulnerabilities in android using static analysis"* **(DHAYA, DEPT. OF CSE, & POONGODI**, 2014) trata sobre la detección de vulnerabilidades en Android, se fundamenta principalmente en un análisis estático de las aplicaciones instaladas en el dispositivo Android. Este análisis estático se basa en un modelo probabilístico de Markov (N-gram), sin embargo, en este trabajo de investigación se contempla pocas vulnerabilidades, además que no se puede aumentar la cantidad de vulnerabilidades que analiza.

Según el trabajo de investigación "*Android SMS Malware: Vulnerability and Mitigation"* (**HAMANDI, BEIRUT, CHEHAB, ELHAJJ, & KAYSSI**, 2013) que trata sobre las vulnerabilidades que se pueden explotar en los mensajes de texto (SMS) utilizando malware. Para este fin se desarrolla una aplicación (malware) que intercepten la comunicación con el API de los SMS (uso de intents), sin embargo, el trabajo de investigación se centra sólo en la explotación de la vulnerabilidad en los SMS, mientras que otras vulnerabilidades no toma atención.

El trabajo de investigación "*Vulnerability Detection of Android System in Fuzzing Cloud"* (**WU, WU, YANG, WU, & WANG**, 2013) versa sobre el análisis de las vulnerabilidades de las aplicaciones Android usando Fuzzing Cloud, que es un software de testing en la nube que descubre errores en el código, vulnerabilidades en software, analizando capa por capa, sin embargo se debe tener el código fuente de la aplicación, cargar a la nube y esperar el análisis, el inconveniente aparece cuando se quiere analizar vulnerabilidades de aplicaciones que no se dispone del código fuente, además que se necesita de una conexión a internet y no se puede personalizar el análisis ya que es una plataforma de código cerrado.

En los trabajos de investigación revisados anteriormente no se ha podido establecer un análisis de vulnerabilidades que sea flexible y que permita analizar la comunicación entre los sandbox, por tal motivo el presente trabajo de investigación se centra en un modelo de análisis de vulnerabilidades con las características mencionadas.

### 1.1.2. Formulación del problema

¿Cuál sería el nivel de mejora en la seguridad en dispositivos móviles Android al implementar una aplicación que escanee las vulnerabilidades en este sistema operativo?

### 1.1.3. Sistematización del problema

- ¿Cuáles son las causas más comunes que crean vulnerabilidades en dispositivos Android?
- ¿Cuáles son las fortalezas que contendrá la aplicación a ser desarrollada?

- ¿Cuáles son las ventajas y desventajas que conlleva el desarrollo de una aplicación que escanee las vulnerabilidades en el dispositivo móvil Android?
- ¿Cuáles son las alternativas existentes en la actualidad que realicen el escaneo de vulnerabilidades en el dispositivo Android?

## 1.2. Justificación de la investigación

### 1.2.1. Justificación teórica

El uso del sistema operativo Android tiene muchas ventajas entre las cuales podemos citar las siguientes:

- El sistema operativo Android puede instalarse en cualquier dispositivo.
- Es de código abierto, lo que permite la adaptación y mejora del sistema operativo.
- Se puede personalizar casi cualquier cosa dentro del sistema operativo.
- Brinda la facilidad de que las aplicaciones sean multitarea.
- Existe además de soporte oficial, comunidades que dan soporte a Android.

El sistema operativo Android por defecto asegura las aplicaciones ejecutándoles dentro de un entorno aislado, y para la comunicación entre ellas hace uso de un sistema de mensajería (intents).

Se ocupará el lenguaje ioSTS (input output Symbolic Transition Systems) para la creación de los patrones de vulnerabilidades, lo que permitirá una mejor estructura del patrón siguiendo la lógica de los autómatas.

El presente trabajo de investigación tiene como finalidad la realización de una aplicación que permita determinar las vulnerabilidades basadas en intents en aplicaciones Android, con el objetivo de asegurar la comunicación entre los diferentes sandbox en los que se ejecutan las aplicaciones y por consiguiente asegurar los datos sensibles que se

transmiten. Además, la aplicación será lo suficientemente flexible para crear patrones que ayuden a la detección de otras vulnerabilidades.

### 1.2.2. Justificación práctica

El presente trabajo de investigación ayudará a determinar y mitigar las vulnerabilidades que se pueden hallar en las aplicaciones Android, para este objetivo se analizará los intents que ocupa la aplicación analizada. Este análisis permitirá dar a conocer al usuario las aplicaciones que son un eminente riesgo de seguridad mediante un reporte que la aplicación generará.

Para tal caso se modificará y adaptará un trabajo de investigación realizado anteriormente para de esta manera poder analizar las aplicaciones de Android y determinar el riesgo que representan las aplicaciones en el dispositivo móvil.

La adaptación consistirá en revisar el código fuente y ajustarlo a la estructura de Android. La modificación consistirá en aumentar la cantidad de patrones, usando el lenguaje autómata ioSTS, para determinar las vulnerabilidades en las aplicaciones móviles.

La seguridad de las aplicaciones se medirá de acuerdo a escalas en los que se determina si una aplicación es vulnerable o no.

Además, después de determinar las vulnerabilidades se realizarán recomendaciones que ayuden a mejorar las prácticas de programación con el fin de mitigarlas, para esto se realizará una prueba en la que se considere las recomendaciones dadas y observar la reducción de vulnerabilidades, y así asegurar de mejor manera los datos que éstas manejan.

La aplicación se sujetará a pruebas en base a diferentes criterios que permitirán la confiabilidad de la misma.

## 1.3. Objetivos

### 1.3.1. General

Desarrollar una aplicación que permita el escaneo de las vulnerabilidades en los dispositivos móviles en Android para mitigar los problemas de seguridad.

### 1.3.2. Específicos

- Analizar las vulnerabilidades de seguridad que se producen en el uso de aplicaciones en Android.
- Desarrollar una aplicación en Android, para analizar el dispositivo móvil y determinar que aplicaciones son de riesgo.
- Evaluar la aplicación desarrollada en los escenarios de pruebas establecidos.
- Crear una guía de las mejores prácticas de desarrollo en Android con el fin de mitigar las vulnerabilidades detectadas.

## 1.4. Hipótesis

La propuesta de aplicación para dispositivos móviles Android contribuirá a la detección de vulnerabilidades para mejorar la seguridad en las aplicaciones escaneadas

# CAPÍTULO II

## MARCO DE REFERENCIA

En este capítulo, se profundizan el estado del arte en el que se encuentra el tema a ser investigado, se tratará acerca de la arquitectura de Android, análisis de vulnerabilidades y el uso de patrones usando ioTS.

### 2.1. Estado del arte

Actualmente se han realizado investigaciones relacionadas con las ciencias en estudio, las cuales han aportado al presente trabajo de investigación para:

- Comprender de mejor manera los diferentes tipos de vulnerabilidades que podemos encontrar en el sistema operativo Android.
- Identificar la descripción del problema y los objetivos planteados en las investigaciones realizadas.
- Conocer el proceso desarrollado por los autores para realizar las investigaciones.
- Determinar las métricas, indicadores que los autores utilizan para realizar las pruebas.
- Determinar el aporte que realizan los autores en las investigaciones realizadas.
- Observar los resultados y conclusiones obtenidas en base a las pruebas realizadas que hacen los autores.

Entre las principales investigaciones se mencionan:

**Investigación "Detecting software vulnerabilities in android using static analysis"**
(**DHAYA, DEPT. OF CSE, & POONGODI**, 2014)

La motivación de los autores para la realización del presente artículo científico se produce debido a que Android no tiene la facilidad de detectar por si sólo nuevos virus, ya que no

puede verificar la actualización de firmas en la instalación de aplicaciones, provocando que cualquier persona al desarrollar una aplicación pueda incluir código malicioso(virus, malware, spyware, gusanos, etc.)

En este trabajo de investigación se propone el desarrollo de un sistema que detecte el malware existente en las aplicaciones Android basándose en el análisis de código utilizando el algoritmo de aprendizaje automático denominado N-gram, detectando las características maliciosas o vulnerables en las aplicaciones móviles.

El proceso desarrollado se resume en lo siguiente:

- Análisis de las aplicaciones existentes en el mercado que realicen el escaneo basado en firmas.
- Planteamiento de los objetivos de la aplicación propuesta
- Creación del diagrama de arquitectura de la aplicación propuesta
- Desarrollo e implementación de la aplicación
- Análisis de los resultados obtenidos
- Definición de conclusiones.

La propuesta de la arquitectura de la aplicación propuesta se muestra a continuación:

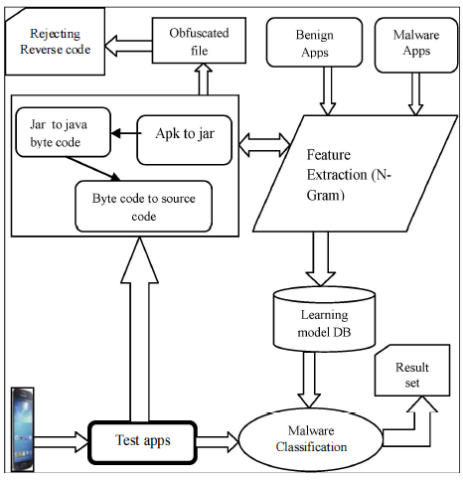



**Gráfico 1-2** Arquitectura N-gram de la aplicación propuesta

**Fuente:** DHAYA, DEPT. OF CSE, & POONGODI, 2014

En el proceso de la creación de la aplicación se ha considerado el desarrollo de cuatro módulos:

- Automatización del proceso de ingeniería inversa. Proceso en el cual se convierte a un código analizable el archivo apk de una aplicación.

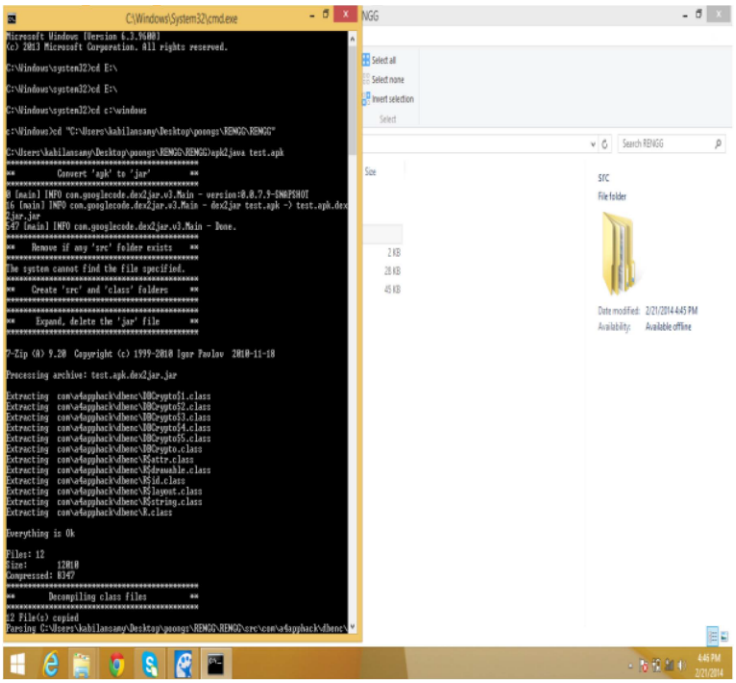


**Gráfico 2-2** Automatización del proceso de ingeniería inversa

**Fuente:** DHAYA, DEPT. OF CSE, & POONGODI, 2014

- Establecimiento de las características iniciales para el reconocimiento de malware: se establece las características más destacables de las diferentes vulnerabilidades provocadas por el malware.
- Construcción del archivo de base de datos y clasificación del malware: se crea un archivo de base de datos en la cual se establece los tipos de malware

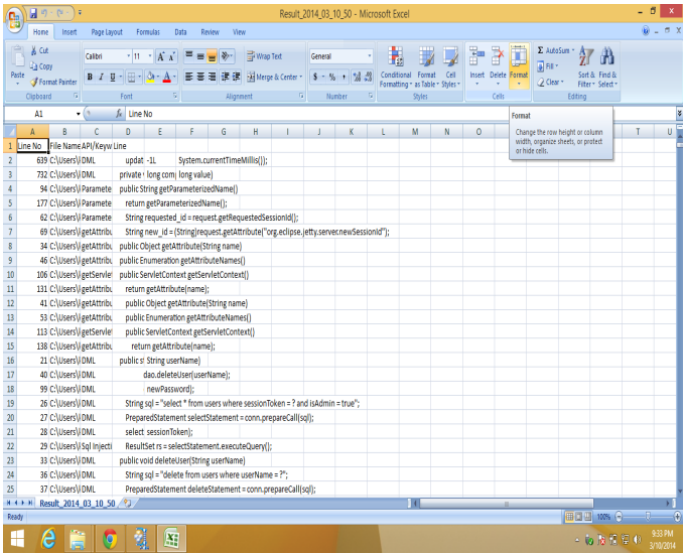


**Gráfico 3-2** Construcción del archivo de base de datos y clasificación del malware

**Fuente:** DHAYA, DEPT. OF CSE, & POONGODI, 2014

- Evaluación del rendimiento de la aplicación: se encuentran las vulnerabilidades en Android mediante el uso de CVSS (Common Vulnerability Scoring System) y posteriormente se analiza y se toma acciones correctivas de las mismas.

De acuerdo a los resultados obtenidos, se concluye que mediante el uso del método de detección basado en firmas brinda una protección cuando la amenaza ha sido identificada con anticipación, este sistema no permite encontrar nuevo malware, debido a que se usa un análisis estático del código de la aplicación.

Sin embargo, la investigación realizada no incluye el proceso de determinación de nuevas vulnerabilidades, además no contiene el aporte de los autores para investigaciones futuras.

**Investigación "Android SMS Malware: Vulnerability and Mitigation" (HAMANDI, BEIRUT, CHEHAB, ELHAJJ, & KAYSSI**, 2013)

La motivación de los autores para la realización del artículo científico se produce debido a que Android permite que varias operadoras de telefonía móvil en todo el mundo permiten el abuso (en privilegios) del sistema de SMS, por ejemplo: transferir crédito

entre usuarios, entre otras convirtiéndose en un hueco de seguridad. Dentro este artículo científico

En este trabajo de investigación se analiza cómo se utiliza dichos procedimientos para instalar malware, además cuales serían las posibles soluciones para la mitigación de dichas vulnerabilidades.

El proceso desarrollado se resume en lo siguiente:

- Análisis del funcionamiento y estructura del sistema operativo Android
- Diseño de la aplicación a desarrollarse
- Análisis de las pruebas realizadas

La propuesta del diseño de la aplicación se muestra a continuación:


**Gráfico 4-2** Propuesta del diseño de la aplicación

**Fuente:** HAMANDI, BEIRUT, CHEHAB, ELHAJJ, & KAYSSI, 2013

- *Main Activity:* se refiere a la interfaz gráfica para leer y enviar los mensajes SMS.
- *Listen Service:* es el componente que escucha la llegada de los mensajes SMS y toma las acciones dependiendo a criterios predefinidos.
- *Sender Service:* este componente maneja el envío no autorizado de la aplicación SMS.
- *Boot Service:* Este componente es necesario para correr los principales servicios en el sistema operativo de Android.

En el proceso de realización de las pruebas se determina que el uso de la los mensajes SMS en Android al momento de realizar transacciones financieras se convierte en un fallo potencial de seguridad ya que mediante el malware se puede corromper e infectar el dispositivo móvil, como soluciones se plantea:

- El uso del servicio llamada “Virus Total” que escanea de manera online las URLs y archivos del dispositivo móvil.

- No permitir que las aplicaciones tengan el control total en el envío de mensajes SMS.
- No confiar en aplicaciones que únicamente su actividad sea el envío de mensajes SMS.
- Se debe establecer por parte de los usuarios permisos específicos para aplicaciones de mensajería SMS, no únicamente conformarse con los permisos de instalación.

De acuerdo a los resultados obtenidos, se concluye que es un riesgo potencial la vulnerabilidad referente a la mensajería SMS, y que el desarrollo de aplicaciones que exploten esta vulnerabilidad pueden ser publicadas fácilmente en el Play Store, por tanto se propone varias soluciones para mitigar dicha vulnerabilidad ya sea en el desarrollo de aplicaciones de mensajería SMS y para el usuario.

Sin embargo, la investigación realizada no incluye el proceso de determinación de nuevas vulnerabilidades, además no contiene el aporte de los autores para investigaciones futuras.

**Investigación "Vulnerability Detection of Android System in Fuzzing Cloud"** (**WU, WU, YANG, WU, & WANG**, 2013)

La motivación de los autores para la realización del artículo científico se debe a que la mayoría de los modelos de análisis de vulnerabilidades se centran en ciertos aspectos, por ejemplo, aplicaciones, permisos o fugas capacidad y en su mayoría de veces provocan goteo de información privada, niegan el servicio, provocan costos potenciales, entre otras desventajas al momento de analizar las aplicaciones.

En este trabajo de investigación se analiza de las vulnerabilidades de las aplicaciones Android usando el método Fuzzing Cloud, que es un software de testing en la nube que descubre errores en el código, vulnerabilidades en software, analizando capa por capa de las aplicaciones Android.

El proceso desarrollado se resume en lo siguiente:

- Diseño del método Fuzzing Cloud
- Adaptación de Fuzzing Cloud al sistema operativo Android
- Implementación de Fuzzing Cloud
- Análisis y evaluación de las pruebas realizadas

Se muestra la arquitectura de Fuzzing Cloud que analiza de una manera flexible las aplicaciones Android.

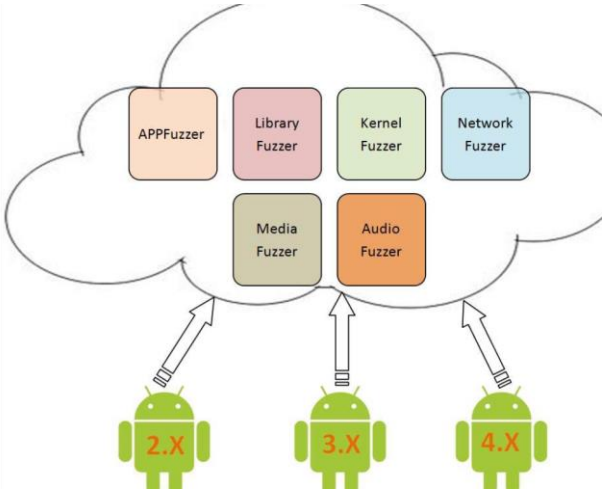



**Gráfico 5-2** Arquitectura de Fuzzing Cloud

**Fuente:** WU, WU, YANG, WU, & WANG, 2013

Como se puede notar cada uno de los *fuzzers* (conjunto de hardware y software para el análisis) permiten la correcta distribución de los procesos para el análisis de las aplicaciones. Para la determinación del número de pruebas para cada aplicación se utiliza un algoritmo, el mismo que se basa en la naturaleza de la aplicación y el número de clases que contiene la misma.

**Algorithm 1** Fuzzing cloud algorithm

```
1: procedure FUZZINGCLOUD(res, mods)                ▷ build fuzzers
2:     fc ← set()
3:     for all m ∈ mods do                          ▷ initialization
4:         m.cpu ← res.cpu                          ▷ CPU assignment
5:         m.disk ← res.disk                        ▷ storage assignment
6:         m.fuzzerinit()                           ▷ initialize fuzzer
7:         fc.add(m)                                ▷ add to cloud
8:     end for
9:     Thread(adjust(fc)).start()                   ▷ dynamic adjustment
10:    return fc                                    ▷ final fuzzers set
11: end procedure
```

**Gráfico 6-2** Algoritmo de Fuzzing Cloud

**Fuente:** WU, WU, YANG, WU, & WANG, 2013

El prototipo de fuzzing cloud ha sido desplegado en un servidor virtualizado con Xen para las pruebas realizadas, de los cuales se puede determinar que al momento de realizar

un análisis de aplicaciones multimedia se produce un error al momento de querer analizar dichas aplicaciones. Pero para el otro tipo de aplicaciones no ha existido mayor novedad al respecto.

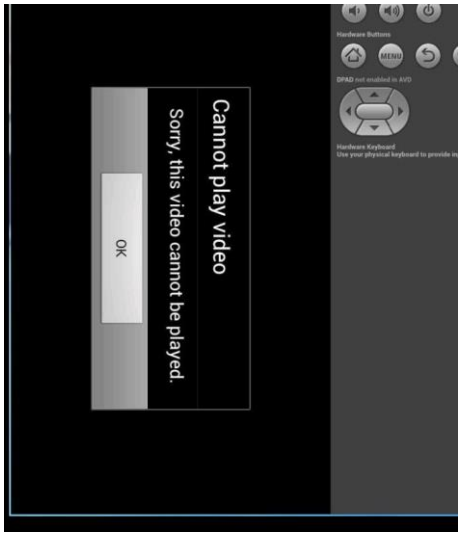



**Gráfico 7-2** Resultado de escaneo de una aplicación multimedia

**Fuente:** WU, WU, YANG, WU, & WANG, 2013

De acuerdo a los resultados obtenidos, se concluye que el prototipo denominado Fuzzing Cloud detecta vulnerabilidades en Android, para el despliegue completo de dicha aplicación se necesita mucho almacenamiento y una gran capacidad de procesamiento; que la arquitectura propuesta es lo suficientemente flexible y escalable.

Sin embargo se debe tener el código fuente de la aplicación, cargar a la nube y esperar el análisis, el inconveniente aparece cuando se quiere analizar las vulnerabilidades de aplicaciones que no se dispone del código fuente, además que se necesita de una conexión a internet y no se puede personalizar el análisis ya que es una plataforma de código cerrado.

## 2.2. Android

### 2.2.1. Antecedentes

Android es un sistema operativo de código abierto para dispositivos móviles que fueron creadas por Google y Open Handset Alliance. La idea de Android se resume en: "Android - La imaginación es el límite".

Android está basado en el kernel de Linux, caracterizándose principalmente por ser un sistema operativo libre, gratuito y multiplataforma. Además, permite programar aplicaciones en una variación de la máquina virtual que usa Java cuyo nombre es *Dalvik* diseñado específicamente para asegurar que la multitarea se ejecute de manera eficiente en un único dispositivo. Android proporciona todas las interfaces necesarias para desarrollar aplicaciones que accedan a las funciones del teléfono (como el GPS, las llamadas, la agenda, etc.) de una forma muy sencilla en un lenguaje de programación muy conocido como es Java. (**XATAKANDROID**, 2015)

En la actualidad Android es el sistema operativo que utilizan más de mil millones de smartphones y tabletas. Actualmente el 47.45% del total de dispositivos en el mundo tienen instalado el sistema operativo Android, viéndose reflejado el esfuerzo de Google para brindar mayores prestaciones y una mejor experiencia para el usuario.

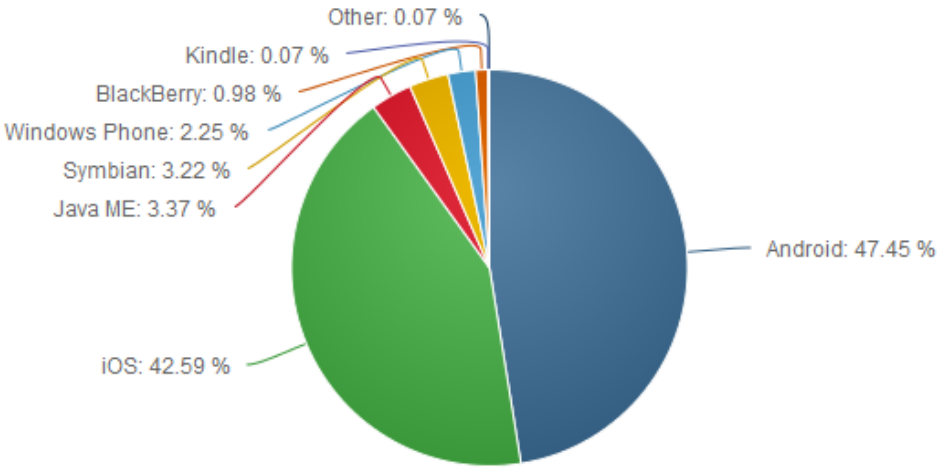



**Gráfico 8-2** Sistemas operativos para dispositivos móviles

**Fuente:** https://blog.uchceu.es/informatica/ranking-de-sistemas-operativos-mas-usados-para-2015/

Los dispositivos móviles hoy en día ofrecen al usuario, en un mismo y reducido aparato, funciones de comunicación y procesamiento de datos que van mucho más allá de las simples llamadas telefónicas o la ejecución de aplicaciones básicas. Android busca ser una firme alternativa a otros sistemas, ya ampliamente extendidos, como IOS de Apple o Windows Phone de Microsoft.

Todas las versiones que Android posee empezaron con el lanzamiento de Android beta (versión de prueba) en noviembre de 2007, y la primera versión comercial fue Android 1.0 la misma que fue lanzada en septiembre de 2008.

Las actualizaciones que se han venido realizando en el transcurso del tiempo han sido básicamente para corregir fallos de las versiones anteriores y además han aprovechado para agregan nuevas funcionalidades al sistema operativo.

A partir de abril de 2009, las versiones de Android han sido desarrolladas bajo un nombre en clave y sus nombres siguen un orden alfabético:

- Apple Pie,
- Banana Bread,
- Cupcake, Donut,
- Éclair, Froyo,
- Gingerbread,
- Honeycomb,
- Ice Cream Sandwich,
- Jelly Bean, KitKat (el nombre de una chocolatina de Nestlé) y
- Lollipop 5.1

La última versión de Android fue anunciada en mayo de 2015, en el evento I/O de Google se trata de Android M.


**Gráfico 9-2** Versiones de Android

**Fuente:** http://eduardotortosamartin.com/todas-las-versiones-de-android/

### 2.2.2. Características

Entre las principales características que posee Android podemos citar las siguientes:

- Adaptabilidad para pantallas de alta resolución y compatibilidad con las librerías OpenGL y gráficos 3D
- Permite el almacenamiento de datos estructurados, mediante el uso de una base de datos llamada SQLite.
- Soporta una variedad de tecnologías de conectividad como, por ejemplo: GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, HSDPA, HSPA+, NFC y WiMAX.GPRS, UMTS y HSDPA+.
- Además de SMS y MMS como métodos de mensajería instantánea de texto se incorpora C2DM (Android Cloud to Device Messaging Framework).
- Android soporta cámaras de fotos, de vídeo, pantallas táctiles, GPS, acelerómetros, giroscopios, magnetómetros, sensores de proximidad y de presión, sensores de luz, gamepad, termómetro, aceleración por GPU 2D y 3D.
- Posee un catálogo organizado de aplicaciones gratuitas o de pago dentro de Google Play, las mismas que pueden ser instaladas en el dispositivo móvil o computadora.
- Soporte nativo para pantallas capacitivas con soporte multi-táctil.
- Permite la reutilización y el reemplazo de los componentes brindando flexibilidad en el desarrollo de aplicaciones.

- La máquina virtual de Android (Dalvik) ha sido optimizada especialmente para dispositivos móviles.
- Compatibilidad y flexibilidad en el manejo de archivos multimedia (audio, video e imagen) mediante el uso de códecs especializados para dispositivos móviles.
- Soporte para streaming.
- Entorno de desarrollo muy completo, debido a que, incluyendo un emulador, herramientas de depuración, de memoria, perfiles de rendimiento, y compatibilidad con diferentes IDEs como son: Eclipse, Netbeans, entre otros. Siendo Android Studio el IDE oficial para el desarrollo de aplicaciones en Android.

### 2.2.3. Arquitectura

Android es una plataforma para dispositivos móviles que contiene un stack de software donde se incluye un sistema operativo, middleware y aplicaciones básicas para el usuario.

A continuación, se muestra una visión global por capas de cuál es la arquitectura empleada en Android. Cada una de estas capas utiliza servicios ofrecidos por las anteriores, y ofrece a su vez los suyos propios a las capas de niveles superiores. (**PASCAL & CHRISTOPHE**, 2012)

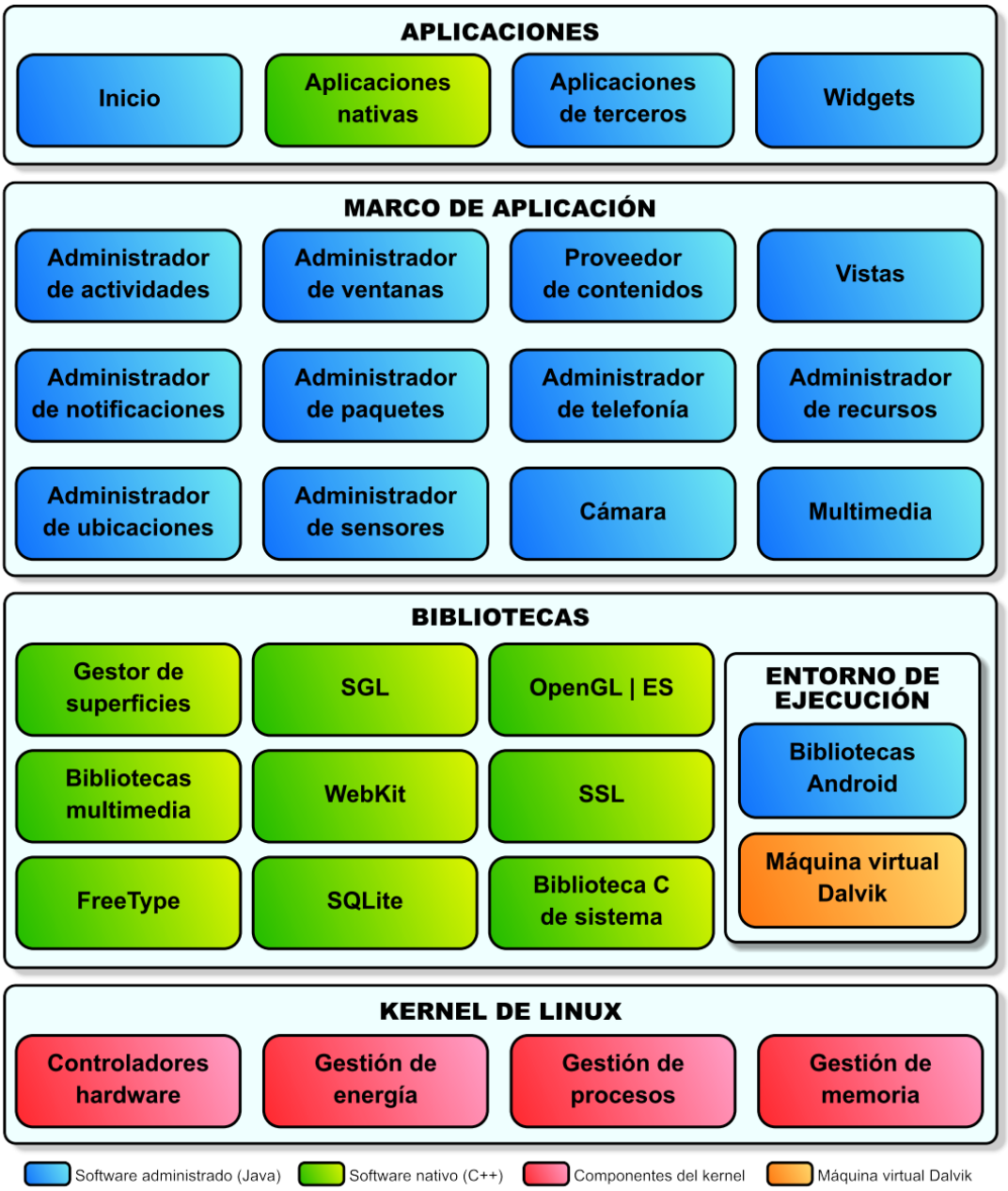



**Gráfico 10-2** Arquitectura de Android

**Fuente:** https://columna80.wordpress.com/2011/02/17/arquitectura-de-android/

- ***Aplicaciones:*** Este nivel contiene, tanto las incluidas por defecto de Android como aquellas que el usuario vaya añadiendo posteriormente, ya sean de terceras empresas o de su propio desarrollo. Todas estas aplicaciones utilizan los servicios, las API y librerías de los niveles anteriores.
- ***Framework de Aplicaciones:*** Representa fundamentalmente el conjunto de herramientas de desarrollo de cualquier aplicación. Toda aplicación que se desarrolle para Android, ya sean las propias del dispositivo, las desarrolladas por Google o

terceras compañías, o incluso las que el propio usuario cree, utilizan el mismo conjunto de API y el mismo "framework", representado por este nivel.

- ***Librerías:*** La siguiente capa se corresponde con las librerías utilizadas por Android. Éstas han sido escritas utilizando C/C++ y proporcionan a Android la mayor parte de sus capacidades más características. Junto al núcleo basado en Linux, estas librerías constituyen el corazón de Android.
- ***Entorno de ejecución de Android:*** Al mismo nivel que las librerías de Android se sitúa el entorno de ejecución. Éste lo constituyen las Core Libraries, que son librerías con multitud de clases Java y la máquina virtual Dalvik.
- ***Núcleo Linux:*** Android utiliza el núcleo de Linux 2.6 como una capa de abstracción para el hardware disponible en los dispositivos móviles. Esta capa contiene los drivers necesarios para que cualquier componente hardware pueda ser utilizado mediante las llamadas correspondientes. Siempre que un fabricante incluye un nuevo elemento de hardware, lo primero que se debe realizar para que pueda ser utilizado desde Android es crear las librerías de control o drivers necesarios dentro de este kernel de Linux embebido en el propio Android.

## 2.3. Diseño y desarrollo de aplicaciones en Android

Android por ser un sistema operativo open source permite el acceso a su código fuente, así como a la lista de incidencias, en las cuales se pueden observar los problemas que se han encontrado además de informar de nuevos problemas, por su parte Windows Phone y iOS no permiten el acceso al código fuente por ser sistemas operativos privativos. Esta característica ha permitido el avance y flexibilidad del sistema operativo ayudado con la comunidad de desarrollo.

Al tener acceso al código fuente no significa que se pueda tener siempre la última versión de Android en un determinado móvil, ya que el código para soportar el hardware (drivers) de cada fabricante normalmente no es público.

En un principio, Android era eminentemente un sistema operativo pensado para usar con teclado, y gracias a un cursor poder navegar entre las aplicaciones. Android es un sistema

operativo muy flexible y altamente personalizable por su naturaleza de open source. Y después se convirtió en un sistema operativo eminentemente táctil.
Las aplicaciones se desarrollan habitualmente en el lenguaje Java mediante Android Software Development Kit (Android SDK), en la actualidad se usa Android Studio que es un IDE (Integrated Development Environment) que tiene incluido el SDK de Android. El desarrollo de aplicaciones para Android no requiere aprender lenguajes complejos de programación. Todo lo que se necesita es un conocimiento aceptable de Java y tener instalado y debidamente configurado el SDK de Android y un IDE cualquiera.

Todas las aplicaciones están comprimidas en formato APK, que se pueden instalar sin dificultad desde cualquier explorador de archivos en la mayoría de dispositivos.

### 2.3.1. Entorno de desarrollo Android

Para el desarrollo en Android primero es necesario disponer en nuestro computador el entorno y herramientas necesarias para comenzar empezar a programar.

a) Se debe descargar el JDK (Java Developer Kit) de la página web de Oracle (http://www.oracle.com/technetwork/java/javase/downloads/index.html), instalarlo dentro de nuestra computadora.
b) Descargar e instalar el IDE que para este caso se instalará Android Studio, que es el entorno de desarrollo oficial de Android.
c) Se debe instalar y actualizar los componentes de Android siendo los componentes más importantes:

- Android SDK Tools
- Android SDK Platform-tools
- Android SDK Build-tools (por ahora la versión más reciente)
- Una o más versiones de la plataforma Android
- Android Support Repository (extras)
- Google Play Services (extras)
- Google Repository (extras)

### 2.3.2. Estructura de un proyecto Android

Dentro de la estructura del proyecto se debe distinguir entre lo que es un proyecto y un módulo. Siendo la principal diferencia que un proyecto es una entidad única que engloba a todos los demás elementos, además un proyecto puede incluir varios módulos, que por ejemplo puede representar aplicaciones diferentes, versiones de las aplicaciones, o también puede ser diferentes componentes dentro de un sistema determinado (librerías, aplicación de escritorio, aplicación web, aplicación móvil, etc.)

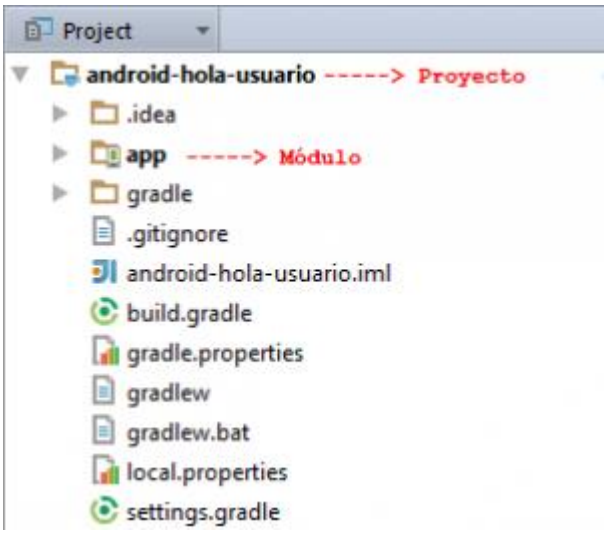



**Gráfico 11-2** Estructura de proyecto Android

**Realizado por:** Villa, H. 2016

Las carpetas más importantes dentro del módulo principal del proyecto son los siguientes:

- Carpeta /app/src/main/java

Esta carpeta contendrá todo el código fuente de la aplicación, como, por ejemplo: clases auxiliares, interfaces, etc. A la clase principal se denomina *MainActitvity* la misma que se encargará de coordinar el funcionamiento correcto de la aplicación.

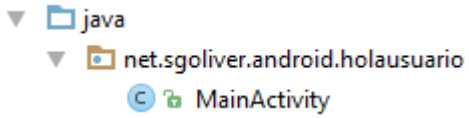



**Gráfico 12-2** Clase principal Android

**Realizado por:** Villa, H. 2016

- Carpeta /app/src/main/res/

Esta carpeta contiene todos los recursos necesarios para el funcionamiento del proyecto, estos pueden ser: imágenes (diferentes tamaños dependiendo si es aplicación móvil, web, escritorio), layouts, cadenas de texto, etc. Todos estos recursos se distribuyen dentro de las siguientes carpetas como se muestra a continuación:

**Tabla 1-2** Estructura recursos Android

| CARPETA | DESCRIPCIÓN |
|---|---|
| **/res/drawable/** | Contiene las imágenes y otros elementos gráficos usados en por la aplicación. Para poder definir diferentes recursos dependiendo de la resolución y densidad de la pantalla del dispositivo se suele dividir en varias subcarpetas:<br>/drawable (recursos independientes de la densidad)<br>/drawable-ldpi (densidad baja)<br>/drawable-mdpi (densidad media)<br>/drawable-hdpi (densidad alta)<br>/drawable-xhdpi (densidad muy alta)<br>/drawable-xxhdpi (densidad muy muy alta) |
| **/res/layout/** | Contiene los ficheros de definición XML de las diferentes pantallas de la interfaz gráfica. Para definir distintos *layouts* dependiendo de la orientación del dispositivo se puede dividir también en subcarpetas:<br>/layout (vertical)<br>/layout-land (horizontal) |
| **/res/anim/**<br>**/res/animator/** | Contienen la definición de las animaciones utilizadas por la aplicación. |
| **/res/color/** | Contiene ficheros XML de definición de colores según estado. |
| **/res/menu/** | Contiene la definición XML de los menús de la aplicación. |
| **/res/xml/** | Contiene otros ficheros XML de datos utilizados por la aplicación. |
| **/res/raw/** | Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyan en el resto de carpetas de recursos. |
| **/res/values/** | Contiene otros ficheros XML de recursos de la aplicación, como por ejemplo cadenas de texto (*strings.xml*), estilos (*styles.xml*), colores (*colors.xml*), arrays de valores (*arrays.xml*), tamaños (*dimens.xml*), etc. |

**Fuente:** http://www.sgoliver.net/blog/estructura-de-un-proyecto-android-android-studio

Todas estas carpetas no son necesarias dentro de todos los proyectos Android ya que depende de la estructura y naturaleza del proyecto. Por ejemplo, se puede tener la siguiente estructura de un proyecto por defecto.

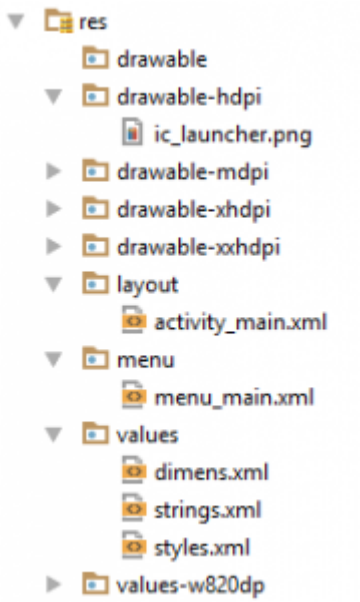



**Gráfico 13-2** Estructura proyecto Android

**Realizado por:** Villa, H. 2016

Además, entre los recursos de un proyecto Android creados por defecto se debe destacar los layouts; al iniciar un proyecto se crea un layout por defecto ***activity_main.xml***, el mismo que contiene la definición de la interfaz gráfica de la pantalla principal de la aplicación. Es mucho más fácil manipular la interfaz gráfica modificando el xml (layout) donde se encuentra definida la pantalla.

Un fichero importante es ***AndroidManifest.xml*** que contiene la definición de varios aspectos fundamentales de la aplicación, como por ejemplo su identificación (nombre, icono,…), sus componentes (pantallas, servicios,…), o los permisos necesarios para su ejecución.

- Carpeta /app/libs

Esta carpeta contiene todas las librerías externas java (.jar) que se utilicen dentro de la aplicación. Las librerías generalmente facilitan diferentes operaciones que ya han sido

optimizadas por otro desarrollador, por ejemplo, el proceso de autenticación de usuarios, operaciones, entre otras.

- Carpeta /app/build/

Esta carpeta alberga un conjunto de archivos que son generados automáticamente cuando se compila el proyecto. Cada vez que se compila el proyecto la máquina virtual de Android, Dalvik, genera varios archivos que entre algunas cosas ayudan al control de uso de recursos por la aplicación, por tal motivo es necesario no realizar cambios en estos archivos a menos que se conozca von exactitud la estructura del archivo.

A continuación, se puede observar la estructura de la carpeta /app/build/

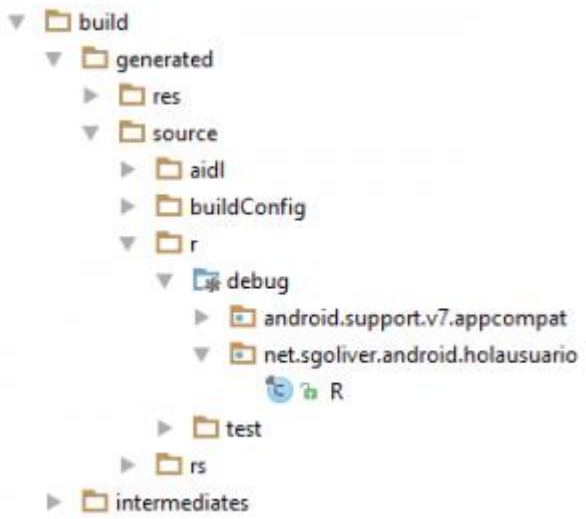



**Gráfico 14-2** Estructura de carpeta /app/build/

**Realizado por:** Villa, H. 2016

Se puede destacar además que dentro de esta carpeta está la clase R.java, que contendrá una serie de constantes con los identificadores (ID) de todos los recursos de la aplicación incluidos los que están en la ruta /app/src/main/res/, de esta forma podemos acceder fácilmente a estos recursos desde nuestro código.

### 2.3.3. Componentes de una aplicación Android

En los lenguajes de programación más conocido como son Java o .NET se utilizan conceptos tales como ventana, control, eventos o servicios para la construcción de una aplicación, en Android se va a disponer de los mismos elementos, pero con diferentes nombres y enfoque. (**HASSAN & NARAYANA**, 2012)

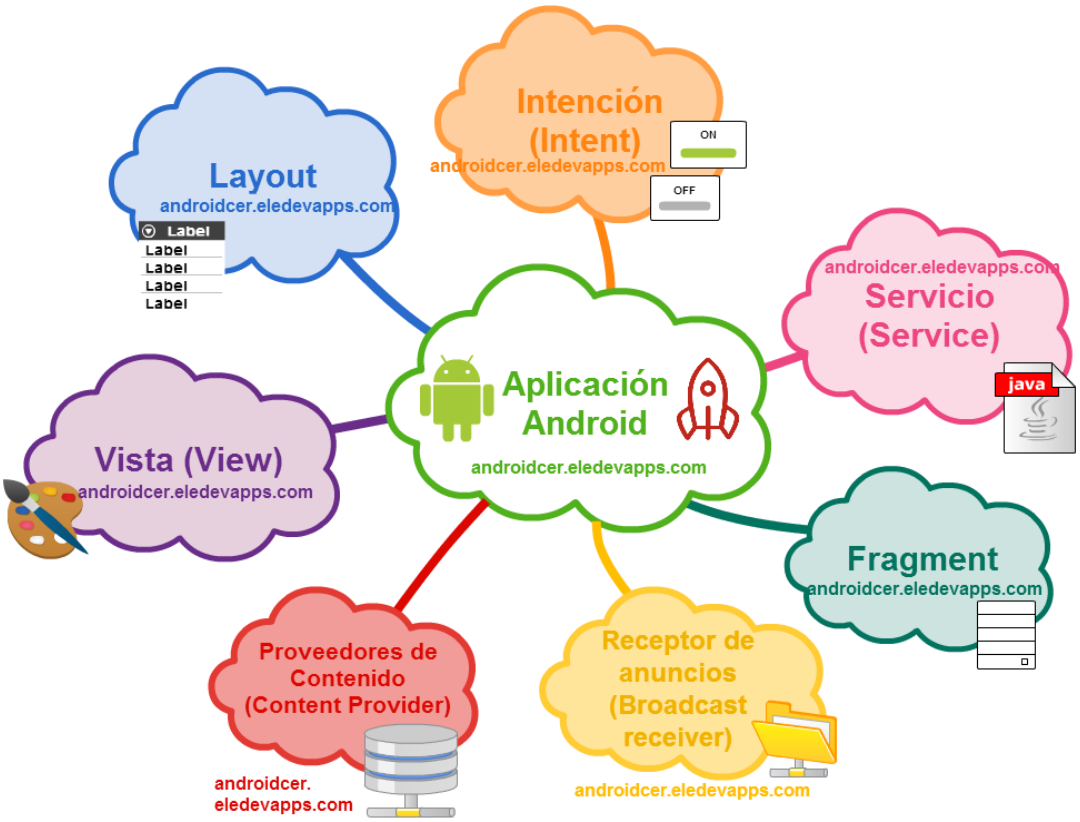



**Gráfico 15-2** Componentes Android

**Fuente:** http://androidcero.eledevapps.com/#sthash.ivZOyEbS.dpuf

Dentro de los componentes de software más importantes que podemos determinar dentro de una aplicación son los siguientes:

***Activity:*** Las actividades (activities) son los componentes más usados al momento de desarrollar una aplicación, estos visualizan las interfaces de usuario usadas para interactuar con la aplicación. Una actividad es iniciada mediante Intents, muestra una pantalla y puede eventualmente retornar una respuesta. Se puede hacer una comparación en la que una activity es un componente similar a una ventana o pantalla en cualquier otro lenguaje de programación visual.

***View:*** Las vistas (view) son los elementos que componen la interfaz gráfica de la aplicación por ejemplo un botón, listas desplegables, cuadros de texto, imágenes, también existe la posibilidad de construir sus propios controles o personalizar los ya existentes. Todas las vistas heredan de la clase View.

***Layout:*** Un layout es la agrupación de vistas en una determinada forma. Se dispone de diferentes tipos de layouts para organizar las vistas de forma lineal, en cuadrícula o indicando la posición absoluta de cada vista. Los layouts también son objetos descendientes de la clase View. Igual que las vistas, los layouts pueden ser definidos en código, aunque la forma habitual de definirlos es utilizando código XML. **(SOURABH, NIKHIL, DEEPAK, & AKASH**, 2013)

***Fragment:*** Cuando salieron las tabletas al mercado conllevó el problema de que las aplicaciones Android debían soportar pantallas más grandes. Por ejemplo, al diseñar una aplicación cualquiera pensada para un dispositivo móvil y ejecutarle en una tableta, el resultado no era el esperado debido a que la pantalla no se ajustaba al tamaño de la tableta. Debido a este problema, desde Android v3.0 aparece el componente denominado fragment. Un fragment está compuesto por la unión de varias vistas (views) para crear un bloque funcional de la interfaz de usuario. Una vez creados los fragments, podemos combinar uno o varios fragments dentro de una actividad, según el tamaño de pantalla disponible.

***Service:*** Los servicios (service) son componentes que no poseen interfaz gráfica y se ejecutan en segundo plano sin necesidad de una interacción con el usuario, son iniciados mediante el uso de intents a través de otros componentes (generalmente actividades). En Android se disponen de dos tipos de servicios: servicios locales, que son ejecutados dentro del mismo proceso y también existen los servicios remotos, que son ejecutados en procesos separados. Los servicios pueden realizar cualquier tipo de acciones, por ejemplo, actualizar datos, lanzar notificaciones, o incluso mostrar elementos visuales. Se puede decir que es algo parecido a un demonio en Linux o un servicio en Windows.

***Content Provider:*** Es el mecanismo estándar que ha definido Android para compartir datos entre aplicaciones sin necesidad de comprometer la seguridad del sistema de ficheros. Mediante estos componentes es posible compartir determinados datos de nuestra

aplicación sin mostrar detalles sobre su almacenamiento interno, su estructura, o su implementación. Los contents provider no son inicializados mediante intents.

***Broadcast Receiver:*** Es componente destinado a detectar y reaccionar ante determinados mensajes o eventos globales (intents de tipo broadcast) generados por el sistema (por ejemplo: Batería baja, SMS recibido, Tarjeta SD insertada) o por otras aplicaciones.

***Widget:*** Son elementos visuales que generalmente son interactivos, estos componentes normalmente se muestran en la pantalla principal del dispositivo móvil.

***Intent:*** Básicamente es un elemento básico que permite la comunicación entre los diferentes componentes que se han descrito anteriormente, representa la voluntad de realizar alguna acción; como realizar una llamada de teléfono, visualizar una página web.

Se utiliza cada vez que queramos:

- Lanzar una actividad
- Lanzar un servicio
- Enviar un anuncio de tipo broadcast
- Comunicarnos con un servicio

Los componentes lanzados pueden ser internos o externos a la aplicación Android. También se utilizará los intents para el intercambio de información entre estos componentes.

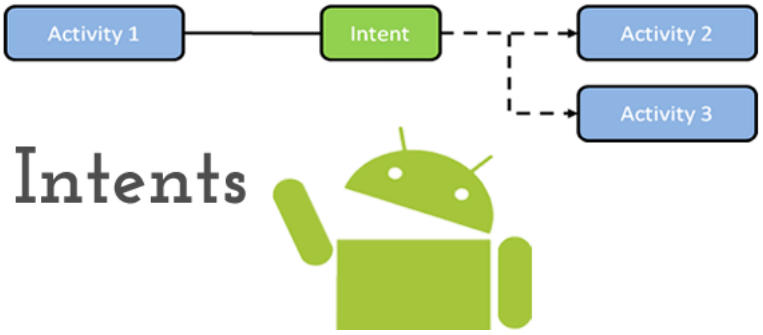



**Gráfico 16-2** Intents Android

**Fuente:** http://androideity.com/2011/10/17/trabajando-con-intents-en-android-overview/

## 2.4. Seguridad y vulnerabilidades en Android

En Android existen diferentes tipos de vulnerabilidades, entendiéndose por vulnerabilidades como puntos débiles del software que permiten que un atacante comprometa la integridad, disponibilidad o confidencialidad del dispositivo.
Es importante destacar que las vulnerabilidades son características de cualquier sistema complejo de software, ya que no se puede afirmar que ningún sistema operativo hasta ahora existente es completamente seguro. (**QINGQING, TAO, TAN, & YIDONG**, 2013)

Algunas de las vulnerabilidades más severas permiten que los atacantes ejecuten código arbitrario, denominadas vulnerabilidades de seguridad, en un sistema comprometido. (**MICROSOFT**, 2015)

Al hablar de seguridad es muy importante identificar los cuatro componentes claves de la seguridad en la información.

- ***Confidencialidad:*** debe ser posible que nadie puede interceptar nuestras comunicaciones y apropiarse de la información delicada, de esta manera hacer mal uso de la misma. Es decir, es la característica que asegura que sólo los que tienen autorización podrán acceder a la información. A esta característica se la denomina también como privacidad.
- ***Autenticación:*** se refiere a que solamente el dueño del dispositivo móvil puede hacer uso de las funcionalidades del mismo. Es decir, es la propiedad que se hace referencia a la identificación, esta puede ser mediante el uso de claves, tokens, biométrico, entre otras. Es decir, es el vínculo entre la información y su emisor.
- ***Integridad:*** trata de que la información de voz, datos viaje en la red sin que sea alterada o modificada. Es decir, es la propiedad en la que se asegura la no alteración de la información, entendiéndose por alteración al cambio, borrado o sustitución de la información.
- ***No repudio:*** no debe haber la posibilidad de que el usuario que haya usado el dispositivo móvil pueda negar el uso del mismo, garantizando que haya pruebas que

el usuario ha hecho determinada acción, en otras palabras, es la característica en la que se asegura que ninguna parte pueda negar una acción realizada anteriormente.

Dentro de todas las características anteriormente descritas la más importante se podría considerar la de autenticación, debido a que sirve de poco obtener la confidencialidad e integridad si resulta que el receptor de la información no es el destinatario correcto.
Los cuatro conceptos de seguridad descritos anteriormente son la base fundamental de la seguridad en todo ámbito.

### 2.4.1. Capas de seguridad en dispositivos móviles

Primero es importante conocer las capacidades que poseen los dispositivos móviles. Principalmente los dispositivos móviles han vivido una importante revolución en cuanto a las aplicaciones que se pueden ejecutar dentro de las mismas. Esta revolución se ha marcado por tres aspectos.

- Un hardware potente, con muchos sensores.
- Un sistema operativo complejo que a través de un SDK sencillo y potente sea sencilla la programación para los desarrolladores.
- Un mercado de aplicaciones completamente integrado en el sistema y muy intuitivo, que facilite las operaciones entre los usuarios y desarrolladores.

Debido a las funcionalidades que se presentaron anteriormente, los dispositivos móviles acaban almacenando gran cantidad de datos, en su gran mayoría confidenciales, por ejemplo, cuentas de redes sociales, cuentas bancarias, documentos e imágenes personales.

El aumento de la información personal almacenada provoca que personas mal intencionadas aumenten su interés en obtenerla. Además, la complejidad de los sistemas operativos actuales para móviles, han aumentado los huecos de seguridad expuestos.

Para poder analizar la seguridad de los dispositivos móviles de manera eficiente, se han organizado en cuatro apartados: la comunicación inalámbrica, el sistema operativo, la aplicación y el usuario.


**Gráfico 17-2** Capas de seguridad en dispositivos móviles

**Realizado por:** Villa, H. 2016

#### 2.4.1.1. Comunicaciones Inalámbricas (Hardware)

Las comunicaciones inalámbricas brindan la facilidad de que dos o más dispositivos móviles puedan comunicarse entre ellos sin necesidad de estar conectados por un medio físico, tal es el caso de cables. Además, crean una capa de abstracción, lo que permite que dispositivos móviles con diferentes sistemas operativos puedan intercambiar información.

El uso de las comunicaciones inalámbricas presenta una serie de ventajas, como por ejemplo la escalabilidad y la movilidad. Sin embargo, refiriéndonos a temas de seguridad la comunicación inalámbrica representa siempre un hueco de seguridad en toda organización empresarial, debido a que la información que viaja a través de este medio es más difícil de proteger.

***Ataques y vulnerabilidades***

Para poder asegurar las redes inalámbricas es necesario primero analizar cuáles son las principales vulnerabilidades que representa este medio.

***Masquerading:*** es la acción en la que el atacante suplanta la identidad de algún ente del sistema (estación base o dispositivo móvil) con el fin de tener acceso al sistema. Este ataque incide directamente en la propiedad de autenticación, por lo tanto, es necesario la implementación de un eficiente proceso de autenticación, por ejemplo, en el caso de los dispositivos móviles se puede usar los portales cautivos.

***Denegación de servicio*:** se refiere a la acción en la que el atacante consigue que el servicio no esté disponible para los usuarios, o que también que el servicio se interrumpa o retrase. Este tipo de ataque es quizá el único que no se puede identificar con ninguna de las propiedades de seguridad definidas anteriormente. Generalmente se lo realizan enviando masivamente solicitudes de algún tipo de servicio por ejemplo dentro de las más conocidas están al ARP flooding, SYN flooding, entre otras.

***Eavesdropping:*** es la acción en la que el atacante obtiene información de una comunicación de la que no es ni emisor ni receptor. En este caso, el atacante vulnera la confidencialidad de la información que se ha interceptado. Este tipo de ataque se clasifica como ataque pasivo, debido a que el atacante obtiene información de los datos del tráfico de red que se genera, pero no puede realizar ninguna acción sobre la red ni sobre la información que circula. Es importante tener en cuenta, sin embargo, que la información obtenida en un ataque de eavesdropping puede dar lugar a un posterior ataque de masquerading.

***Confidencialidad de posicionamiento:*** es la acción en la que el atacante obtiene, mediante diferentes técnicas, la posición física de un dispositivo móvil. Este tipo de ataque puede afectar seriamente a la privacidad de las personas, debido a que pueden ser localizadas (mediante el GPS u otros medios) en cualquier momento por el mero hecho de tener un dispositivo móvil en funcionamiento. Por ejemplo, al instalar una aplicación de dudosa procedencia se puede permitir el acceso al GPS y localizar a la víctima.

***Mecanismos de prevención***

Debido al creciente uso de los entornos inalámbricos en los diferentes escenarios de la vida cotidiana, como por ejemplo aplicaciones de comercio electrónico, es importante el aseguramiento de este medio.
Al hablar de los métodos y técnicas para prevenir riesgos de seguridad se debe hacer una distinción entre los que trabajan en el nivel físico de la comunicación y los que trabajan en el resto de los niveles, tanto si se trata del nivel de enlace como el nivel de la aplicación.

Las técnicas de prevención más habituales que se aplican al medio físico son las de difusión del espectro, las mismas que basan su funcionamiento en fraccionar la señal de radio y transmitirla de manera imperceptible por diferentes frecuencias. De esta manera, si no se conoce el modo como la señal ha sido distribuida por las diferentes frecuencias, no se puede reconstruir, ya que las diferentes señales que se reciben en cada frecuencia son percibidas como ruido.

Las técnicas de difusión de espectro también permiten atenuar los ataques de jamming, ya que la señal emitida en una frecuencia concreta para producir el ataque solo afectará a una parte de los datos enviados.

Las técnicas de difusión no son seguras, pero al incorporar la **criptografía** a esta técnica se puede conseguir buenos niveles de seguridad dependiendo del algoritmo que se use.
La mayoría de los sistemas de comunicación inalámbrica llevan a cabo el servicio de autenticación por medio del *modelo reto-respuesta*. El mismo que consiste en un intercambio de mensajes entre las dos partes que se quieren autenticar para asegurarse de que cada una de ellas conoce cierta información previamente intercambiada y que, por lo tanto, es quien dice ser. La ventaja de este modelo es que el reto varía aleatoriamente para cada proceso de autenticación, además se puede incorporar a este modelo el método de autenticación en 2 etapas la misma que consiste en el uso de las preguntas de desafío y además de un envío de un código al correo o al dispositivo móvil para la completa confirmación de la autenticidad del usuario.

Refiriéndose a la confidencialidad en el ambiente inalámbrico se implementa esquemas de criptografía basados en el uso de la clave compartida. La implementación de este

proceso no representa ningún problema a los modelos de comunicación que existen actualmente.

De esta forma mediante la criptografía se puede obtener las características de autenticación y confidencialidad reduciendo el índice de ataques al medio inalámbrico.

#### 2.4.1.2. Sistema operativo

El sistema operativo es la capa que se encuentra entre el hardware y las aplicaciones de software, en los últimos años la complejidad de los sistemas operativos móviles ha aumentado considerablemente, lo que ha provocado que la seguridad se convierta en un requisito primordial. En la actualidad la mayoría de los dispositivos móviles de gama alta ejecutan un sistema operativo muy potente, por lo que en muchos casos están llenos de vulnerabilidades.

***Ataques y vulnerabilidades***

Las principales amenazas que existen en el ámbito del sistema operativo son causadas por errores en el aislamiento de los recursos, ya sea debido a sus diseños, a errores en el software o a una mala configuración de sus servicios.

Estas vulnerabilidades pueden ser aprovechadas para ejecutar ataques tanto desde los servicios que ofrece el propio sistema operativo, como desde la capa de aplicaciones. Por ejemplo, el riesgo que se corre al instalar aplicaciones fuera del repositorio de Google Play, las mismas que pueden tener código malicioso

Por otra parte, también circulan versiones no oficiales de los sistemas operativos móviles, denominadas ROM. Pueden ser copias de las versiones oficiales de los sistemas operativos o ROM personalizadas. Además, las ROM personalizadas pueden contener código malicioso.

***Mecanismos de prevención***

Los mecanismos de seguridad más importantes que se deben tomar en cuenta son los privilegios de usuarios, el aislamiento de procesos y las actualizaciones

***Privilegios de usuario:*** Generalmente, todas las aplicaciones se ejecutan con los privilegios del usuario normal, limitando mucho los cambios o desperfectos que el usuario puede causar al sistema. Por una parte, esto es muy importante, ya que en caso de que haya una vulnerabilidad, los daños que se podrán producir estarán limitados por los privilegios que el usuario tenga. Pero también es una limitación para el usuario, ya que únicamente podrá realizar las acciones que el sistema operativo le permita llevar a cabo con los privilegios actuales.

***Aislamiento de procesos:*** Aislar la ejecución de cada aplicación garantiza que no pueden interferir en el funcionamiento de las otras, lo que hace el sistema operativo mucho más robusto. No obstante, cuando varias aplicaciones tienen que compartir algún servicio, hay que utilizar un sistema de permisos más complejo. La buena implementación de este aislamiento y la gestión de los recursos compartidos son fundamentales para que esta medida sea efectiva.

***Actualizaciones:*** Los sistemas operativos también ofrecen actualizaciones mayores, las que además de solucionar errores en el software, también añaden nuevas funcionalidades y mejoran su rendimiento. Dado que estas actualizaciones realizan grandes cambios en el sistema, algunas llevan a cabo un borrado completo del dispositivo.

### 2.4.1.3. Aplicaciones

Uno de los componentes más importantes en los dispositivos móviles son las aplicaciones que se pueden ejecutar sobre ellos, ya que con ellas interactuarán los usuarios, en este nivel es muy importante revisar las medidas de seguridad que se han desarrollado para que las aplicaciones no puedan desestabilizar el sistema, debido a que las vulnerabilidades que se detecten en este nivel son críticas.

***Ataques***

Dentro de este nivel los ataques se pueden dividir en 2 tipos:

- Ataques que se pueden ejecutar desde cualquier tipo de aplicación
- Ataques que se pueden realizar únicamente desde el navegador web

Siendo estas los puntos más críticos en la seguridad a este nivel.

***Ataques al software: malware***

Dentro del software hay varias formas de crear huecos de seguridad o vulnerabilidades dentro de las cuales tenemos el malware o también denominado como código malicioso. El malware es una aplicación de software que tiene un objetivo malicioso en el dispositivo móvil donde se instala y se ejecuta sin el consentimiento del propietario. Puede tener objetivos muy variados, siendo los más comunes obtener datos personales y beneficio económico. Su modo de funcionamiento puede ser automático o controlado remotamente. Los principales tipos de malware son:

***Virus:*** Es un programa malicioso que infecta a otros archivos del sistema con la intención de modificarlos o hacerlos inservibles. Una vez un archivo ha sido infectado, también se convierte en portador del virus y, por lo tanto, en una nueva fuente de infección. Para que un virus se propague, este archivo debe ser ejecutado por el usuario. Generalmente tiene un objetivo oculto, como puede ser obtener contraseñas o realizar un ataque de denegación de servicio.

***Gusano.*** Es un programa malicioso autorreplicable que aprovechará las vulnerabilidades de la red para propagarse. Al igual que el virus, generalmente tiene un objetivo oculto.
*Troyano*. Es un pequeño programa oculto en otra aplicación. Su objetivo es pasar inadvertido por el usuario e instalarse en el sistema cuando el usuario ejecuta la aplicación. Una vez instalado, puede realizar diversas acciones, pero todas ellas sin el consentimiento del usuario. Además, estas acciones pueden realizarse instantáneamente o estar fijadas para realizarse en un futuro.

***Puerta falsa.*** Es un programa cuyo objetivo es abrir un acceso al ordenador para el desarrollador del malware, ignorando el proceso normal de autenticación. Esto implica que el dispositivo móvil infectado puede ser controlado remotamente por el atacante.

***Spyware.*** Es una aplicación que recoge información sobre una persona u organización sin su consentimiento. Generalmente, el objetivo final de esta información recopilada es venderla a empresas de publicidad.

***Keylogger.*** Es una aplicación encargada de almacenar todas las pulsaciones de teclado. Por lo tanto, puede capturar información confidencial, como el número de la tarjeta de crédito o las contraseñas.

***Hijacker.*** Es un programa que realiza cambios en la configuración del navegador web. Un ataque típico es cambiar la página de inicio por una página de publicidad.

***Dialer.*** Es un programa que de manera oculta realiza llamadas a teléfonos con tarifas especiales. De esta manera, el atacante puede obtener beneficios económicos.

Hoy en día el malware tiene un alto nivel de peligrosidad lo que implica un gran riesgo a los dispositivos móviles ya que se convierten en puntos vulnerables para los atacantes.

Además, aumenta el riesgo cuando el usuario desconoce de los posibles peligros a los que están expuestos dichos.

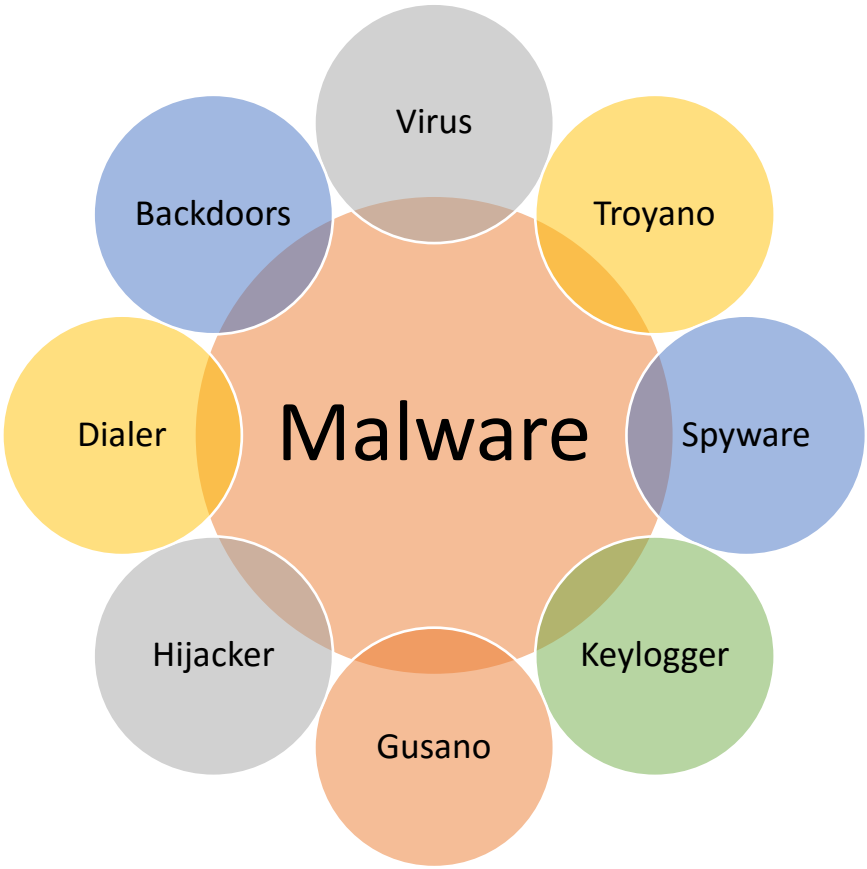



**Gráfico 18-2** Tipos de malware en dispositivos móviles

**Realizado por:** Villa, H. 2016

En el caso de ser desarrolladores una práctica muy habitual es la de reutilizar librerías en el desarrollo de nuevas aplicaciones, pero el problema principal es que no se conoce al 100% el código fuente de la librería lo que implicaría que hay la posibilidad de que contenga código malicioso y por tanto estaríamos auspiciando la propagación del malware.

***Ataques en la web***

Se analizará las principales vulnerabilidades que pueden afectar a los navegadores web con el fin de analizar las diferentes formas de ataque por este medio. Por el momento analizaremos el web spoofing o phishing y el clickjacking.

*Web spoofing o phishing* es un ataque que consiste en suplantar una página web, reemplazándola por una completamente igual, pero con una dirección web parecida y a

partir de ahí obtener información como por ejemplo usuarios, contraseñas, números de cuentas, entre otras.

Clickjacking es una técnica que engaña al usuario para que haga un clic sobre elementos de un sitio web que no haría voluntariamente. Esto se consigue superponiendo dos páginas. La principal es la que contiene un elemento que al atacante le interesa que pulsemos, como puede ser la confirmación de la habilitación de un permiso. La otra, la que nos engaña, está superpuesta a la primera. Obviamente, la página superpuesta debe tener elementos que nos incentiven a pulsar los botones que al atacante le interesan, como por ejemplo, algún tipo de juego.

Sin entrar demasiado en detalle, esta técnica se basa en la superposición de iframes. Estos son elementos HTML que permiten la inclusión de un recurso externo dentro de nuestra página. Y aunque tienen una serie de limitaciones a la hora de su acceso mediante JavaScript, es posible utilizarlos para engañar a los usuarios.

***Mecanismos de prevención***

Dentro de la capa de aplicación se tratará de utilizar diferentes mecanismos de seguridad para que se pueden utilizar en la web, como por ejemplo

***Mercado de aplicaciones*:** es importante el uso de las aplicaciones que están dentro de un mercado de aplicaciones en las cuales el usuario puede descargarse con confianza e instalarlo en su dispositivo móvil, ya que previamente el código ha sido revisado por una tercera persona y en este caso se brinde una valoración a la aplicación por ejemplo Play Store (Android), AppStore (iOS)

***Navegador Web:*** Por defecto, todos los sistemas operativos se venden con una aplicación encargada de la navegación web. Aunque no parecer un punto crítico en la seguridad del sistema, la experiencia en los ordenadores corrobora que sí lo es, debido a que el navegador web ejecuta código muy complejo con los mismos privilegios que el usuario tenga establecido.

Por tal motivo es necesario tener actualizado el navegador que usamos, y tener cuidado en las páginas que visitamos ya que pueden crear puntos de fallo. Además, como medidas

de reducción de los ataques se usa el protocolo HTTPS, destinado básicamente a crear un canal seguro de comunicación.

Para que una conexión HTTPS sea segura, debe cumplirse que:

- El usuario confíe en la autoridad de certificación.
- El sitio web proporcione un certificado válido, firmado por la misma autoridad de certificación en la que confía el usuario, y que este certificado identifique correctamente el sitio web.
- El navegador web del usuario alerte cuando detecta que hay un certificado inválido.
- El protocolo utilizado para el cifrado sea de confianza.

***Aplicaciones de seguridad:*** Existen diferentes aplicaciones que pueden añadir nuevas capas de seguridad a los dispositivos móviles, ya sea con métodos de autenticación adicionales más restrictivos, sistemas de copia de seguridad, cifrado de los datos, aplicaciones antivirus o cortafuegos.

#### 2.4.1.4. Usuario

Dentro de este nivel están las intrusiones físicas en el dispositivo, al hablar de intrusión física decimos de alguien que ha tenido acceso a nuestro dispositivo móvil de manera física o tangible. Este tipo de intrusión es el más peligroso que podemos sufrir, ya que nuestro dispositivo está completamente vulnerable.

***Ataques y vulnerabilidades***

Los ataques que se pueden realizar dependen básicamente de si el dispositivo está en funcionamiento o no y del tiempo que el atacante tenga para comprometer el sistema.

En el caso de ataques momentáneos, el atacante tendrá poco tiempo para realizar acciones.

Si el dispositivo está apagado y protegido con el código PIN, lo podemos considerar como seguro. En cambio, si está encendido, se pueden emprender varias acciones:

- Leer la información fácilmente accesible, como los contactos, SMS, correo electrónico o fotografías.
- Eliminar información fácilmente accesible.
- Reenviar alguna información importante, como un correo electrónico que contenga la contraseña para algún servicio web.
- Conectar el dispositivo móvil a un ordenador y copiar parte del contenido de su memoria, donde puede haber, por ejemplo, documentos, imágenes, contraseñas o datos de aplicaciones.
- Instalar código malicioso.
- Copia de toda la memoria.
- Técnicas forenses para recuperar información que ha sido borrada recientemente.
- Intentar encontrar las contraseñas mediante ataques de fuerza bruta.

***Mecanismos de prevención***

Dentro de los mecanismos de prevención que se puede establecer son:

- Establecer un mecanismo de autenticación al encender el dispositivo móvil evitando el uso de contraseñas débiles o fáciles de deducir,
- Cifrar la información dentro del dispositivo móvil
- Poder eliminar la información de manera remota
- Instalar aplicaciones que permitan el rastreo del dispositivo móvil

## 2.5. Modelo ioSTS (Input/Output Symbolic Transition System)

Varios tipos de formalismos han sido propuestos para representar políticas de seguridad, propiedades o vulnerabilidades por ejemplo se ha hecho el uso de expresiones regulares, el uso de lógicas no clásicas (lógicas temporales y deónticas), core typed languages o state machines. En este contexto se ha considerado el uso del Modelo ioSTS (input output Symbolic Transition Systems) para la construcción de los patrones que determinarán si una aplicación es vulnerable o no, sin el riesgo de cometer ambigüedad en el análisis.

El modelo de ioSTS permite modelar sistemas críticos o imperativos sin el uso de la recursividad y la comunicación con su entorno, permitiéndoles ser más autónomos. Un formalismo de ioSTS está hecha de variables, las acciones de entrada y salida que llevan parámetros de comunicación realizadas por acciones, guards y tareas. **(BERND KLEINJOHANN**, 2007)

Hay que aclarar que un *guard* es una *expresión boleana que ayuda a realizar una comprobación de la integridad del programa* para evitar errores en la ejecución. Un ejemplo típico es comprobar que una referencia a ser ejecutado no debe ser nulo, para evitar errores de punteros nulos.

Este modelo es ampliamente usado en la verificación y testing en diferentes campos. Dentro del modelamiento de grandes y complejos sistemas, por ejemplo, se puede modelar las composiciones de servicios web, sistemas críticos, aplicaciones web y de escritorio. En el caso de que la documentación de Android sea enriquecida o si se descubren nuevas vulnerabilidades, el modelo de ioSTS se adaptará muy bien a estos nuevos requerimientos.

Por lo tanto, este modelo muestra una gran flexibilidad y acoplamiento a los nuevos avances que se tendrían en Android, característica que no disponen los formalismos anteriormente descritos.

### 2.5.1. Patrones de vulnerabilidad

Los patrones de vulnerabilidad que se determinarán ayudarán a describir si los comportamientos de los componentes son vulnerables o no, en base a autómatas simbólicos las mismas que están compuestas por variables y guards sobre variables que permiten escribir las restricciones sobre las acciones que se puede realizar.

Estos patrones servirán para mitigar las vulnerabilidades que se pueden explotar en base a los intents se centran en que éste lleva información delicada desde una aplicación (actividad) a otra. Y esta puede ser monitoreada mediante diferentes técnicas. **(CHIN, FELT,**

**GREENWOOD, & WAGNER**, 2011). Estas vulnerabilidades pueden ser mitigadas primero detectándolas y después usando mejores prácticas de programación en Android.

Un formalismo de ioSTS es un tipo de modelo de autómata que posee dos tipos de variables, variables internas que se usan para almacenar datos y los parámetros que ayudan a enriquecer las acciones que se desarrollan dentro del modelo.

Para la creación de los patrones se basará en la documentación oficial de Android, referente a las aplicaciones y algoritmos que éste usa.

Los beneficios de usar el formalismo de ioSTS son algunos en los cuales podemos describir:

- Nos permite reusar los patrones ya creados, los cuales nos ayudará a generar los casos de prueba.
- El uso de operadores facilita la creación de los patrones, y por ende la determinación de las vulnerabilidades.
- El uso de la documentación oficial de Android dentro de ioSTS, permitirá precisar de mejor manera la detección de vulnerabilidades.
- En el análisis no se comete el riesgo de obtener ambigüedad en los resultados, ya que es una característica de ioSTS.
- Permite flexibilidad en la creación de patrones.

# CAPÍTULO III

## DISEÑO DE INVESTIGACIÓN

Dentro de este capítulo se detalla el tipo de investigación el diseño, los métodos y técnicas utilizadas, los instrumentos, con su respectiva validación, que se van a usar dentro del trabajo de investigación, adicionalmente se crea la propuesta de la aplicación que escanee las vulnerabilidades en aplicaciones Android conjuntamente con la guía para las mejores prácticas para mitigar las vulnerabilidades basadas en intents.

### 3.1. Tipo de investigación

El presente trabajo utiliza el tipo de investigación experimental.

- **Experimental:** el trabajo investigativo se fundamenta en pruebas realizadas en escenarios de laboratorio, en las que se observa los elementos más importantes del objeto de estudio que se investiga y así obtener una captación de los fenómenos a primera vista.

### 3.2. Diseño de la Investigación

El diseño de la investigación del presente trabajo es del tipo experimental ya que en base a estudios anteriores se desarrolla una aplicación que escanee las vulnerabilidades en base a los intents en las aplicaciones Android, además los datos de prueba son generados por el autor de esta investigación.

## 3.3. Métodos y técnicas

Los métodos y técnicas que se utilizan en la presente investigación son los siguientes

### 3.3.1. Métodos

La presente investigación se desarrollará bajo el método científico, ya que utiliza varias etapas para obtener un conocimiento válido desde el punto de vista científico, para esto se utiliza instrumentos que resulten fiables y seguros.

El método científico consta de las siguientes etapas:

- Planteamiento del problema
- Formulación de la hipótesis
- Levantamiento de la información
- Análisis e interpretación de resultados
- Comprobación de la hipótesis
- Difusión de resultados

Para el desarrollo de la aplicación móvil se ocupará una metodología ágil, que permitirá el avance correcto en el desarrollo de la aplicación.

### 3.3.2. Técnicas

Las técnicas que serán utilizadas en el presente trabajo de investigación son:

- **Búsqueda de información:** permite obtener la información necesaria acerca del objeto de estudio de la investigación para su desarrollo, utilizando las fuentes secundarias disponibles.
- **Pruebas:** permite realizar experimentos en escenarios de laboratorio.

- **Observación:** permite determinar resultados de las pruebas realizadas en los escenarios de laboratorio.
- **Análisis:** permite determinar los resultados de la investigación.
- **Estadística descriptiva:** nos ayuda a la demostración de la hipótesis planteada.

## 3.4. Instrumentos

Los instrumentos que se usan para el desarrollo del trabajo de investigación son los siguientes:

- **Netbeans:** es un entorno de desarrollo (IDE) que está escrito en Java, además que es un producto libre sin restricciones para el uso. Netbeans permite el desarrollo de aplicaciones JAVA, PHP, HTML, entre otras; este desarrollo se lo realiza a partir de un conjunto de componentes software denominados módulos. Este entorno de desarrollo se utilizará para el desarrollo de la aplicación que genere los casos de prueba JUnit para la detección de vulnerabilidades en aplicaciones móviles Android.
- **Eclipse:** es un entorno de desarrollo (IDE) que permiten el desarrollo de aplicaciones Android para dispositivos móviles, dentro del trabajo de investigación se utilizará para la creación de las aplicaciones que sirvan de tester y test de las vulnerabilidades.
- **SDK de Java:** es un conjunto de herramientas software que permite al desarrollador de software crear aplicaciones concretas, es decir, es una API (Interfaz de programación de aplicaciones).

## 3.5. Validación de instrumentos

Los instrumentos de software que se han utilizado dentro del trabajo de investigación han sido seleccionados de acuerdo a diferentes criterios como afinidad, ventajas, versatilidad y que sea actual.

**Netbeans**


**Gráfico 1-3** Logo netbeans
**Fuente: http:**//www.netbeans.org

Se ha seleccionado el IDE de Netbeans debido a las diferentes características que posee entre las que constan las siguientes:

- Netbeans se usa para desarrollar cualquier tipo de aplicación Java, ya que posee compatibilidad con el uso de plugins mediante el cual se puede adquirir funciones de acuerdo a las necesidades del proyecto a desarrollar.
- Este IDE permite la reutilización de módulos dentro del desarrollo facilitando la reutilización de código.
- Netbeans permite actualizaciones simples
- Posee soporte para PHP
- Posee un editor que facilita la programación

Dentro del trabajo de investigación se lo ha utilizado para la creación de la aplicación que genere los casos de prueba que serán específicos para la aplicación que vayamos a analizar las vulnerabilidades.

**Eclipse**

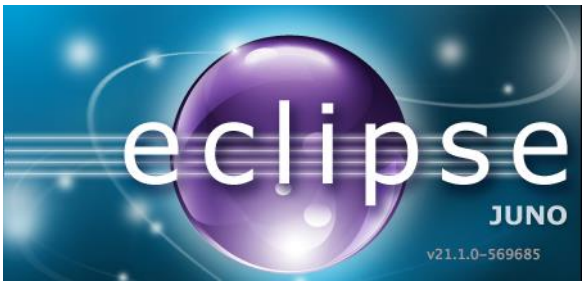



**Gráfico 2-3** Logo eclipse
**Fuente:** http://www.eclipse.org

Eclipse Juno ayudará en el trabajo de investigación en la creación de las aplicaciones que sirvan de tester y test de las vulnerabilidades. Debido a que presenta las siguientes características:

- Facilidad en la instalación
- Interfaz gráfica amigable
- Posibilita la rápida creación de aplicaciones Android
- Integración con los comandos Ant
- Compilación en tiempo real
- Permite la creación de pruebas unitarias JUnit
- Uso de plugins que permiten añadir características específicas al IDE

## 3.6. Fuentes

Las principales fuentes que serán utilizadas en el estudio de investigación serán:

**Primaria**

- Pruebas
- Observación de resultados

**Secundaria**

- Tesis realizadas internacionales y nacionales de cuarto nivel.
- Trabajos de investigaciones internacionales y nacionales.
- Artículos científicos en base de datos de bibliotecas virtuales.
- Libros especializados en la biblioteca y electrónicos.
- Diccionarios especializados.
- Conferencias académicas, congresos, seminarios.
- Revistas indexadas y no indexadas publicadas de prestigio.
- Revistas electrónicas.
- Páginas de internet que brinden información confiable.

## 3.7. APSEBI

### 3.7.1. Introducción

Se planteó el desarrollo de una nueva aplicación para el escaneo de vulnerabilidades debido a que proyectos anteriores citados en el estado del arte del presente trabajo de investigación no proveían un análisis de vulnerabilidades que sea flexible y que permita analizar la comunicación entre los sandbox a través de intents, por tal motivo el presente trabajo de investigación se plantea el desarrollo de una aplicación que conjugue dichas características.

La aplicación Android que se ha desarrollado en este trabajo de investigación, está basada en el estudio y análisis de los proyectos anteriormente citados, a la misma se ha denominado APSEBI (**A**plicación para **P**ruebas de **SE**guridad **B**asadas en **I**ntents).

APSEBI determina las vulnerabilidades basadas en intents en un proyecto Android; cabe indicar que el análisis lo realiza a todo el proyecto y no solo el archivo APK (Application Package File).

APSEBI analiza diferentes componentes del proyecto Android, enfocándose en las Actividades (Activities) y Servicios (Services) del proyecto; además analiza la manera de cómo estos componentes se relacionan con el ContentProviders, teniendo en cuenta que éste no puede ser llamado directamente por los Intents, pero de alguna forma pueden exponer datos sensibles (datos personales, contraseñas, etc.) a través de otros componentes.

Para el análisis del proyecto Android, APSEBI lo realiza mediante el uso de casos de prueba, los mismos que son generados de una mezcla entre los archivos del mismo proyecto que se analiza (componentes compilados, Manifest) y de patrones que describen modelos de vulnerabilidades específicas en Android.

Como se describió en el capítulo II, APSEBI se basa en un modelo de pruebas denominado Model-Based, en el cual se usa el formalismo ioSTS que ayuda a la descripción de los patrones de vulnerabilidad, los mismos que ayudan a determinar si los componentes de una aplicación tienen comportamientos que expongan vulnerabilidades o no, en los datos que manejan.

En el capítulo II además se describen las ventajas del uso del formalismo ioSTS destacándose que es lo suficientemente flexible y adaptable para en la creación de los patrones de vulnerabilidad (basadas en intents), ya que se puede describir y reutilizar una variedad de vulnerabilidades de una manera sencilla y fácil.

### 3.7.2. Arquitectura

La arquitectura de la aplicación está conformada de dos partes principales:

- El generador de casos de prueba para analizar las aplicaciones
- El framework que ejecuta los casos de prueba

## GENERADOR DE CASOS DE PRUEBAS

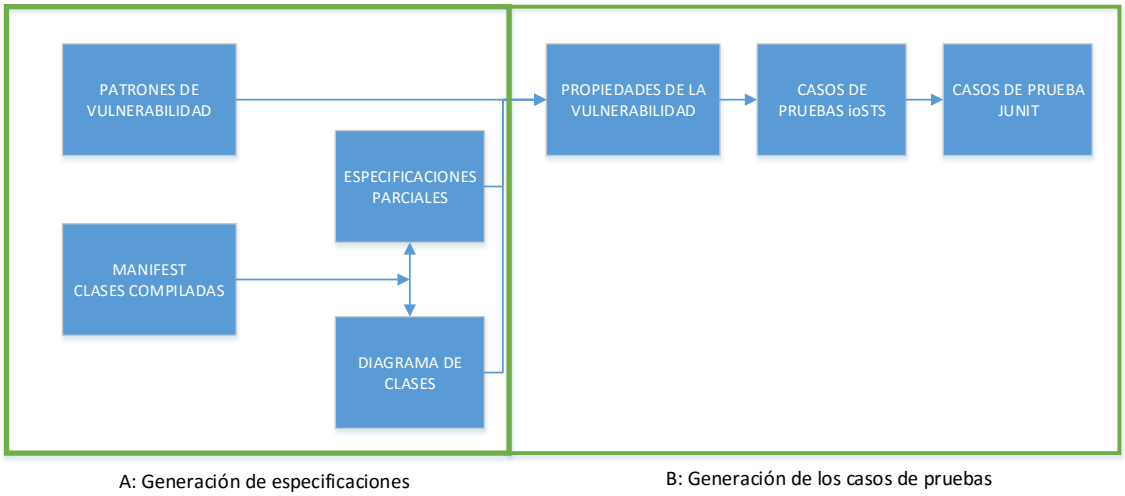



**Gráfico 3-3** Generador de casos de pruebas
**Realizado por:** Villa, H. 2016

La figura anterior representa los principales pasos que se siguen para la generación de los casos de prueba.

El generador de clases de prueba está creado bajo la estructura de la arquitectura MVC (Model View Controller).

### *Patrones de Vulnerabilidad*

Los patrones de vulnerabilidad que usa APSEBI son creados por expertos en el desarrollo de Android, éstos generalmente se basan en las vulnerabilidades que ya han sido descubiertas; por ejemplo OWASP (Open Web Application Security Project) es una organización que se dedica a la creación, mantenimiento de aplicaciones confiables y seguras, por lo tanto, contribuye en la búsqueda de huecos de seguridad, en este caso dentro del sistema Android y crea informes del funcionamiento y mitigación de los mismos y los publica en la página web. Estos patrones se guardan como archivos con extensión .dot, los mismos que permiten la visualización y un mejor entendimiento del patrón.

***Especificaciones Parciales ioSTS***

APSEBI genera **Especificaciones Parciales ioSTS** usando la reflexión de Java, a través de la documentación oficial de Android.

Estas especificaciones parciales ayudan a depurar el resultado final de las pruebas realizadas a los componentes compatibilizándolas con la documentación oficial de Android, previniendo falsos positivos, ya que cada componente es sometido a prueba de acuerdo a la especificación generada.

***Diagrama parcial de clases***

APSEBI también genera un diagrama parcial de clases usando el mismo método, la reflexión de Java.

Este diagrama lista los componentes de la aplicación en la cual se describe que tipo de componentes son y las relaciones que mantienen entre ellos.

De esta forma mediante éste diagrama se tienen una idea más amplia de la estructura de la aplicación Android.

Las *Especificaciones parciales ioSTS* y el *Diagrama parcial de clases* son creadas en base a los archivos generados por el mismo proyecto que se analiza (Manifest, Componentes Compilados)

***Propiedades de las Vulnerabilidades***

A la combinación de los patrones de vulnerabilidad, las especificaciones parciales ioSTS y el diagrama de clases se les denomina **propiedades de las vulnerabilidades**.

Estas propiedades pueden ser refinadas mediante de los intents (explícitos e implícitos) que recibe la aplicación Android.

***Casos de prueba ioSTS***

Al definir las propiedades de las vulnerabilidades se obtiene los casos de prueba, los mismos que serán completados con valores al ejecutar la aplicación.

Finalmente, los resultados de los casos de prueba ioSTS conjuntamente con las transiciones, variables y guards son convertidas en los **casos de prueba JUNIT** que son los que realmente se ejecutan en la aplicación.

## FRAMEWORK DE EJECUCIÓN DE LOS CASOS DE PRUEBA DE JUNIT

En la figura a continuación se muestra el funcionamiento del framework que ejecuta los casos de prueba JUNIT, la ejecución se lo puede realizar directamente en el dispositivo Android o en algún AVD (Android Virtual Device).

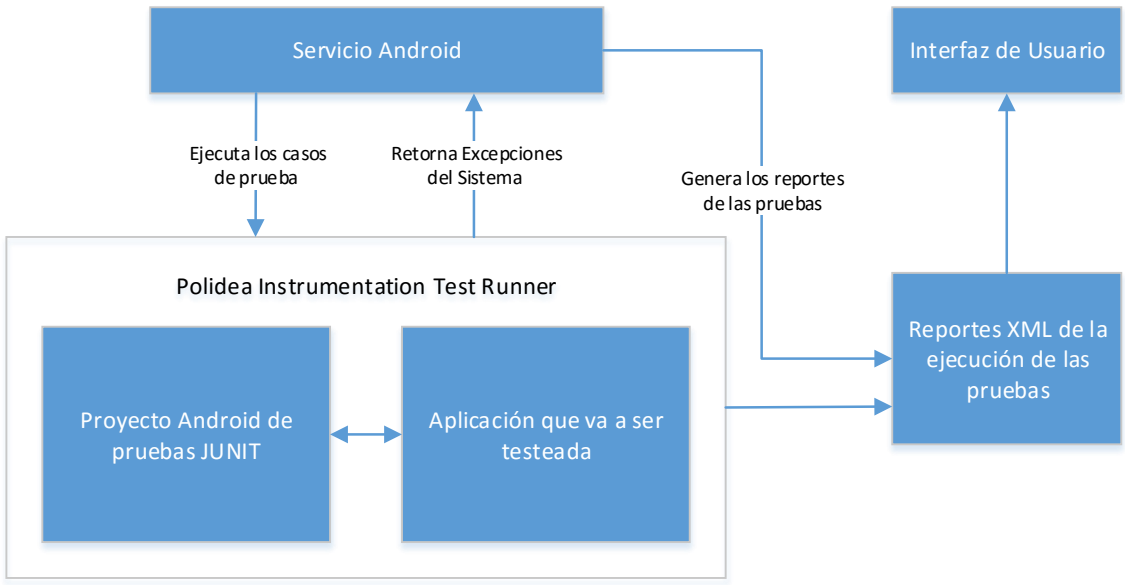



**Gráfico 4-3** Framework de ejecución de los casos de prueba de JUNIT
**Realizado por:** Villa, H. 2016

El framework está compuesto por la herramienta de ejecución de pruebas Android y complementada con el software de *PolideaInstrumentation* que permite crear reportes formato en XML para tener un control y visualización de las vulnerabilidades.

La ejecución de los casos de prueba lo realiza un servicio Android, el cual despliega los resultados en el dispositivo o en el emulador.

El **Test Runner** inicia el componente en el que se va a ejecutar los casos de prueba al cual se lo denominará Component Under Test (CUT), y después realiza el proceso de manera iterativa, ejecutando en procesos separados los casos de prueba en cada uno de los componentes del proyecto Android.

Es necesario separar los procesos de ejecución de las pruebas ya que ayuda a capturar de mejor forma las excepciones de los componentes, y además gestionar de mejor forma la ejecución de la aplicación en el caso de que se bloquee el componente objeto de las pruebas.

Una vez que todos los casos de prueba han sido ejecutados, el servicio de Android se encarga de mostrar todos los resultados en la pantalla y genera un archivo XML, que es el reporte detallado de la ejecución de los casos de prueba.

Estos son los tipos de resultados que pueden aparecer en la ejecución de los casos de prueba:

- **VUL:** cuando encuentra que algún componente es vulnerable a los tipos de ataques descritos en los patrones de vulnerabilidad
- **NVUL:** si ninguna vulnerabilidad ha sido detectada en la aplicación a analizarse
- **INCONCLUSIVE:** ocurre cuando un escenario de vulnerabilidad no puede ser testeado

Además, los resultados pueden ser VUL/FAIL y NVUL/FAIL que son asignados cuando algún componente que está siendo testeado no cumple con la especificación de Android (por tal motivo es importante para la generación de las especificaciones parciales ioSTS tomar en cuenta que los algoritmos de reflexión de Android). Por ejemplo, una actividad es llamada a través de los intents usando el PICK action, y no retorna una respuesta. Si no responde mientras se está realizando el test de la aplicación, se tendrá como resultado FAIL o NVUL/FAIL.

### 3.7.3. Desarrollo

APSEBI está compuesto por dos aplicaciones que ayudarán:

- El generador de especificaciones (SpecGen) y
- La herramienta que ayuda a testear las aplicaciones Android (Testing Tool).

**GENERADOR DE ESPECIFICACIONES (SpecGen)**

El generador de especificaciones permite La estructura del aplicativo se muestra a continuación.

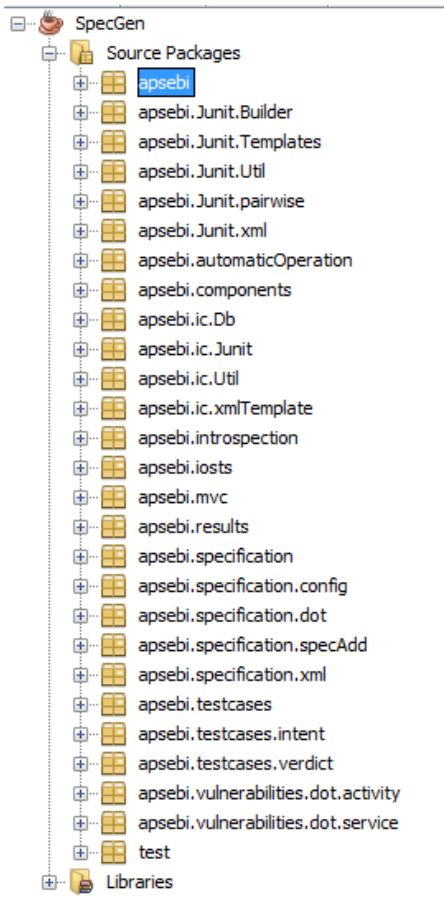



**Gráfico 5-3** Estructura del Generador de especificaciones (SpecGen)
**Realizado por:** Villa, H. 2016

Se puede observar que está compuesto por 26 paquetes que ayudan a generar las especificaciones, a continuación, se detallan el contenido de los paquetes que conforman el generador de las pruebas unitarias.

*Paquete apsebi*

Este paquete contiene la clase Main.java, la cual permite ejecutar la aplicación desde la línea de comandos con los siguientes argumentos:

-f: Ruta del proyecto Android
-v: Tipo de vulnerabilidad
-adb: ADB
-n: número de pruebas
-sdk: Android SDK
-t: Ruta del proyecto de prueba Android
-tester: Ruta de la herramienta de prueba

*Paquete apsebi.Junit.Builder*

Dentro de este paquete se encuentran las clases que nos ayudan a generar el código para la creación de los casos de prueba Junit, de acuerdo al tipo de patrón de vulnerabilidad seleccionado, en este caso Integrity (Integridad) y Availability (Disponibilidad).

*Paquete apsebi.Junit.Templates*

Este paquete agrupa las plantillas, clases abstractas y demás para la generación de los casos de prueba Junit tanto para las actividades y para los servicios.

*Paquete apsebi.Junit.Util*

El paquete considera los métodos que serán comunes en la creación de los casos de prueba Junit.

*Paquete apsebi.Junit.pairwise*

Este paquete considera la clase que crea los valores de secuencia que serán tomados en la generación de los casos de prueba Junit.

*Paquete apsebi.Junit.xml*

Este paquete contiene las clases que gestionan la creación del reporte xml al realizar las pruebas de casos en los componentes de la aplicación Android.

*Paquete apsebi.automaticOperation*

Este paquete contiene 3 clases en las cuales se gestionan operaciones de manera automática, siendo estas:

- AntManager.java: Gestiona toda la ejecución de los comandos ANT que se realiza de forma habitual en la aplicación, en este caso son: Debug, Installd, Clean.
- FileManager.java: Gestiona copia de archivos desde un origen a un destino específico.
- RunManager.java: Gestiona la instalación, desinstalación y ejecución de la aplicación en el dispositivo físico o en el emulador.

*Paquete apsebi.components*

Este paquete contiene las clases que ayudarán a obtener información referente a Actividades, ContentProvider, Receivers y Servicios del archivo Manifest de la aplicación objeto de análisis.

*Paquete apsebi.ic.Db*

Este paquete contiene las clases que estructura y gestiona la base de datos de la aplicación para la generación de los casos de prueba.

*Paquete apsebi.ic.Junit*

Este paquete contiene las clases que articulan el código de las pruebas unitarias con lo que se obtuvo de información del archivo Manifest de la aplicación Android.

*Paquete apsebi.ic.Util*

El paquete considera los métodos que serán comunes en la articulación entre las pruebas unitarias y el Manifest.

*Paquete apsebi.ic.xmlTemplate*

Este paquete contiene los formatos xml para la generación de los reportes de los casos de prueba.

*Paquete apsebi.introspection*

Este paquete contiene la clase que ayuda a revisar si un componente tiene un ContentResolver, un tipo de URI y esta información le envía al ContentProvider para su revisión. De igual manera realiza el mismo proceso para los Intent.

*Paquete apsebi.iosts*

Este paquete contiene las clases que gestionan la generación de los casos de prueba ioSTS, que en lo posterior serán la base para la generación de los casos de prueba Junit.

*Paquete apsebi.mvc*

Este paquete contiene las clases que estructuran la aplicación con la estructura Model View Controller (MVC).

*Paquete apsebi.results*

Este paquete contiene la clase que gestiona la manera en la cual se van a mostrar los resultados de las pruebas Junit.

*Paquete apsebi.specification*

Este paquete contiene las clases que generan las especificaciones para en lo posterior la creación de las propiedades de las vulnerabilidades.

*Paquete apsebi.specification.config*

Contiene las clases que ayudarán en la configuración de las especificaciones generadas para los casos de prueba, es un paquete transversal en la aplicación.

*Paquete apsebi.specification.dot*

Este paquete contiene las gráficas que se generan de las especificaciones parciales que se obtiene de los archivos del proyecto Android.

*Paquete apsebi.specification.specAdd*

Este paquete contiene las clases que añade las especificaciones parciales generadas mediante reflexión de la documentación oficial de Android.

*Paquete apsebi.specification.xml*

Este paquete contiene las clases que ayudarán a la generación de los archivos XML de las especificaciones.

*Paquete apsebi.testcases*

Este paquete contiene las clases que ayudan a combinar las propiedades de las vulnerabilidades, las especificaciones parciales, y el diagrama de clases para que la generación de los casos de prueba ioSTS.

*Paquete apsebi.testcases.intent*

Contiene la clase que ayuda a la generación de los casos de prueba ioSTS pero basado en los intent.

*Paquete apsebi.testcases.verdict*

Este paquete contiene las clases que ayudan a gestionar los resultados de las pruebas, determinando si un componente es vulnerable, no vulnerable o no se puede concluir un resultado.

*Paquete apsebi.vulnerabilities.dot.activity*

Este paquete contiene los patrones de vulnerabilidades (integridad y disponibilidad) para las actividades.

*Paquete apsebi.vulnerabilities.dot.service*

Este paquete contiene los patrones de vulnerabilidades (integridad y disponibilidad) para los servicios.

La pantalla inicial de la aplicación se muestra a continuación

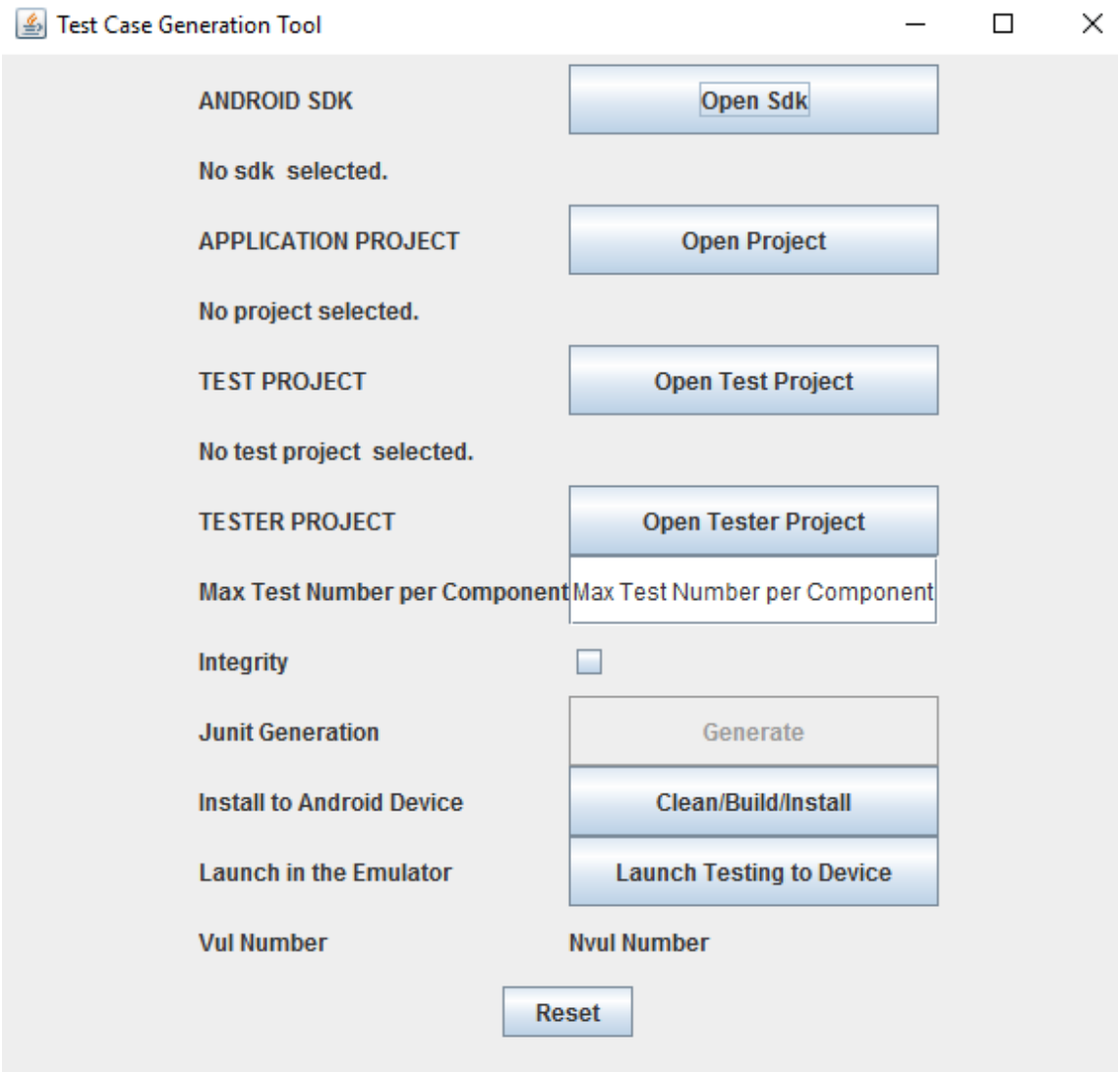



**Gráfico 6-3** Pantalla inicial de la aplicación
**Realizado por:** Villa, H. 2016

## LA HERRAMIENTA PARA TESTEAR LAS APLICACIONES (Testing Tool).

La estructura del proyecto Android que ayuda a testear la aplicación es la siguiente:

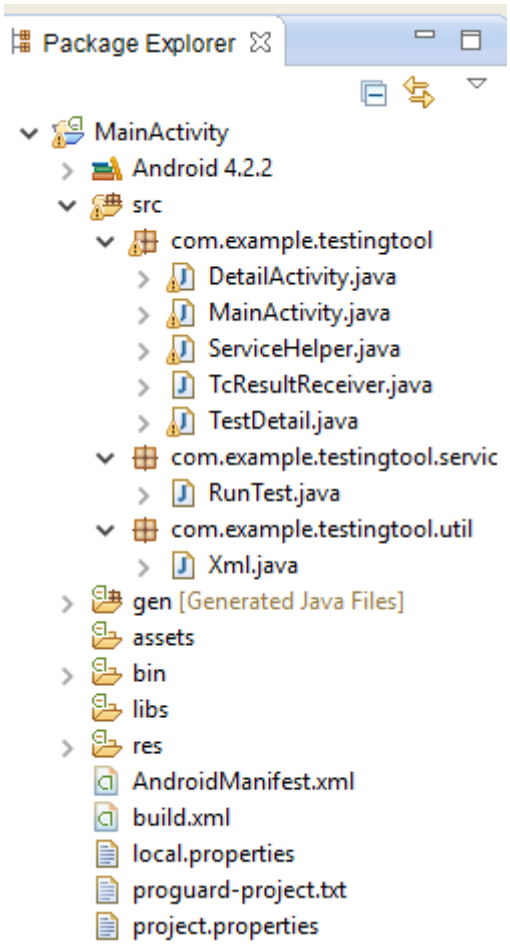



**Gráfico 7-3** Estructura de la aplicación Testing Tool
**Realizado por:** Villa, H. 2016

Está compuesto por 3 paquetes, que están organizados de acuerdo a su funcionalidad, a continuación, se detallan:

*Paquete com.example.testingtool*

Dentro de este paquete se encuentran las clases principales que ayudan a ejecutar las especificaciones creadas anteriormente en el dispositivo Android.

*Paquete com.example.testingtool.service.binder*

Paquete que se encarga de vincular el servicio de ejecución Android con la herramienta de pruebas para la detección de vulnerabilidades

*Paquete com.example.testingtool.util*

Este paquete contiene métodos que servirá para la creación del archivo XML como reporte de las vulnerabilidades encontradas

## PATRONES DE VULNERABILIDAD

Después de describir la arquitectura y el funcionamiento de APSEBI, lo prioritario es conocer cómo escribir los patrones de vulnerabilidad para capturar las vulnerabilidades en las aplicaciones Android.

*NOTACIONES*

Para facilitar la escritura de los patrones de vulnerabilidad se han definido algunas notaciones para determinar el comportamiento de los componentes de la aplicación Android utilizando ioSTS.

**Tabla 1-3** Notaciones

| Notación | Significado |
|---|---|
| $AuthAct_{type}$ | Conjunto de acciones de un determinado tipo de componente |
| $ACT_r$ | Conjunto de las acciones de los intents que requieren una respuesta |
| $ACT_{nr}$ | Conjunto de acciones de los intents que no es requerida una respuesta |
| ***?intent(Cp, a, d, c ,t, ed)*** | Un intent compuesto por:<br>***Cp:*** llamada del componente,<br>***a:*** una acción,<br>***d:*** datos,<br>***c:*** categoría de la acción,<br>***t:*** tipo de dato y,<br>***ed:*** datos extras |
| ***C*** | Conjunto de categorías Android |

| | |
|---|---|
| ***T*** | Conjunto de tipos Android |
| ***U R I*** | Conjunto de URI encontradas en una aplicación y completadas con otras URIs a lazar |
| ***RV*** | Conjunto de valores predefinidos y aleatorios |
| ***I N J*** | Conjunto de inyecciones SQL y XML |
| **!Display** | Lo que muestra y despliega en la pantalla del dispositivo o emulador un intent |
| **!Running** | El servicio que está en ejecución |
| **!ComponentExp** | Excepción creada por un componente |
| **!SystemExp** | Excepción creada por el sistema |
| ***?call(Cp, requ est, tableU R I)*** | El componente ContentProvider llama con una petición a la tabla URI |
| ***?callResp(Cp,resp)*** | El componente ContentProvider responde con ***resp*** el contenido de la respuesta solicitada |

**Realizado por:** Villa, H. 2016

A continuación, se procede a ampliar las notaciones descritas, con el objetivo de entender de mejor manera las notaciones.

Teniendo en cuenta que los componentes se comunican a través de intents, se puede denotar de esta manera

$$intent(Cp, a\ , d, c, t, ed)$$

Siendo:

$Cp$: El componente que se invoca
$a$: La acción que va a ser ejecutada
$d$: Un dato expresado como una URI
$c$: La categoría de un componente de la cual podemos extraer información adicional acerca de la acción que se va a ejecutar.

$t$: Un tipo específico de MIME type de los datos que se envían en el intent
$ed$: Representa los datos extras que se envían con el intent.

Sabiendo que las acciones de los intents tienen diferentes propósitos, por ejemplo

- La acción VIEW permite que una actividad muestre algo en la pantalla,
- La acción PICK permite escoger un ítem de entre las opciones, de esta manera retorna una URI del componente que se está llamando.

Por lo tanto, se ha divido el conjunto de acciones (ACT) en 2 categorías: $ACT_r$ que es el conjunto de acciones que requieren una respuesta y $ACT_{nr}$ que es el conjunto de las otras acciones.

Se denota con *C*, el conjunto de categorías predefinidas en Android y por último con *T* el conjunto de tipos.

Por ejemplo,

*?intent(Act, PICK, content://com.android/contacts, DEFAULT,,)*

Representa la llamada de una actividad Act la cual le permite al usuario escoger un contacto inicialmente guardado en una lista de contactos.

Al momento de ejecutar las pruebas en las aplicaciones los componentes de Android pueden crear excepciones que se agrupan en dos categorías:

- Las excepciones que son creadas por el mismo sistema operativo Android, las mismas que son representadas con *!SystemExp*
- Las excepciones que son creadas debido a la colisión entre componentes, las mismas que son representadas con *!ComponentExp*.

Se denota con *!Display(A)* cuando la acción se despliega en la pantalla lo que produjo la acción *A*.

Cuando un servicio es creado por la ejecución de la acción *A*, el mismo que se ejecutan en background y generalmente retornan datos, se denota con *!Running(A).*

Al usar estas notaciones se puede deducir fácilmente que:

$AuthAct_{Activity}$ es el conjunto de

$$\{?\,\boldsymbol{intent}(\boldsymbol{C_p}, \boldsymbol{a}, \boldsymbol{d}, \boldsymbol{c}, \boldsymbol{t}, \boldsymbol{ed}\,), !\,\boldsymbol{Display}(\boldsymbol{A}), !\,\boldsymbol{System\ Exp}, !\,\boldsymbol{ComponentExp}, !\,\boldsymbol{\delta}\}$$

y además que $AuthAct_{Service}$ es el conjunto de acciones representadas de esta forma

$$\{?\,\boldsymbol{intent}(\boldsymbol{C_p}, \boldsymbol{a}, \boldsymbol{d}, \boldsymbol{c}, \boldsymbol{t}, \boldsymbol{ed}\,), !\,\boldsymbol{Running}(\boldsymbol{A}), !\,\boldsymbol{System\ Exp}, !\,\boldsymbol{ComponentExp}, !\,\boldsymbol{\delta}\}$$

En este sentido los ContentProviders son componentes ligeramente diferentes, debido a que estos componentes no reciben intents por el contrario ellos procesan consultas. Las consultas SQL se denota $?\,call(Cp, request, table\ URI)$ donde *Cp* es el ContentProvider y *request* es la consulta que se realiza sobre la tabla (*tableURI*). Una respuesta emitida por el ContentProvider se representa *!callResp.* Además, como respuesta se puede obtener un cursor que se representa de esta forma $(C_p, cursor)$

De esta manera se puede deducir que $AuthAct_{ContentProvider}$ es el conjunto de $\{?\,\boldsymbol{call}(\boldsymbol{C_p}, \boldsymbol{request}, \boldsymbol{table\ URI}\,), !\,\boldsymbol{callResp}(\boldsymbol{C_p}, \boldsymbol{cursor}), !\,\boldsymbol{\delta}, !\,\boldsymbol{ComponentExp}, !\,\boldsymbol{System\ Exp}\,\}$

Todas las definiciones que se han detallado tienen su correspondiente función codificada en APSEBI, por ejemplo, la acción *!Display(A)* está representada por la función *Display()* la misma que retorna verdadero si una interfaz ha sido mostrada en el dispositivo o emulador.

Esta relación entre las acciones y el código Java hace que sea mucho más sencillo la construcción de los patrones de vulnerabilidad y por ende de los casos de prueba dentro de APSEBI. Además, estas acciones pueden ser actualizadas en cualquier momento en base a las nuevas vulnerabilidades encontradas.

*CREACIÓN DE LOS PATRONES DE VULNERABILIDAD*

En vez de definir una vulnerabilidad de una cada especificación parcial que se genera, es decir, escribir una vulnerabilidad para cada una de las especificaciones parciales creadas, lo que se realiza es definir patrones de vulnerabilidades que engloben en su mayoría las vulnerabilidades de los componentes Android basadas en intents. En términos generales, un patrón de vulnerabilidad describe un conjunto amplio de vulnerabilidades que permitirán evaluar los diferentes componentes de la aplicación. En el caso de que se requiera se puede crear patrones específicos para un determinado componente o aplicación.

Antes que todo se define la notación que se va a utilizar en la creación de los patrones de vulnerabilidad. Un patrón de vulnerabilidad se denota con $V$, el mismo que está compuesto por las acciones que ejecutan los componentes, este conjunto de acciones $\Delta_v$ (acciones del patrón de vulnerabilidad) es igual a $AuthAct_{type}$.

Algunos de los guards están compuestos de predicados específicos que facilitan la escritura de los mismos, por ejemplo:

- *In:* representa una función boleana que retorna verdadero si un parámetro pertenece a un conjunto de valores específicos.
- *Streq:* que retorna verdadero si dos cadenas de caracteres son iguales.

Además, se considera varios valores fijos que ayudan a categorizar si se trata de valores y ataques maliciosos que pueden afectar o dañar el dispositivo.

- *RV*: es un conjunto de valores conocidos para mitigar los bugs y vulnerabilidades
- *INJ*: es un conjunto de inyecciones XML y SQL que se obtiene a través de la tabla URI de la aplicación que se testea.
- *URI*: es un conjunto de las URI encontradas en la aplicación Android, además completada con otras URI de manera aleatoria que han sido creadas anteriormente.

Este conjunto de valores escritos en XML pueden ser modificados en acorde a las diferentes necesidades de los patrones de vulnerabilidad, por ejemplo si la organización OWASP encuentra nuevas vulnerabilidades pues se pueden redefinir estos conjuntos de valores adaptando a lo definido por OWASP.

Además, APSEBI completa este conjunto de valores de acuerdo al análisis del código de la aplicación (Manifest, Clases compiladas).

Es importante mencionar que APSEBI también genera peticiones dinámicas a los diferentes ContentProviders que ayudan a recolectar información de la base de datos para completar el conjunto de valores *INJ*.

Nuevos conjuntos de valores pueden ser definidos dentro de APSEBI, pero con la condición de que se debe mantener los nombres que han sido definidos anteriormente, ya que de esta manera no se crea ambigüedades al momento del análisis de la aplicación.

Los patrones de vulnerabilidades se han escrito específicamente para las actividades y servicios de la aplicación. Estos patrones tratan de vulnerabilidades relacionados a la integridad de los datos que manejan las aplicaciones.

*Patrón 2: Integrity Activities*

Es un patrón sencillo de vulnerabilidad relacionado con la integridad. El objetivo es chequear si una actividad llamada con intents compuestos con datos maliciosos no puede alterar el contenido de una tabla de base de datos manejada por el ContentProvider

*Patrón 2: Integrity Services*

El objetivo es chequear si la estructura de la base de datos manejada por el ContentProvider no es alterado (modificación de los atributos de nombres, eliminación de tablas, etc.) por SQL injection enviado por una actividad que reciba intents.

La principal diferencia entre el patrón de vulnerabilidad (Patrón 2) es en la manera de llamar al ContentProvider ya que lo hace con las tres primeras transiciones con el fin de extraer la estructura de la base de datos que es almacenada en la variable de origen. Después del envío del intent compuesto por SQL injections, el componente no es vulnerable si la estructura de la base de datos es igual a la original (en ese caso se usa un nuevo predicado *equalstructure*).

El patrón de vulnerabilidad de integridad relacionado con las actividades es el siguiente:

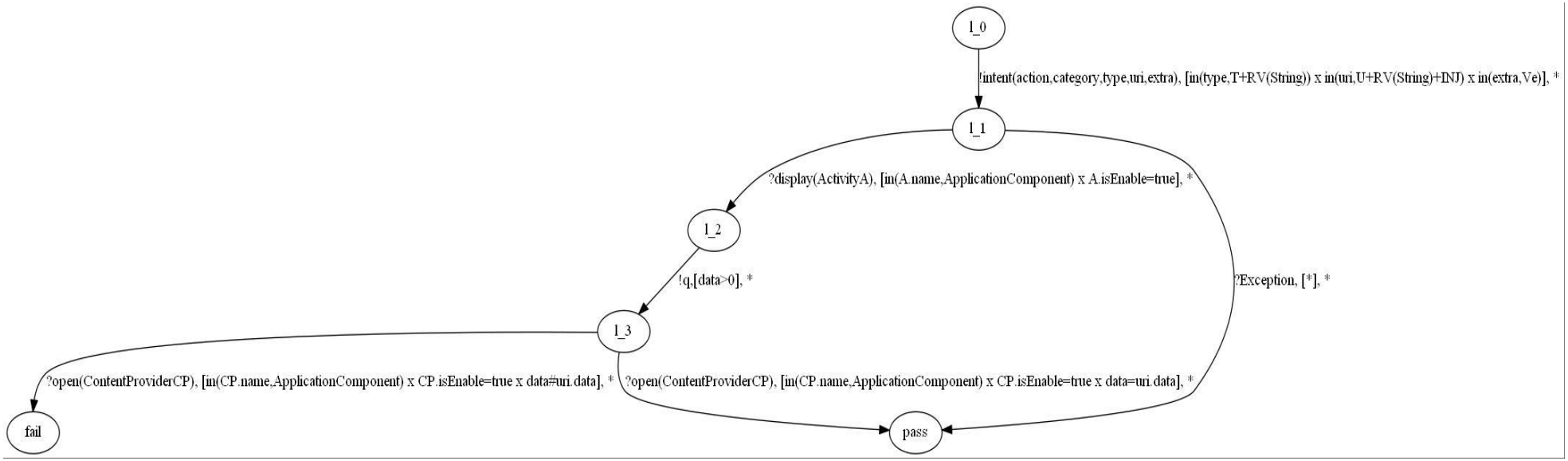



**Gráfico 8-3** Patrón de integridad relacionada con las actividades
**Realizado por:** Villa, H. 2016

El patrón de vulnerabilidad de integridad relacionado con los servicios es el siguiente:

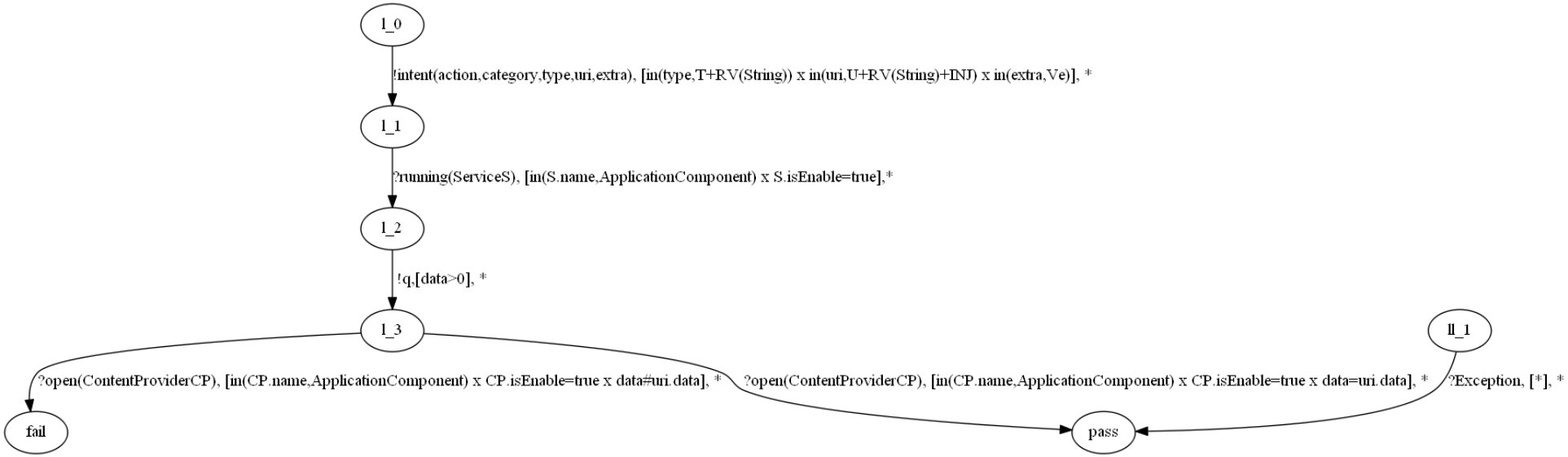



**Gráfico 9-3** Patrón de integridad relacionada con los servicios
**Realizado por:** Villa, H. 2016

## 3.8. Guía de mejores prácticas para la realización de una programación segura en Android

La presente guía contiene lineamientos que deben seguir los desarrolladores de software al momento de crear aplicaciones Android seguras.

Dentro de las características más importantes que podemos destacar de la guía realizada son las siguientes:


**Gráfico 10-3** Características de la Guía de mejores prácticas para la programación segura en Android
**Realizado por:** Villa, H. 2016

- **Sencilla:** Las recomendaciones que se detallan en la Guía son fáciles de entender para los programadores en Android.
- **Práctica:** Debido a que las recomendaciones son sencillas, son fáciles de implementar en el desarrollo de nuevas aplicaciones Android.
- **Escalable:** La Guía puede ser mejorada de acuerdo a las nuevas versiones del sistema operativo Android que salgan al mercado, ya que pueden aparecer nuevas vulnerabilidades relacionadas a los temas planteados en la Guía.

- **Concreta:** Al estructurar los lineamientos de la Guía se ha considerado dos aspectos importantes, la descripción de la vulnerabilidad y el planteamiento de la solución a la misma.

### 3.8.1. No se debe almacenar información confidencial en un dispositivo de almacenamiento externo (tarjeta SD) sin que antes sea cifrado

El sistema operativo Android ofrece diferentes maneras de almacenar los datos de una aplicación determinada, dentro de los cuales existe el almacenamiento externo (/sdcard, /mnt/sd), estas tarjetas externas se adaptan con facilidad al dispositivo móvil ya que en la mayoría existen ranuras para dicho fin.

Al momento de programar para que una aplicación pueda almacenar archivos en la tarjeta externa, desde Android 1 al Android 4.3, únicamente se requiere el permiso de WRITE_EXTERNAL_STORAGE. A partir de Android 4.4, los grupos y modelos de archivos se crean en base a una estructura de directorios que permite a una aplicación manejar los permisos de lectura y escritura de archivos. Además, la estructura de los directorios que se crean se basa en el nombre del paquete de la aplicación. Desde Android 4.4 los usuarios están aislados del almacenamiento externo primario de las otras aplicaciones controladas por Android.

Como se puede notar los archivos almacenados en la tarjeta SD (almacenamiento externo) pueden ser modificados o leídos desde otras aplicaciones instaladas en el dispositivo Android (las versiones de Android con permisos de lectura/escritura) o por cualquier persona que tenga acceso a los archivos del teléfono por ejemplo mediante una PC al conectar el dispositivo vía USB. En consecuencia, de la falta de restricciones descritas anteriormente.

Por tal motivo los desarrolladores no deben almacenar datos sensibles o personales en el almacenamiento externo, a menos que ésta sea encriptada previamente, debido a que almacenar los archivos la tarjeta SD rompe los principios de seguridad de la información (disponibilidad, integridad y confidencialidad).

**Ejemplo de codificación incorrecta**

El siguiente código crea un archivo en el almacenamiento externo (tarjeta SD) y guarda información confidencial en el archivo

```
private String filename = "myfile"

private String string = "sensitive data such as credit card number"
FileOutputStream fos = null;

try {
  file file = new File(getExternalFilesDir(TARGET_TYPE), filename);
  fos = new FileOutputStream(file, false);
  fos.write(string.getBytes());
} catch (FileNotFoundException e) {
  // handle FileNotFoundException
} catch (IOException e) {
  // handle IOException
} finally {
  if (fos != null) {
    try {
    fos.close();
    } catch (IOException e) {
    }
  }
}
```

**Gráfico 11-3** Ejemplo de codificación incorrecta para almacenamiento externo
**Realizado por:** Villa, H. 2016

Generalmente la ruta donde se almacenan los archivos es similar a la siguiente:

```
/sdcard/Android/data/com.company.app/files/save/appdata/save_appdata
```

**Gráfico 12-3** Ruta por defecto de almacenamiento de archivos
**Realizado por:** Villa, H. 2016

Para solucionar este inconveniente se puede tener diferentes soluciones, por ejemplo:

**Primera solución: Guardar el archivo en el almacenamiento interno**

El método openFileOutput() crea el archivo "myfile" en el directorio de la aplicación con la peculiaridad de que es creado con permisos de MODE_PRIVATE para que otras aplicaciones no puedan acceder al archivo.

```
private String filename = "myfile"
private String string = "sensitive data such as credit card number"
FileOutputStream fos = null;

try {
   fos = openFileOutput(filename, Context.MODE_PRIVATE);
   fos.write(string.getBytes());
   fos.close();
} catch (FileNotFoundException e) {
  // handle FileNotFoundException
} catch (IOException e) {
  // handle IOException
} finally {
  if (fos != null) {
    try {
      fos.close();
    } catch (IOException e) {
    }
  }
}
```

**Gráfico 13-3** Solución para guardar archivos en almacenamiento interno
**Realizado por:** Villa, H. 2016

**Segunda solución: Encriptar la información**

Si en el caso de que se tiene que almacenar información en la tarjeta SD (almacenamiento externo) es importante encriptar la información que se almacene. Además, al momento de encriptar la información se debe usar métodos seguros, que no puedan ser descifrados muy fácilmente.

### 3.8.2. Limitar la accesibilidad al proveedor de contenidos para no compartir datos confidenciales entre aplicaciones.

La clase ContentProvider proporciona un mecanismo para gestionar y compartir datos con otras aplicaciones.

Al compartir los datos entre diferentes aplicaciones a través de este componente, se debe mantener un control de acceso muy específico para prohibir el uso no autorizado de datos confidenciales por aplicaciones maliciosas.

Hay tres maneras para limitar (controlar) el acceso al ContentProvider

- Público
- Privado
- Acceso Restringido

**Público**

Al especificar el atributo *android:exported* en el archivo AndroidManifest.xml, el proveedor de contenidos se hace público para otras aplicaciones. Para las aplicaciones Android antes de la versión 16, un proveedor de contenido es por defecto público, a menos que se especifique explícitamente lo contrario mediante esta línea en el archivo AndroidManifest.xml *android:exported="false".* Por ejemplo:

```
<provider android:exported="true" android:name="MyContentProvider" android:authorities="com.example.mycontentprovider" />
```

**Gráfico 14-3** Limitación pública al content provider
**Realizado por:** Villa, H. 2016

Si un proveedor de contenido se debe hacer público, los datos almacenados pueden ser accesados por las todas las aplicaciones sin dificultad alguna. Por lo tanto, está diseñado específicamente para manejar sólo la información no confidencial.

**Privado**

Se puede hacer que el proveedor de contenido sea privado mediante la inclusión del atributo *android:exported* en el archivo AndroidManifest.xml. A partir de la versión 17 y posteriores versiones, un proveedor de contenido es privado si no se especifica el atributo explícitamente. Por ejemplo:

```
<provider android:exported="false" android:name="MyContentProvider" android:authorities="com.example.mycontentprovider" />
```

**Gráfico 15-3** Limitación privada al content provider
**Realizado por:** Villa, H. 2016

Si no desea compartir el proveedor de contenidos con otras aplicaciones, se debe declarar el atributo *android:exported="false"* en el archivo de AndroidManifest.xml. Hay que tomar en cuenta que desde la versión 8 y versiones anteriores, incluso si se declara explícitamente *android:exported="false"*, el proveedor de contenido es accesible desde otras aplicaciones.

**Ejemplo de codificación incorrecta**

Por ejemplo, MovatwiTouch, una aplicación cliente de Twitter, utiliza el proveedor de contenidos para gestionar las credenciales de Twitter (clave, token, información secreta). Sin embargo, el proveedor de contenido de esta aplicación se hizo público, y por tanto todas las aplicaciones instaladas en los dispositivos de los usuarios podían acceder a esta información confidencial.

La siguiente entrada en el AndroidManifest.xml no tiene el atributo *android:exported*, que significa, antes de la versión 16, que el proveedor de contenido se hace público

**AndroidManifest.xml**

```
<provider android:name=".content.AccountProvider" android:authorities="jp.co.vulnerable.accountprovider" />
```

**Gráfico 16-3** Ejemplo de codificación incorrecta de accesibilidad al content provider
**Realizado por:** Villa, H. 2016

El código que se muestra a continuación indica la manera de cómo explotar esta falla de seguridad al momento de programar.

```
// check whether movatwi is installed.
try {
  ApplicationInfo info = getPackageManager().getApplicationInfo("jp.co.vulnerable", 0);[cjl5]
} catch (NameNotFoundException e) {
  Log.w(TAG, "the app is not installed.");
  return;
}
// extract account data through content provider
Uri uri = Uri.parse("content://jp.co.vulnerable.accountprovider");
Cursor cur = getContentResolver().query(uri, null, null, null, null);[cjl6]
StringBuilder sb = new StringBuilder();
if (cur != null) {
  int ri = 0;
  while (cur.moveToNext()) {
    ++ri;
    Log.i(TAG, String.format("row[%d]:", ri));
    sb.setLength(0);
    for (int i = 0; i < cur.getColumnCount(); ++i) {
      String column = cur.getColumnName(i);
      String value = cur.getString(i);
      if (value != null) {
        value = value.replaceAll("[\r\n]", "");
      }
      Log.i(TAG, String.format("\t%s:\t%s", column, value));
    }
  }
} else {
  Log.i(TAG, "Can't get the app information.");
}
```

**Gráfico 17-3** Codificación para explotación de la vulnerabilidad de accesibilidad al content provider

**Realizado por:** Villa, H. 2016

### Solución

Para solucionar este inconveniente de seguridad se debe añadir la entrada antes ya detallada *android:exported="false"* en el archivo AndroidManifest.xml lo que hace al proveedor de contenido privado, para que otras aplicaciones no puedan acceder a los datos

```
<provider android:name=".content.AccountProvider" android:exported="false" android:authorities="jp.co.vulnerable.accountprovider" />
```

**Gráfico 18-3** Solución para la vulnerabilidad de accesibilidad al content provider

**Realizado por:** Villa, H. 2016

### 3.8.3. No permita que el WebView tenga acceso a recurso local confidencial a través del esquema del archivo

La clase de WebView muestra las páginas web como parte del diseño de la actividad. El comportamiento del WebView puede ser personalizado mediante el uso de WebSettings, que se obtiene mediante el siguiente comando WebView.getSettings().

Los principales problemas de seguridad que tiene WebView están relacionados con los métodos setJavaScriptEnabled (), setPluginState (), y setAllowFileAccess ().

**setJavaScriptEnabled()**

Este método indica al WebView que permita la ejecución de JavaScript. Para establecer si es cierto:

```
webview.getWebSettings().setJavaScriptEnabled(true);
```

El valor predeterminado es falso.

**setPluginState()**

Este método indica al WebView la activación, desactivación, o mantener plugins habilitados de acuerdo a la necesidad o demanda.

**Tabla 2-3** Significado de los estados de la clase WebView

| | |
|---|---|
| ON | Cualquier objeto se cargará incluso si no existe un plugin para manejar el contenido |
| ON_DEMAND | Si hay un plugin que puede gestionar el contenido, se muestra un marcador de posición hasta que el usuario hace click en el marcador de posición. Una vez pulsado, el plugin se habilitará. |
| OFF | Todos los plugins se desactivará |

**Realizado por:** Villa, H. 2016

El valor predeterminado es OFF.

**setAllowFileAccess()**

Este método habilita o deshabilita el acceso a los archivos dentro de WebView.

El valor predeterminado es verdadero.

**setAllowContentAccess()**

Este método habilita o deshabilita el acceso al contenido de la URL dentro del WebView. El acceso a contenido de la URL permite que WebView cargue el contenido desde un proveedor de contenido instalado en el sistema.

El valor predeterminado es verdadero.

**Problemas de seguridad para la clase WebView**

Cuando una actividad posee integrado el elemento WebView para mostrar páginas web, cualquier aplicación puede crear y enviar un objeto de tipo Intent con una determinada URI para solicitar que se muestre una página web.

WebView puede reconocer una variedad de esquemas (schemes), entre ellos *file:scheme*. Una aplicación maliciosa puede crear y almacenar un contenido "elaborado" en el almacenamiento interno del dispositivo, después hace que sea accesible mediante el permiso de *MODE_WORLD_READABLE*, y envía la URI (usando el *file:scheme*) que direcciona al contenido malicioso a la actividad objeto del ataque. La actividad objeto del ataque renderiza mediante el WebView el contenido que el atacante elaboró.

Cuando la actividad objeto del ataque (objeto WebView) activa la ejecución del código JavaScript, mediante el código antes elaborado, se pueden aprovechar de ello para acceder a los recursos de la aplicación objeto del ataque.

Android 4.1 proporciona métodos adicionales para controlar el acceso al *file:scheme*:

- WebSettings#setAllowFileAccessFromFileURLs
- WebSettings#setAllowUniversalAccessFromFileURLs

**Ejemplo de codificación incorrecta**

En el siguiente ejemplo se usa el componente de WebView con JavaScript habilitado, y éste procesa las URI, que vienen de la actividad anterior, sin ninguna validación:

```
public class MyBrowser extends Activity {
  @override
  public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    WebView webView = (WebView) findViewById(R.id.webview);


    // turn on javascript
    WebSettings settings = webView.getSettings();
    settings.setJavaScriptEnabled(true);

    String turl = getIntent().getStringExtra("URL");
    webView.loadUrl(turl);
  }
}
```

**Gráfico 19-3** Ejemplo de codificación incorrecta de la vulnerabilidad del acceso del WebView

**Realizado por:** Villa, H. 2016

El siguiente código muestra como la vulnerabilidad anterior descrita puede ser explotada:

```
// Malicious application prepares some crafted HTML file,
// places it on a local storage, makes accessible from
// other applications. The following code sends an
// intent to a target application (jp.vulnerable.android.app)
// to make it access and process the malicious HTML file.

String pkg = "jp.vulnerable.android.app";
String cls = pkg + ".DummyLauncherActivity";
String uri = "file:///[crafted HTML file]";
Intent intent = new Intent();
intent.setClassName(pkg, cls);
intent.putExtra("url", uri);
this.startActivity(intent);
```

**Gráfico 20-3** Explotación de la vulnerabilidad de acceso del WebView

**Realizado por:** Villa, H. 2016

**Solución**

Cualquier URI recibida a través de un intent que no sea confiable (fuera de la aplicación) debe ser validado antes de ser mostrado en el componente WebView. Por ejemplo, el siguiente código comprueba una URI recibida y la abre sólo cuando no es un esquema de archivo (*file:scheme*), de esta manera se soluciona uno de los problemas que tiene el componente WebView.

```
String intentUrl = getIntent().getStringExtra("url");
String localUrl = "about:blank";
if (!intentUrl.startsWith("file:")) {
  loadUrl = intentUrl;
}
```

**Gráfico 21-3** Solución para la vulnerabilidad de acceso del WebView
**Realizado por:** Villa, H. 2016

### 3.8.4. No realizar una transmisión tipo broadcast con información sensible mediante un intent implícito

Los componentes principales de las aplicaciones de Android, tales como las actividades, los servicios y los broadcast receivers se activan a través de mensajes, llamados intents. Las aplicaciones pueden comunicarse mediante intents de tipo broadcast para enviar mensajes a múltiples aplicaciones (por ejemplo, notificaciones de eventos). Además, una aplicación puede recibir intents de broadcast enviados por el sistema.

Para trasmitir un intent de tipo broadcast se usa el método *Context.sendBroadcast()* y las aplicaciones que estén interesadas receptarán el intent, dinámicamente se puede registrar en las aplicaciones usando el método *Context.registerReceiver()* al cuál se debe especificar el argumento *intentFilter*, que básicamente filtra que tipo de intent broadcast recibirá la aplicación. Otra forma de registrar que tipo de intents recibirá una aplicación es especificándolo de manera estática en el archivo AndroidManifest.xml. mediante la etiqueta <*receiver*>

El problema de enviar intents de tipo broadcast es que estás emisiones son muy vulnerables a ataques como eavesdropping, ataques de denegación de servicio. Por ejemplo, en el caso del ataque de eavesdropping, alguien de manera maliciosa puede interceptar todas las emisiones (sin algún tipo de firma o certificado) que realice la aplicación y conocer todos los datos que se expongan

Además, si una aplicación maliciosa envía un intent broadcast y lo hace con una alta prioridad entonces reemplazaría a los intents broadcast verdaderos y del sistema, de esta manera realizaría una denegación de servicios o además podría inyectar información corrupta para dañar o robar datos sensibles en las aplicaciones.

También advierte que se puede producir un secuestro de una actividad o de un servicio por el uso de intents implícitos. Es decir, una actividad o servicio malicioso puede interceptar un intent implícito y de esa forma iniciar la actividad o servicio que va a ser usado para robar datos sensibles o provocar la denegación de un servicio.

Es importante destacar que cuando se usa el método *sendBroadcast(),* normalmente cualquier otra aplicación, incluyendo una aplicación maliciosa, puede recibir la transmisión broadcast.

Entonces, se debe restringir quienes reciben los intents mediante la transmisión tipo broadcast. Un modo para restringir a quiénes exactamente les debe llegar la información es utilizar un intent explícito. Un intent explícito puede especificar un componente *(using setComponent(ComponentName))*o una clase *(using setClass(Context, Class))*, de esta manera el componente o la clase específica pueda saber cómo reaccionar al intent explícito. Además, esto se lo puede lograr mediante el uso de permisos, otra manera, desde Android 4.0 se puede restringir el broadcast a una sola aplicación mediante la línea de código *Intent.setPackage().*

También existe otro método y es usar la clase *LocalBroadcastManager.* Al utilizar esta clase la transmisión boradcast del intent no sale del proceso que lo está ejecutando. Se debe recalcar que al usar la clase *LocalBroadcastManager* proporciona varias ventajas en relación al uso de *Context.sendBroadcast(Intent)*:

- Usted sabe que los datos que este transmitiendo no saldrán de su aplicación, por tal motivo no es necesario preocuparse de la fuga de datos sensibles.
- Para otras aplicaciones no es posible enviar otros tipos de broadcast a su aplicación, de esta manera por eso no es necesario preocuparse de tener huecos de seguridad que los atacantes puedan aprovechar.
- Es más eficiente que enviar una transmisión global a través del sistema.

**Ejemplo de codificación incorrecta**

Este ejemplo de código muestra una aplicación (*com/simple/ServerService.java*) con un método vulnerable *d()* usando un intent implícito *v1* como un argumento a *this.sendBroadcast()* para transmitir el intent. El intent incluye información bien delicada como dirección IP del dispositivo (*local_ip*), el número de puerto (*port*) y la contraseña para conectarse al dispositivo (*código*).

```
public class ServerService extends Service {
  // ...
  private void d() {
    // ...
    Intent v1 = new Intent();
    v1.setAction("com.sample.action.server_running");
    v1.putExtra("local_ip", v0.h);
    v1.putExtra("port", v0.i);
    v1.putExtra("code", v0.g);
    v1.putExtra("connected", v0.s);
    v1.putExtra("pwd_predefined", v0.r);
    if (!TextUtils.isEmpty(v0.t)) {
      v1.putExtra("connected_usr", v0.t);
    }
  }
  this.sendBroadcast(v1);
}
```

**Gráfico 22-3** Codificación incorrecta para la vulnerabilidad trasmisión de tipo broadcast con un intent implícito

**Realizado por:** Villa, H. 2016

Una aplicación podría recibir el mensaje broadcast usando un broadcast receiver de la siguiente manera:

```
final class MyReceiver extends BroadcastReceiver {
  public final void onReceive(Context context, Intent intent) {
    if (intent != null && intent.getAction() != null) {
      String s = intent.getAction();
      if (s.equals("com.sample.action.server_running") {
        String ip = intent.getStringExtra("local_ip");
        String pwd = intent.getStringExtra("code");
        String port = intent.getIntExtra("port", 8888);
        boolean status = intent.getBooleanExtra("connected", false);
      }
    }
  }
}
```

**Gráfico 23-3** Codificación para recibir el broadcast usando un broadcast receiver
**Realizado por:** Villa, H. 2016

Conociendo esto, un atacante puede implementar un broadcast receiver para interceptar los intents implícitos enviados por una aplicación vulnerable como la anteriormente desarrollada, y de esta manera hacerse de la información sensible transmitida:

```
public class BcReceiv extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent){

    String s = null;
    if (intent.getAction().equals("com.sample.action.server_running")){
      String pwd = intent.getStringExtra("connected");
      s = "Airdroid  => [" + pwd + "]/" + intent.getExtras();
    }
    Toast.makeText(context, String.format("%s Received", s),
                   Toast.LENGTH_SHORT).show();
  }
}
```

**Gráfico 24-3** Explotación de la vulnerabilidad transmisión broadcast usando un intent implícito
**Realizado por:** Villa, H. 2016

**Solución**

Si un tipo de intent es transmitido o recibido por la misma aplicación se debe utilizar *LocalBroadcastManager* y de esta manera se asegura que otras aplicaciones no puedan recibir el mensaje de transmisión, lo que reduce el riesgo de fuga de información delicada. En conclusión, en vez de usar Context.sendBroadcast(), use LocalBroadcastManager.sendBroadcast(), como se muestra a continuación:

```
Intent intent = new Intent("my-sensitive-event");
intent.putExtra("event", "this is a test event");
LocalBroadcastManager.getInstance(this).sendBroadcast(intent);
```

**Gráfico 25-3** Solución para la vulnerabilidad transmisión broadcast usando un intent implícito
**Realizado por:** Villa, H. 2016

### 3.8.5. No registre información sensible o delicada

Android provee las facilidades para registrar logs de las diferentes aplicaciones, por ejemplo las aplicaciones pueden enviar información al log de salida mediante la clase *Android.util.Log.* Para obtener todos los logs se debe utilizar el comando *logcat.*

La clase android.util.log permite varias posibilidades:

| | | |
|---|---|---|
| Log.d (Debug) | Log.e (Error) | |
| Log.i (Info) | Log.v (Verbose) | Log.w (Warn) |

**Gráfico 26-3** Alternativas de la clase Android.util.log
**Realizado por:** Villa, H. 2016

Ejemplo:

```
Log.v("method", Login.TAG + ", account=" + str1);
Log.v("method", Login.TAG + ", password=" + str2);
```

**Gráfico 27-3** Ejemplo de las alternativas de la clase Android.util.log
**Realizado por:** Villa, H. 2016

*Para obtener el registro de salida*

Para obtener este registro se debe declara el permiso de READ_LOGS en el archivo de manifiesto, para que la aplicación pueda leer el archivo de salida.
AndroidManifest.xml

```
<uses-permission android:name="android.permission.READ_LOGS"/>
```

**Gráfico 28-3** Manera para obtener el registro de salida
**Realizado por:** Villa, H. 2016

Para llamar al *logcat* desde una aplicación se lo realiza de la siguiente manera:

```
Process mProc = Runtime.getRuntime().exec(
    new String[]{"logcat", "-d", "method:V *:S$Bc`W^(B)"});

BufferedReader mReader = new BufferedReader(
    new InputStreamReader(proc.getInputStream()));
```

**Gráfico 29-3** Invocar al logcat desde una aplicación
**Realizado por:** Villa, H. 2016

Antes de Android 4.0, cualquier aplicación con el permiso de *READ_LOGS* podía obtener todos los registros de salida de todas las aplicaciones. Después de Android 4.1, la especificación del permiso *READ_LOGS* ha sido cambiada, ya que ahora las aplicaciones con este permiso ya no pueden obtener este registro.

Sin embargo, cuando se conecta el dispositivo Android a una PC, se puede obtener el registro de salida de las aplicaciones sin mayor dificultad, por tal motivo es importante que las aplicaciones no envíen información delicada a dicho registro

**Ejemplo de codificación incorrecta**

El SDK de Facebook para Android contiene el siguiente código que envía los tokens de acceso de Facebook al registro de salida en formato de texto plano.

```
Log.d("Facebook-authorize", "Login Success! access_token="
      + getAccessToken() + " expires="
      + getAccessExpires());
```

**Gráfico 30-3** Ejemplo de codificación incorrecta relacionada a la vulnerabilidad registra información sensible
**Realizado por:** Villa, H. 2016

Además, por ejemplo, se puede observar un informe meteorológico generado por Android en el que se observa que se registra en el log la ubicación del usuario que generó este reporte.

```
I/MyWeatherReport( 6483): Re-use MyWeatherReport data
I/ ( 6483): GET JSON: http://example.com/smart/repo_piece.cgi?
arc=0&lat=26.209026&lon=127.650803&rad=50&dir=-999&lim=52&category=1000
```

**Gráfico 31-3** Ejemplo de un registro de salida vulnerable
**Realizado por:** Villa, H. 2016

Si un usuario está utilizando la versión de Android 4.0 o anteriores, otras aplicaciones con permiso de READ_LOGS pueden obtener la información de la ubicación del usuario sin declarar el permiso ACCESS_FINE_LOCATION en el archivo de manifiesto.

**Solución**

El siguiente ejemplo muestra la manera de obtener el registro de salida (log) de una aplicación vulnerable es el siguiente:

```
final StringBuilder slog = new StringBuilder();

try {
  Process mLogcatProc;
  mLogcatProc = Runtime.getRuntime().exec(new String[]
      {"logcat", "-d", "LoginAsyncTask:I APIClient:I method:V *:S" });

  BufferedReader reader = new BufferedReader(new InputStreamReader(
      mLogcatProc.getInputStream()));

  String line;
  String separator = System.getProperty("line.separator");

  while ((line = reader.readLine()) != null) {
    slog.append(line);
    slog.append(separator);
  }
  Toast.makeText(this, "Obtained log information", Toast.LENGTH_SHORT).show();

} catch (IOException e) {
  // handle error
}

TextView tView = (TextView) findViewById(R.id.logView);
tView.setText(slog);
```

**Gráfico 32-3** Solución a la vulnerabilidad registra información sensible
**Realizado por:** Villa, H. 2016

Al momento de programar se debe asegurar que las aplicaciones no envíen (registren) información confidencial al registro de salida (log). Si la aplicación incluye una biblioteca de terceros, el desarrollador debe asegurarse de que la biblioteca no envíe información confidencial al registro de salida (log). Una solución común al desarrollar una aplicación es la de declarar y utilizar una clase de registro personalizado, de modo que este registro sea automáticamente activado/desactivado basado en la forma de ejecución debug/release. Se recomienda usar ProGuard para borrar llamadas a métodos específicos.

En conclusión, se necesario no registrar información confidencial fuera un límite de confianza.

### 3.8.6. No conceder permisos URI en intents implícitos

Los datos almacenados en el proveedor de servicio (service provider) de una aplicación pueden ser referenciados por identificadores URI que se incluyen en los intents. Si el receptor del intent no tiene los privilegios necesarios para acceder a la URI, el remitente del intent puede establecer cualquiera de las siguientes banderas: FLAG_GRANT_READ_URI_PERMISSION o FLAG_GRANT_WRITE_URI_PERMISSION en el intent.

Si el remitente ha especificado en el archivo de manifiesto que los permisos de URI pueden concederse, entonces el receptor de la intención será capaz de leer o escribir (respectivamente) datos en la URI.

Si un componente malicioso es capaz de interceptar el intent, entonces se puede acceder a los datos en la URI. Los intents implícitos pueden ser interceptados por cualquier componente de modo que, si se desea que los datos sean privados se debería utilizar intents explícitos antes que intents implícitos. Por tanto, una aplicación puede interceptar la intención implícita y tener acceso a los datos en el URI.

### 3.8.7. No actúe sobre maliciosos intents

Una aplicación maliciosa podría enviar un intent a un componente exportado que no está esperando recibirlo, lo que puede lograr es engañar a la aplicación receptora para que realice una acción o acciones inapropiadas con resultados no deseados. Entonces si el componente no va a ser público, pero va a ser exportado, se podría exponer a un ataque.

Si un Broadcast Receiver exportado confía ciegamente en un broadcast intent, este puede realizar acciones inapropiadas como por ejemplo operar con datos sensibles de la aplicación o también puede corromperlo. Los Receivers a menudo envían datos y comandos a los Servicios y Actividades, si este es el caso puede provocar que el intent malicioso se propague en toda la aplicación.

Los componentes registrados para recibir broadcast intents con acciones del sistema (DEVICE_STORAGE, BATTERY_LOW, ACTION_POWER_CONNECTED,

ACTION_SHUTDOWN, entre otras) son particularmente vulnerables a este tipo de ataques. Los intents que informan a las aplicaciones sobre los eventos del sistema son transmitidos mediante broadcast por el sistema operativo. Algunas de los acction string contenidas en los intents sólo pueden ser añadidas por el sistema operativo. Cuando un componente se registra para recibir broadcast del sistema operativo, este Broadcast Receiver se convierte automáticamente de acceso público, de esta manera una aplicación maliciosa puede enviar un intent explícito a la dirección del Receiver (sin incluir el acction string del sistema operativo). Sin embargo, si el Receiver no verifica la identidad del autor de la llamada, entonces se puede realizar acciones que sólo el sistema operativo puede realizar.

Las actividades exportadas pueden ser lanzadas por otras aplicaciones con intents explícitos o implícitos. Este ataque es similar al cross-site request forgeries (CSRF) en sitios web.

**Solución**

Se debe limitar la exposición del componente mediante el establecimiento de requisitos de permiso en el manifiesto o dinámicamente controlando la identidad del autor de la llamada.

### 3.8.8. Proteger los servicios exportados con fuertes permisos

Si un servicio es exportado y no es protegido con fuertes permisos, cualquier aplicación puede iniciar y comprometer el servicio. Dependiendo de las funciones de un servicio específico, se podría filtrar información delicada o realizar tareas no autorizadas. Los servicios muchas de las veces mantienen un estado singleton dentro de la aplicación, lo cual provoca que se a veces mantienen el estado de la aplicación, que podría estar dañado.

Para protegerse contra tales eventualidades, al momento de exportar un servicio siempre debe estar protegido con fuertes permisos.

**Solución**

Establecer permisos fuertes en el archivo de manifiesto de la aplicación al momento de exportar un servicio. Estos permisos deben estar relacionados directamente a la actividad que vaya a realizar el servicio.

### 3.8.9. Limitar el acceso a actividades delicadas.

En Android, al declarar un *intent filter* para una actividad en el AndroidManifest.xml significa que la actividad puede ser exportada para ser usada por otras aplicaciones. Si la actividad está destinada únicamente para el uso interno de la aplicación que se desarrolla y se declara un *intent filter*, entonces cualquier otra aplicación, incluyendo malware, puede activar la actividad para un uso no deseado.

Por ejemplo, tomando el caso de la vulnerabilidad encontrada en la aplicación Twicca (en versiones desde 0.7.0 a la 0.9.30), sucede que al lanzar la actividad de Twicca, otra aplicación que no tiene permiso para acceder a la tarjeta externa (SD card) o a la red, haciendo uso de la actividad lanzada de Twicca, por tanto, la aplicación puede subir imágenes o videos grabados en la tarjeta externa a una red social con la cuenta de usuario Twitter de Twicca.

**Ejemplo de codificación incorrecta**

Se muestra un archivo AndroidManifest.xml para una aplicación que exporta la actividad hacia otras aplicaciones, pero no restringe el acceso a una actividad delicada:

AndroidManifest.xml

```
<activity android:configChanges="keyboard|keyboardHidden|orientation"
  android:name=".media.yfrog.YfrogUploadDialog"
  android:theme="@style/Vulnerable.Dialog"
  android:windowSoftInputMode="stateAlwaysHidden">
    <intent-filter android:icon="@drawable/yfrog_icon" android:label="@string/YFROG">
        <action android:name="jp.co.vulnerable.ACTION_UPLOAD" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="image/*" />
        <data android:mimeType="video/*" />
    </intent-filter>
</activity>
```

**Gráfico 33-3** Ejemplo de codificación incorrecta de la vulnerabilidad limitar el acceso a actividades limitadas

**Realizado por:** Villa, H. 2016

El *android:name* se refiere al nombre de la clase que implementa esta actividad.

El nombre del paquete es *"jp.co.vulnerable"* el nombre completo de la clase que implementa esta actividad es *jp.co.vulnerable.media.yfrog.YfrogUploadDialog*.

Pero desde el momento que se define el *intent filter*, esta actividad puede ser exportada hacia otras aplicaciones.

**Solución (No exporte la actividad)**

La solución a este problema de seguridad es que la actividad no es exportada:

AndroidManifest.xml

```
<activity android:configChanges="keyboard|keyboardHidden|orientation"
          android:name=".media.yfrog.YfrogUploadDialog"
          android:theme="@style/ VulnerableTheme.Dialog"
          android:windowSoftInputMode="stateAlwaysHidden"
          android:exported="false">
</activity>
```

**Gráfico 34-3** Solución a la vulnerabilidad limitar el acceso a actividades limitadas

**Realizado por:** Villa, H. 2016

Declarando *android:exported="false"* para la actividad que no se desea exportar y este cambio se lo realiza en el archivo de manifiesto *AndroidManifest.xml*, de esta manera esta

actividad es restringida para aceptar solo intents desde adentro de la misma aplicación o desde una aplicación con el mismo ID del usuario.

**Solución (Caso Twicca)**

Esta vulnerabilidad en Twica fue arreglada a partir de la versión 0.9.31. En vez de declarar la actividad *exported="false"* en *AndroidManifest.xml*, Twicca arregló esta vulnerabilidad validando a quien hace la llamada de dicha actividad. En el método *onCreate()* de la clase de la actividad, se añadió el código para controlar si el nombre del paquete de quien hace la llamada es el mismo nombre de la actividad llamada.

jp.r246.twicca.media.yfrog.YfrogUploadDialog

```
public void onCreate(Bundle arg5) {
    super.onCreate(arg5);
    ...
    ComponentName v0 = this.getCallingActivity();
    if(v0 == null) {
        this.finish();
    } else if(!jp.r246.twicca.equals(v0.getPackageName())) {
        this.finish();
        } else {
            this.a = this.getIntent().getData();
            if(this.a == null) {
                this.finish();
            }
            ...
        }
    }
}
```

**Gráfico 35-3** Solución al caso twicca
**Realizado por:** Villa, H. 2016

Un programador Android puede arbitrariamente escoger un nombre del paquete, por tanto, diferentes programadores podrían escoger el mismo nombre del paquete. Por tal motivo, generalmente no es recomendable usar el nombre del paquete para validar a quien llama la actividad. Lo más recomendable es validar el certificado del programador en vez del nombre del paquete.

Sin embargo, considerando los siguientes hechos, la solución de Twicca podría ser lógica y segura para solucionar dicha vulnerabilidad:

- Solo una aplicación con un nombre particular del paquete puede existir en Google Play.
- Si un usuario trata de instalar una aplicación de la cual ya existe el nombre del paquete en el dispositivo, la instalación fallará o sobrescribirá la aplicación instalada anteriormente.

### 3.8.10. No libere aplicaciones que son debuggable.

Android permite que el atributo *android.debuggable* sea establecido con el valor de *true* en el archivo de manifiesto, de esta manera la aplicación puede ser debugged. Por defecto, este atributo es deshabilitado y asignado el valor de *false*, pero podría ser establecido verdadero para ayudar con el proceso de "debugging" durante el desarrollo de la aplicación. Sin embargo, una aplicación nunca debería ser descargada con este atributo puesto en verdadero ya que permite que los usuarios obtengan el acceso a detalles de la aplicación que deberían ser mantenidos seguros. Con el atributo puesto en *true*, los usuarios pueden realizar debug de la aplicación, pero sin el acceso al código de fuente.

**Ejemplo de codificación incorrecta**

Este ejemplo muestra una aplicación que tiene el atributo *android.debuggable* puesto en *true*, permitiendo ser accedida para revelar datos delicados.

```
$ adb shell
shell@android:/ $ run-as com.example.someapp sh
shell@android:/data/data/com.example.someapp $ id
uid=10060(app_60) gid=10060(app_60)
shell@android:/data/data/com.example.someapp $ ls files/
secret_data.txt
shell@android:/data/data/com.example.some $ cat files/secret_data.txt
password=GoogolPlex
account_number=31974286
```

**Gráfico 36-3** Ejemplo de codificación incorrecta de la vulnerabilidad no libere aplicaciones que son debuggable

**Realizado por:** Villa, H. 2016

Claramente, con el atributo *android.debuggable* con el valor de *true*, los datos delicados relacionados a la aplicación pueden ser revelados a cualquier usuario.

**Solución**

Asegurarse de que el atributo *android.debuggable* tenga el valor de *false* antes de que la aplicación sea liberada:

```
android:debuggable="false"
```

**Gráfico 37-3** Solución de la vulnerabilidad no libere aplicaciones que son debuggable
**Realizado por:** Villa, H. 2016

Note que varios ambientes de desarrollo (incluyendo Eclipse/ADT y Ant) automáticamente establecen *android.debuggable* con el valor de *true* para construcciones incrementales o debugging pero establecerlo con el valor de *false* al liberar por ejemplo al Play Store las aplicaciones creadas.

### 3.8.11. Asegurarse de que los datos sensibles sean mantenidos seguros

En las aplicaciones Android, los datos pueden ser transmitidos mediante intents, o los datos pueden ser escritos en archivos, distribuidos utilizando preferencias compartidas o pueden ser guardados en bases de datos. En todos estos casos que se han mencionado, si los datos son sensibles, es importante mantener los datos seguros. Esto es, no debería ser posible para otras aplicaciones o personas (o, más estrictamente, aplicaciones con diferentes ID de usuarios) acceder a este tipo de datos.

La seguridad de los datos (para canales de comunicación que no sean intents) puede ser respaldada mediante la creación de un archivo, asignando a la base de datos el permiso MODE_PRIVATE en el almacenamiento interno o además se puede encriptar (usando técnicas de encriptación seguras o además se puede complementar la seguridad mediante el uso de una llave de encriptación que sólo puede tener la(s) aplicación(es) permitidas) si este archivo o base de datos va a ser guardada en el almacenamiento externo (SD card). El MODE_PRIVATE es un constante definido a través de la clase

*android.content.Context*, que podría ser utilizado como parámetro en los métodos *openFileOutput()*, *getSharedPreferences()*, y *openOrCreateDataBase()* (los cuales son definidos en la clase *android.content.Context*).

**Ejemplo de codificación incorrecta**

Este ejemplo muestra una aplicación que crea un archivo que puede ser leído universalmente, y por lo tanto no seguro.

```
openFileOutput("someFile", MODE_WORLD_READABLE);
```

**Gráfico 38-3** Ejemplo de codificación incorrecta de la recomendación que los datos sensibles sean mantenidos seguros
**Realizado por:** Villa, H. 2016

Cualquier aplicación podría leer el archivo y acceder a cualquier dato guardado en él.

**Solución**

En esta solución el archivo es creado usando MODE_PRIVATE, por eso no puede ser accedido por aplicaciones con el mismo ID del usuario como la aplicación que creó el archivo.

```
openFileOutput("someFile", MODE_PRIVATE);
```

**Gráfico 39-3** Solución para que los datos sensibles sean mantenidos seguros
**Realizado por:** Villa, H. 2016

### 3.8.12. No provea el método de acceso *addJavascriptInterface* en un WebView que podría tener contenido no confiable

Esta regla aplica para las versiones de Jelly Bean y anteriores.

Al permitir a una aplicación usar el método *addJavascriptInterface* con un contenido no confiable dentro de un *WebView* deja la aplicación vulnerable a ataques scripting usando procesos de reflexión y de esa forma acceder a métodos públicos mediante el uso de JavaScript.

El método *addJavascriptInterface(Object, String)* es llamado desde la clase *android.webkit.WebView*, de esta manera los datos delicados y control de aplicación estarían expuestos a ataques de scripting.

**Ejemplo de codificación incorrecta**

Este ejemplo muestra una aplicación que llama el método *addJavascriptInterface()* el mismo que no es seguro si se usa la versión Jelly Bean y anteriores.

```
WebView webView = new WebView(this);
setContentView(webView);
...
class JsObject {
     private String sensitiveInformation;

     ...
     public String toString() { return sensitiveInformation; }

}
 webView.addJavascriptInterface(new JsObject(), "injectedObject");
 webView.loadData("", "text/html", null);
 webView.loadUrl("http://www.example.com");
```

**Gráfico 40-3** Ejemplo de codificación incorrecta del método addJavascriptInterface() dentro de la clase WebView
**Realizado por:** Villa, H. 2016

De la manera que se detalla anteriormente usando JavaScript se puede controlar el host. Usando Java reflection y los permisos de la aplicación se puede acceder a cualquiera de los métodos públicos de un objeto.

**Solución**

La solución a esta vulnerabilidad es abstenerse de llamar al método *addJavascriptInterface()*.

```
WebView webView = new WebView(this);
setContentView(webView);
...
```

**Gráfico 41-3** Solución al uso del método addJavascriptInterface() dentro de la clase WebView
**Realizado por:** Villa, H. 2016

Otra solución es especificar en el archivo de manifiesto de la aplicación que la misma está orientada sólo para versiones de Jelly Bean y anteriores. Para estos niveles API, solo los métodos públicos que son anotados con *JavascriptInterface* pueden ser accedidos desde JavaScript.

```
<manifest>
<uses-sdk android:minSdkVersion="17" />
...

</manifest>
```

**Gráfico 42-3** Solución alternativa al uso del método addJavascriptInterface() dentro de la clase WebView
**Realizado por:** Villa, H. 2016

### 3.8.13. Considere la posibilidad de problemas de privacidad al usar API de geolocalización

La Geolocation API, la cual es especificada por W3C, habilita a los navegadores web la capacidad de acceder a la información referente a la ubicación geográfica del dispositivo de un usuario.

En la especificación de esta API, está explícitamente prohibido que los agentes de usuario envíen información de la ubicación del usuario a los sitios web sin el permiso previo del mismo:

> 4.1 Privacy considerations for implementers of the Geolocation API
>
> User agents must not send location information to Web sites without the express permission of the user. User agents must acquire permission through a user interface, unless they have prearranged trust relationships with users, as described below. The user interface must include the host component of the document's URI [URI]. Those permissions that are acquired through the user interface and that are preserved beyond the current browsing session (i.e. beyond the time when the browsing context [BROWSINGCONTEXT] is navigated to another URL) must be revocable and user agents must respect revoked permissions.
>
> Some user agents will have prearranged trust relationships that do not require such user interfaces. For example, while a Web browser will present a user interface when a Web site performs a geolocation request, a VOIP telephone may not present any user interface when using location information to perform an E911 function.

**Gráfico 43-3** Extracto de la especificación de Geolocation API
**Realizado por:** Villa, H. 2016

Una implementación de correcta de esta API debe adquirir permiso a través una interfaz de usuario antes de enviar la geolocation del usuario al sitio web.
Un ejemplo Javascript para usar Geolocation API es la siguiente:

```
<script>
navigator.geolocation.getCurrentPosition(
  function(position) {
         alert(position.coords.latitude);
         alert(position.coords.longitude);
  },
  function(){
  // error
});
</script>
```

**Gráfico 44-3** Ejemplo de javascript para el uso correcto de Geolocation API
**Realizado por:** Villa, H. 2016

El código Javascript anterior mostrará la ubicación del dispositivo en una pantalla.
Para habilitar la geolocalización en una aplicación usando la clase *WebView*, los siguientes permisos y el uso del paquete *webkit* son necesarios:

**Permisos**

- android.permission.ACCESS_FINE_LOCATION
- android.permission.ACCESS_COARSE_LOCATION
- android.permission.INTERNET

**Paquete webkit**

- WebSettings#setGeolocationEnabled(true)
- WebChromeClient#onGeolocationPermissionsShowPrompt()

De entre los permisos y métodos anteriormente citados, la implementación del método *WebChromeClient#onGeolocationPermissionsShowPrompt()* necesita consideraciones de seguridad. Hay aplicaciones vulnerables y ejemplos de códigos que anulan este método de manera que la localización de un usuario es enviado a los servidores sin el consentimiento del usuario. Con tal implementación, los datos de ubicación del usuario se filtrarán solo visitando estos sitios maliciosos.

**Ejemplo de codificación incorrecta**

Este ejemplo envía la información de geolocalización del usuario sin obtener el permiso del mismo.

```
public void onGeolocationPermissionsShowPrompt(String origin, Callback callback){
        super.onGeolocationPermissionsShowPrompt(origin, callback);
        callback.invoke(origin, true, false);
}
```

**Gráfico 45-3** Ejemplo de codificación incorrecta de la Geolocation API
**Realizado por:** Villa, H. 2016

**Solución**

Esta solución muestra una UI para pedir por el consentimiento del usuario para el envío de la geolocalización del mismo a un determinado servidor web. Dependiendo de la respuesta del usuario, la aplicación puede controlar la transmisión de los datos de geolocalización.

```
public void onGeolocationPermissionsShowPrompt(String origin, Callback callback) {
        super.onGeolocationPermissionsShowPrompt(origin, callback);
        // Ask for user's permission
        // When the user disallows, do not send the geolocation information
}
```

**Gráfico 46-3** Solución para el uso correcto de la Geolocation API
**Realizado por:** Villa, H. 2016

Otra solución a este problema es arreglar previamente la aplicación vulnerable.

```
public void onGeolocationPermissionsShowPrompt(String origin, GeolocationPermissions$Callback callback) {
        super.onGeolocationPermissionsShowPrompt(origin, callback);
        if(MyPreferences.getBoolean("SECURITY_ENABLE_GEOLOCATION_INFORMATION", true)) {
            WebViewHolder.a(this.a).permissionShowPrompt(origin, callback);
        }
        else {
            callback.invoke(origin, false, false);
        }
}
```

**Gráfico 47-3** Solución alternativa para el uso correcto de la Geolocation API
**Realizado por:** Villa, H. 2016

De acuerdo al código anterior, si la configuración de geolocalización del usuario está habilitada, el código mostrará una pantalla para pedir por el permiso del usuario. Si la configuración es deshabilitada, no transmitirá los datos de geolocalización.

### 3.8.14. Verifique correctamente el certificado del servidor en SSL/TLS

Las aplicaciones Android que usan los protocolos SSL/TLS para la comunicación segura deberían verificar correctamente los certificados del servidor. La verificación básica incluye:

- Verificar que el sujeto (CN) del certificado X.509 y el URL coincidan
- Verifique que el certificado este firmado por el CA fidedigno.
- Verifique que la firma sea correcta.
- Verifique que el certificado no este vencido.

Android SDK 4.0 y posteriores ofrece paquetes para implementar funciones que establezcan conexiones de red. Por ejemplo, usando java.net, javax.net, Android.net u

org.apache.http, un programador puede crear conectores de servidor o conexión HTTP. Org.webkit ofrece funciones necesarias para implementar funciones de navegación web.

Un programador tiene la libertad de personalizar su implementación SSL. El programador debería usar correctamente SSL como apropiado al intent de la aplicación y el entorno de las aplicaciones son allí usados. Si el SSL no es usado correctamente, los datos delicados del usuario podrían quizá filtrar a través del canal de comunicación SSL vulnerable.

A continuación, se describe los siguientes patrones del uso inseguro de SSL:

- Confiar en todos los certificados: el programador implementa el interfaz TrustManager así confiará en todo el certificado del servidor (independientemente de quien lo firmó, que es el CN etc.)
- Permitir todos los nombres de host: La aplicación no verifica si el certificado está emitido para el URL al cual el cliente se está conectando. Por ejemplo, cuando un cliente se conecta a example.com, este aceptará un certificado de servidor emitido para some_other_domain.com.
- Mixed_Mode/No SSL: Un programador mezcla conexiones seguras e inseguras en la misma aplicación o no usa SSL para nada.

En Android, usar HttpURLConnection es recomendado para la implementación del cliente HTTP.

**Ejemplo de codificación incorrecta**

El siguiente código implementa una clase MySSLSocketFactory personalizada que hereda javax.net.ssl.SSLContext:

```
public class extends SSLSocketFactory {
    SSLContext sslContext;
public MySSLSocketFactory (KeyStore truststore) throws NoSuchAlgorithmException,
     KeyManagementException,  KeyStoreException, UnrecoverableKeyException {
    super(truststore);
    this.sslContext = SSLContext.getInstance("TLS");
    this.sslContext.init (null, new TrustManager[] {new X509TrustManager() {
            public void checkClientTrusted(X509Certificate[] chain,
                String authType) throws CertificateException
            {
            }
            public void checkServerTrusted(X509Certificate[] chain,
                String authType) throws CertificateException
            {
            }
            public X509Certificate[] getAcceptedIssuers() {
                return null;
            }
        }}, null);
}
    public Socket createSocket() throws IOException {
        return this.sslContext.getSocketFactory().createSocket();
    }
    public Socket createSocket(Socket socket, String host, int port,
      boolean autoClose) throws IOException, UnknownHostException {
        return this.sslContext.getSocketFactory().createSocket(socket, host,
              port, autoClose);
    }
}

 public static HttpClient getNewHttpClient() {
           DefaultHttpClient v6;
           try {
               KeyStore v5 = KeyStore.getInstance(KeyStore.getDefaultType());
               v5.load(null, null);
               MySSLSocketFactory mySSLScoket = new MySSLSocketFactory(v5);
               if(DefineRelease.sAllowAllSSL) {
                   ((SSLSocketFactory)mySSLScoket).setHostnameVerifier
                           (SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER);
               }
               BasicHttpParams v2 = new BasicHttpParams();
               HttpConnectionParams.setConnectionTimeout(((HttpParams)v2),
                      30000);
               HttpConnectionParams.setSoTimeout(((HttpParams)v2), 30000);
               HttpProtocolParams.setVersion(((HttpParams)v2),
                      HttpVersion.HTTP_1_1);
               HttpProtocolParams.setContentCharset(((HttpParams)v2), "UTF-8");
               SchemeRegistry v3 = new SchemeRegistry();
               v3.register(new Scheme("http", PlainSocketFactory.
                      getSocketFactory(), 80));
               v3.register(new Scheme("https", ((SocketFactory)mySSLScoket),
                      443));
               v6 = new DefaultHttpClient(new ThreadSafeClientConnManager
                      (((HttpParams)v2), v3), ((HttpParams)v2));
           }
           catch(Exception v1) {
               v6 = new DefaultHttpClient();
           }
           return ((HttpClient)v6);
}
```

**Gráfico 48-3** Codificación incorrecta para la verificación del certificado de un servidor en SSL/TLS

**Realizado por:** Villa, H. 2016

En el ejemplo anterior, checkClientTrusted() y checkServerTrusted() están anuladas para hacer una implementación vacía de manera que SSLSocketFactory no verifica el certificado SSL. La clase MySSLSocketFactory es usada para crear un ejemplo de HttpClient en otra parte dela aplicación.

sAllowALLSSL, que es un miembro estático de la clase DefineRelease, es inicializado a verdadero en su constructor estático. Esto habilitará el uso de SSLSocketFactory.ALLOW_ALL_HOSTNAME_VERIFIER. Como resultado, la verificación del nombre del host debería tener lugar cuando el establecer una conexión SSL sea deshabilitada y conducirá a la misma situación cuando todo el certificado sea confiable.

**Solución**

La solución compatible podría quizá variar, dependiendo de la actual implementación. Para ejemplos de implementación segura tales como usar un certificado de servidor self_signed.

## 3.9. Definición de los escenarios de pruebas

**Ambiente de pruebas**

Se establece un ambiente de pruebas común en el que comparten los escenarios para el análisis de vulnerabilidades basadas en Intents de aplicaciones Android. Las condiciones del ambiente de pruebas para los 2 escenarios son:

- Aplicación desarrollada en Android
- Dispositivo móvil no rooteado
- Tener el código fuente de la aplicación a ser escaneada (No APK)
- Generar el archivo build.xml de la aplicación a ser escaneada

**Escenarios**

En el ambiente de pruebas se definen dos escenarios:

- **Escenario 1**

En el primer escenario se utilizará el Prototipo I, que utilizará la aplicación desarrollada para escanear las vulnerabilidades basadas en intents de un ejemplo en Android.

- **Escenario 2**

En el segundo escenario se utilizará el Prototipo II, que utilizará la aplicación desarrollada para escanear las vulnerabilidades basadas en intents de un ejemplo en Android, considerando la Guía de mejores prácticas descrita en el punto 3.8.

**Resultados**

Posteriormente se realizan las pruebas en los dos escenarios planteados con la finalidad de demostrar la mejora de la seguridad de las aplicaciones escaneadas.

- Prototipo I que incluye la aplicación a escanearse con el programa desarrollado
- Prototipo II que incluye la aplicación a escanearse con el programa desarrollado, además se considera directrices brindadas en la guía de mejores prácticas para programación en Android.

## 3.10. Hipótesis

### 3.10.1. Determinación de variables

De acuerdo a la hipótesis planteada, luego de se determinan las siguientes variables:

- Aplicación de Android

- Detección de vulnerabilidades

### 3.10.2. Operacionalización Conceptual

**Tabla 3-3** Operacionalización conceptual

| VARIABLE | TIPO | CONCEPTO |
|---|---|---|
| Aplicación de Android | Independiente | Aplicación que ayude a determinar y mitigar las vulnerabilidades |
| Detección de vulnerabilidades | Dependiente | Puntos débiles del software que permiten que un atacante comprometa la integridad, disponibilidad o confidencialidad |

**Realizado por:** Villa, H. 2016

### 3.10.3. Operacionalización Metodológica

**Tabla 4-3** Operacionalización metodológica

| VARIABLE | INDICADOR | TÉCNICA | INSTRUMENTO/FUENTE |
|---|---|---|---|
| Aplicación de Android | - Líneas de código<br>- Recursos utilizados<br>- Patrones | • Investigación de vulnerabilidades<br>• Guía<br>• Pruebas<br>• Observación | • Eclipse Juno<br>• Android SDK<br>• SQLite<br>• Dispositivos<br>• JDK, JRE<br>• ioSTS (input output Symbolic Transition Systems) |
| Detección de Vulnerabilidades | Escala para medición de las vulnerabilidades en la aplicación:<br>- NVUL: no vulnerable<br>- VUL: vulnerable<br>- INC: no se puede llegar a una conclusión | • Pruebas<br>• Análisis<br>• Resultados | • Escenarios de pruebas<br>• Reporte generado |

**Realizado por:** Villa, H. 2016

# CAPÍTULO IV

# RESULTADOS Y DISCUSIÓN

En este capítulo, se desarrollan las pruebas en los escenarios establecidos, se analizan, comparan los resultados obtenidos y se demuestra la hipótesis definida.

## 4.1. Desarrollo de las pruebas

Para el desarrollo de las pruebas se ha seleccionado tres aplicaciones de ejemplo en la documentación oficial de Android, las mismas que están publicadas en el Play Store de Android, a continuación se detalla:

- NotePad
- SearchableDictionary
- WifiDirectDemo

Las aplicaciones para el desarrollo de las pruebas han sido seleccionadas de acuerdo a los criterios de:

- Número de clases que posee la aplicación
- Tipo de aplicación
- Si está basada en la documentación oficial de Android

**Tabla 1-4** Criterios para la selección de las aplicaciones

| Criterios | NotePad | Searchable Dictionary | WifiDirectDemo |
|---|---|---|---|
| **No. Clases** | 6 | 4 | 5 |
| **Tipo de Aplicación** | Aplicación de productividad | Aplicación de productividad | Aplicación de productividad |
| **Basada en documentación oficial Android** | Si | Si | Si |

**Realizado por:** Villa, H. 2016

Con el objetivo de establecer escenarios comunes para el establecimiento de los prototipos y el desarrollo de las pruebas.

### 4.1.1. Prototipo I

En el Prototipo I, se utilizará la aplicación desarrollada para escanear las vulnerabilidades basadas en intents de tres aplicaciones de ejemplo en Android.
Para escanear las vulnerabilidades de la aplicación en Android se utiliza el programa desarrollado en el presente trabajo de investigación.

#### 4.1.1.1. Ejecución de la aplicación desarrollada

**NotePad**

**Tabla 2-4** Información general de la aplicación NotePad

| | |
|---|---|
| **Nombre de la aplicación** | NotePad |
| **Versión** | 8 |
| **Máximo de pruebas por componente** | 10 |
| **Clases** | • NoteEditor.java<br>• NotePad.java<br>• NotePadProvider,java<br>• NotesList.java<br>• NotesLiveFolder.java<br>• TitleEditor.java |

**Realizado por:** Villa, H. 2016

A continuación, se observa la interfaz de la aplicación NotePad

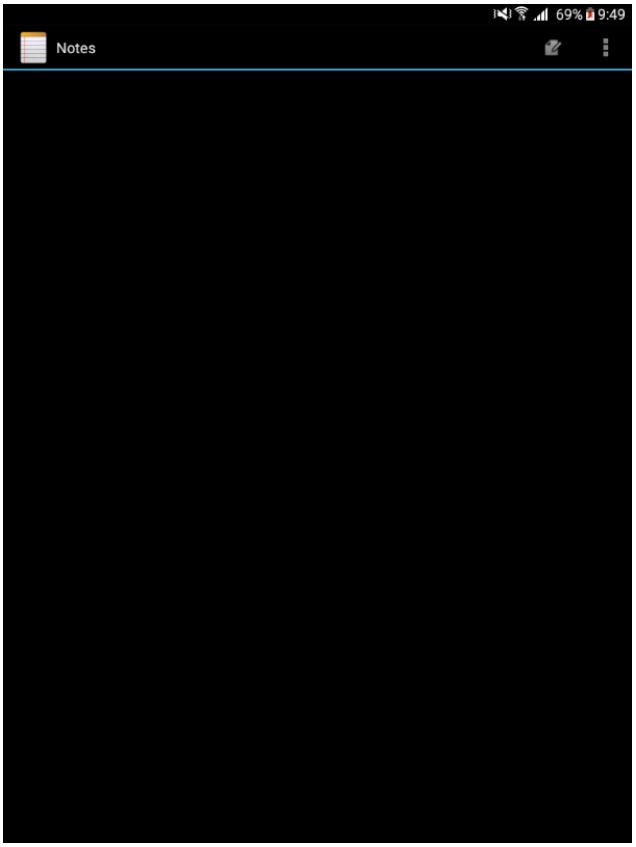



**Gráfico 1-4** Interfaz de la aplicación NotePad
**Realizado por:** Villa, H. 2016

Luego de ejecutar el escaneo de la aplicación NotePad con el programa desarrollado se obtiene el siguiente resultado


**Gráfico 2-4** Resultados del escaneo de la aplicación NotePad
**Realizado por:** Villa, H. 2016

Archivo XML de resultado de una de las pruebas realizadas.

```
<?xml version="1.0" encoding="UTF-8"?>
- <testsuites>
  - <testsuite name="com.example.android.notepad.test.Intent.TitleEditorCPtest"
    package="com.example.android.notepad.test" time="" failures="1" errors="0">
      - <testcase name="testIntegrit" time=""
        classname="com.example.android.notepad.test.Intent.TitleEditorCPtest">
            <failure type="SystemException">VULNERABLE
                com.example.android.notepad.test.Intent.TitleEditorCPtest:INSTRUMENTATION_RESULT:
                shortMsg=java.lang.NullPointerException INSTRUMENTATION_RESULT:
                longMsg=java.lang.NullPointerException: Attempt to invoke virtual method
                'java.lang.String android.net.Uri.getScheme()' on a null object reference
                INSTRUMENTATION_CODE: 0 </failure>
        </testcase>
    </testsuite>
    <system-out/>
    <systeme-err/>
    <properties/>
</testsuites>
```

**Gráfico 3-4** Ejemplo de un resultado del escaneo de la aplicación NotePad
**Realizado por:** Villa, H. 2016

En la tabla a continuación se observa los resultados obtenidos en la aplicación NotePad

**Tabla 3-4** Resultados del escaneo de la aplicación Notepad

| VUL | NVUL | INCONCLUSIVE |
|---|---|---|
| 3 | 41 | 0 |

**Realizado por:** Villa, H. 2016

## Searchable Dictionary

**Tabla 4-4** Información general de la aplicación Searchable Dictionary

| | |
|---|---|
| **Nombre de la aplicación** | Searchable Dictionary |
| **Versión** | 8 |
| **Máximo de pruebas por componente** | 10 |
| **Clases** | • DictionaryDatabase.java<br>• DictionaryProvider.java<br>• SearchableDictionary.java<br>• WordActivity.java |

**Realizado por:** Villa, H. 2016

A continuación, se observa la interfaz de la aplicación Searchable Dictionary


**Gráfico 4-4** Interfaz de la aplicación Searchable Dictionary
**Realizado por:** Villa, H. 2016

Luego de ejecutar el escaneo de la aplicación Searchable Dictionary con el programa desarrollado se obtiene el siguiente resultado

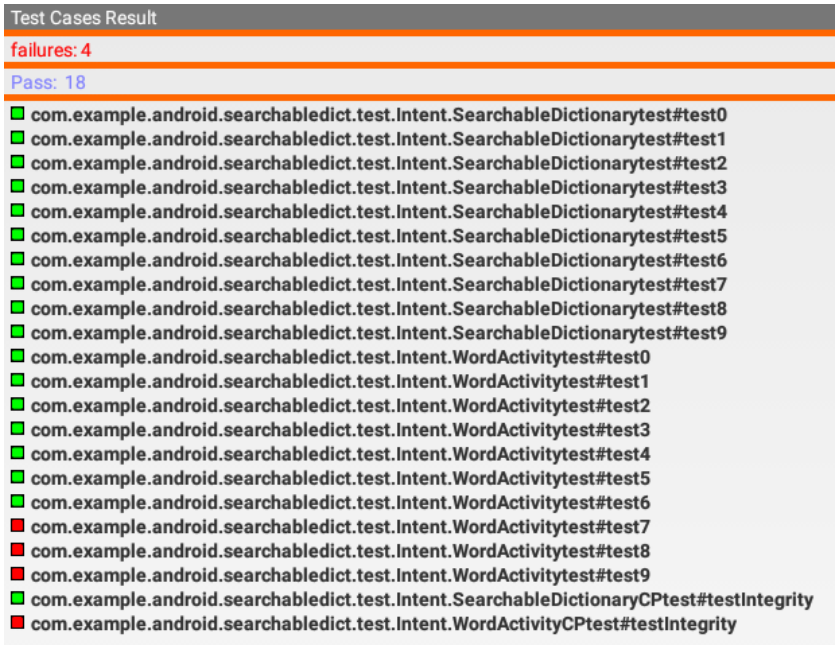



**Gráfico 5-4** Resultados del escaneo de la aplicación Searchable Dictionary
**Realizado por:** Villa, H. 2016

Archivo XML de las pruebas realizadas

```
<?xml version="1.0" encoding="UTF-8"?>
- <testsuites>
  - <testsuite name="com.example.android.searchabledict.test.Intent.WordActivitytest"
    package="com.example.android.searchabledict.test" time="" failures="1" errors="0">
      - <testcase name="test1" time=""
        classname="com.example.android.searchabledict.test.Intent.WordActivitytest">
            <failure type="SystemException">VULNERABLE
                com.example.android.searchabledict.test.Intent.WordActivitytest: Failure in test17:
                junit.framework.AssertionFailedError: VULNERABLE at
                com.example.android.searchabledict.test.Intent.WordActivitytest.test17
                (WordActivitytest.java:1358) at android.test.InstrumentationTestCase.runMethod
                (InstrumentationTestCase.java:214) at android.test.InstrumentationTestCase.runTest
                (InstrumentationTestCase.java:199) at
                android.test.ActivityInstrumentationTestCase2.runTest
                (ActivityInstrumentationTestCase2.java:192) at android.test.AndroidTestRunner.runTest
                (AndroidTestRunner.java:191) at android.test.AndroidTestRunner.runTest
                (AndroidTestRunner.java:176) at android.test.InstrumentationTestRunner.onStart
                (InstrumentationTestRunner.java:555) at
                android.app.Instrumentation$InstrumentationThread.run(Instrumentation.java:1873)
                Test results for PolideaInstrumentationTestRunner=.F Time: 0.102 FAILURES!!! Tests
                run: 1, Failures: 1, Errors: 0 </failure>
        </testcase>
    </testsuite>
    <system-out/>
    <systeme-err/>
    <properties/>
</testsuites>
```

**Gráfico 6-4** Ejemplo de un resultado del escaneo de la aplicación Searchable Dictionary
**Realizado por:** Villa, H. 2016

En la tabla a continuación se observa los resultados obtenidos en la aplicación Searchable Dictionary

**Tabla 5-4** Resultados del escaneo de la aplicación Searchable Dictionary

| VUL | NVUL | INCONCLUSIVE |
|---|---|---|
| 4 | 18 | 0 |

**Realizado por:** Villa, H. 2016

**WifiDirectDemo**

**Tabla 6-4** Información general de la aplicación WifiDirectDemo

| | |
|---|---|
| **Nombre de la aplicación** | WifiDirectDemo |
| **Versión** | 8 |
| **Máximo de pruebas por componente** | 10 |
| **Clases** | • DeviceDetailFragment.java<br>• DeviceListFragment.java<br>• FileTransferService.java<br>• WifiDirectActivity.java<br>• WifiDirectBroadcastReceiver.java |

**Realizado por:** Villa, H. 2016

A continuación, se observa la interfaz de la aplicación WifiDirectDemo

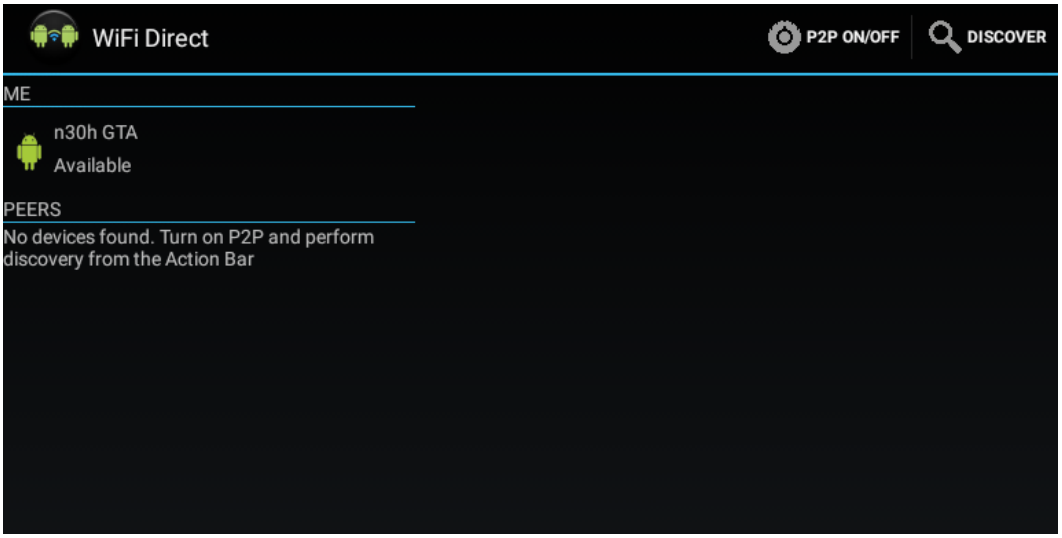



**Gráfico 7-4** Interfaz de la aplicación WifiDirectDemo
**Realizado por:** Villa, H. 2016

Luego de ejecutar el escaneo de la aplicación WifiDirectDemo con el programa desarrollado se obtiene el siguiente resultado

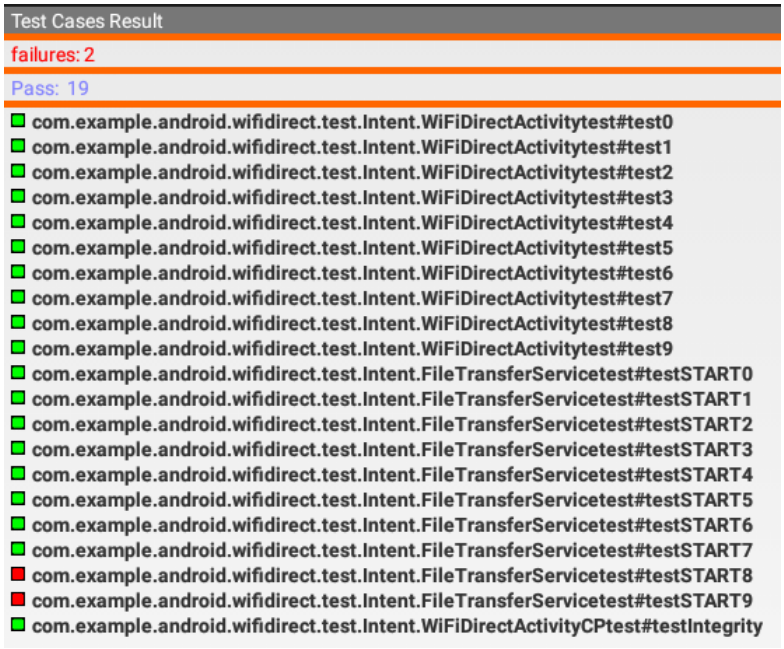



**Gráfico 8-4** Resultados del escaneo de la aplicación WifiDirectDemo
**Realizado por:** Villa, H. 2016

Archivo XML de las pruebas realizadas

```
<?xml version="1.0" encoding="UTF-8"?>
- <testsuites>
  - <testsuite timestamp="2016-02-01T11:35:56" time="0.007" tests="1"
    package="com.example.android.wifidirect.test.Intent"
    name="com.example.android.wifidirect.test.Intent.FileTransferServicetest" failures="1" errors="0">
      - <testcase time="0.007" name="testSTART9"
        classname="com.example.android.wifidirect.test.Intent.FileTransferServicetest">
            <failure>junit.framework.AssertionFailedError: VULNERABLE at junit.framework.Assert.fail
              (Assert.java:50) at
              com.example.android.wifidirect.test.Intent.FileTransferServicetest.testSTART9
              (FileTransferServicetest.java:717) at java.lang.reflect.Method.invoke(Native Method) at
              java.lang.reflect.Method.invoke(Method.java:372) at junit.framework.TestCase.runTest
              (TestCase.java:168) at junit.framework.TestCase.runBare(TestCase.java:134) at
              junit.framework.TestResult$1.protect(TestResult.java:115) at
              junit.framework.TestResult.runProtected(TestResult.java:133) at
              junit.framework.TestResult.run(TestResult.java:118) at junit.framework.TestCase.run
              (TestCase.java:124) at android.test.AndroidTestRunner.runTest
              (AndroidTestRunner.java:191) at android.test.AndroidTestRunner.runTest
              (AndroidTestRunner.java:176) at android.test.InstrumentationTestRunner.onStart
              (InstrumentationTestRunner.java:555) at
              android.app.Instrumentation$InstrumentationThread.run(Instrumentation.java:1873)
              </failure>
        </testcase>
        <properties/>
        <system-out/>
        <system-err/>
    </testsuite>
</testsuites>
```

**Gráfico 9-4** Ejemplo de un resultado del escaneo de la aplicación WifiDirectDemo
**Realizado por:** Villa, H. 2016

En la tabla a continuación se observa los resultados obtenidos en la aplicación WifiDirectDemo

**Tabla 7-4** Resultados del escaneo de la aplicación WifiDirectDemo

| VUL | NVUL | INCONCLUSIVE |
|---|---|---|
| 2 | 19 | 0 |

**Realizado por:** Villa, H. 2016

### 4.1.2. Prototipo II

En el Prototipo II, se utilizará la aplicación desarrollada para escanear las vulnerabilidades basadas en intents de un ejemplo en Android, considerando la ***"Guía de mejores prácticas para la realización de una programación segura en Android"*** descrita en el Capítulo III.

Para escanear las vulnerabilidades de la aplicación en Android se utiliza el programa desarrollado.

#### 4.1.1.1. Ejecución de la aplicación desarrollada

**NotePad**

**Tabla 8-4** Información general de la aplicación NotePad

| | |
|---|---|
| **Nombre de la aplicación** | NotePad |
| **Versión** | 8 |
| **Máximo de pruebas por componente** | 10 |
| **Clases** | • NoteEditor.java<br>• NotePad.java<br>• NotePadProvider,java<br>• NotesList.java<br>• NotesLiveFolder.java<br>• TitleEditor.java |

A continuación, se observa la interfaz de la aplicación NotePad

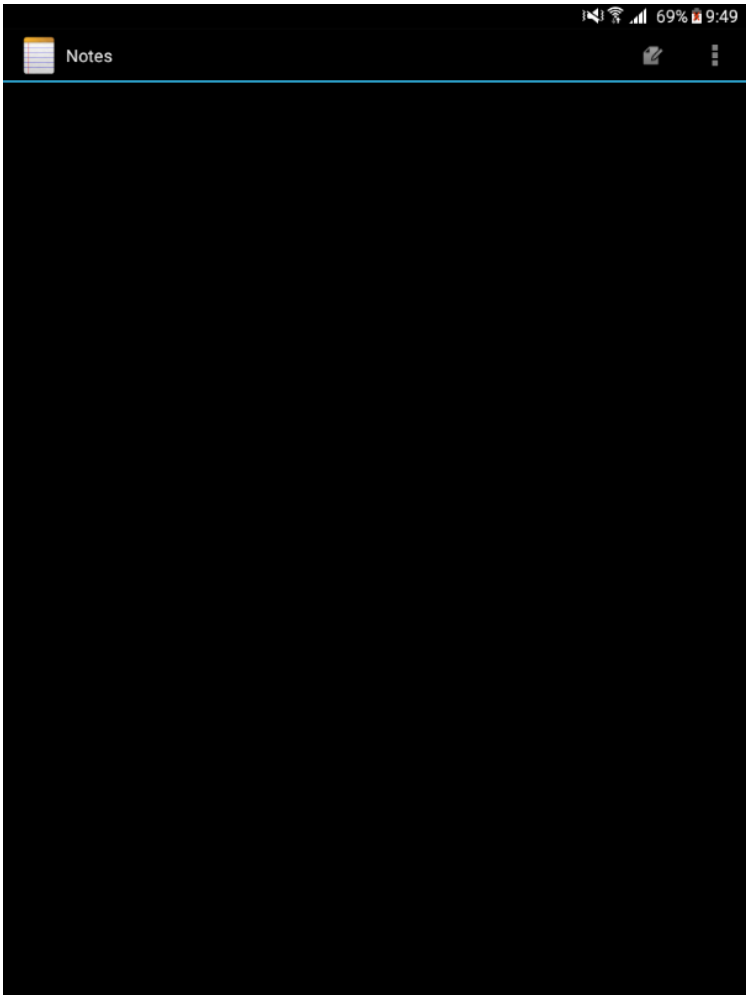



**Gráfico 10-4** Interfaz de la aplicación NotePad
**Realizado por:** Villa, H. 2016

Luego de ejecutar el escaneo de la aplicación NotePad con el programa desarrollado se obtiene el siguiente resultado


**Gráfico 11-4** Resultados del escaneo de la aplicación NotePad
**Realizado por:** Villa, H. 2016

En la tabla a continuación se observa los resultados obtenidos en la aplicación NotePad

**Tabla 9-4** Resultados del escaneo de la aplicación Notepad

| VUL | NVUL | INCONCLUSIVE |
|---|---|---|
| 1 | 43 | 0 |

**Realizado por:** Villa, H. 2016

La disminución de las vulnerabilidades se produce debido a que se han aplicado las siguientes recomendaciones de la **"*Guía de mejores prácticas para la realización de una programación segura en Android*":**

- Limitar la accesibilidad al proveedor de contenidos para no compartir datos confidenciales entre aplicaciones: se ha seguido la recomendación y se ha aumentado la línea de *android:exported="false"*, en el archivo de manifiesto de esta manera no se comparte datos confidenciales entre aplicaciones.

```
<provider android:name="NotePadProvider"
            android:authorities="com.google.provider.NotePad"
            android:exported="false">
            <grant-uri-permission android:pathPattern=".*" />
        </provider>
```

**Searchable Dictionary**

**Tabla 10-4** Información general de la aplicación Searchable Dictionary

| | |
|---|---|
| **Nombre de la aplicación** | Searchable Dictionary |
| **Versión** | 8 |
| **Máximo de pruebas por componente** | 10 |
| **Clases** | • DictionaryDatabase.java<br>• DictionaryProvider.java<br>• SearchableDictionary.java<br>• WordActivity.java |

**Realizado por:** Villa, H. 2016

A continuación, se observa la interfaz de la aplicación Searchable Dictionary


**Gráfico 12-4** Interfaz de la aplicación Searchable Dictionary
**Realizado por:** Villa, H. 2016

Luego de ejecutar el escaneo de la aplicación Searchable Dictionary con el programa desarrollado se obtiene el siguiente resultado

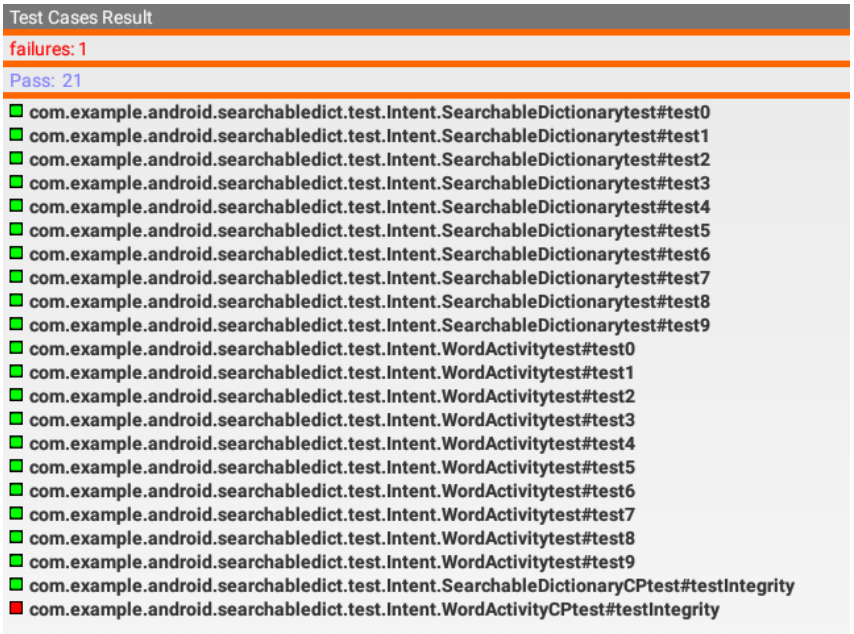



**Gráfico 13-4** Resultados del escaneo de la aplicación Searchable Dictionary
**Realizado por:** Villa, H. 2016

La disminución de las vulnerabilidades se produce debido a que se han aplicado las siguientes recomendaciones de la ***"Guía de mejores prácticas para la realización de una programación segura en Android":***

- Limitar la accesibilidad al proveedor de contenidos para no compartir datos confidenciales entre aplicaciones: se ha seguido la recomendación y se ha aumentado la línea de *android:exported="false"*, en el archivo de manifiesto de esta manera no se comparte datos confidenciales entre aplicaciones.

```
<provider
      android:name=".DictionaryProvider"
      android:authorities="com.example.android.searchabledict.Dic
      tionaryProvider"
       android:exported="false"

/>
```

En la tabla a continuación se observa los resultados obtenidos en la aplicación Searchable Dictionary

**Tabla 11-4** Resultados del escaneo de la aplicación Searchable Dictionary

| VUL | NVUL | INCONCLUSIVE |
|---|---|---|
| 1 | 21 | 0 |

**Realizado por:** Villa, H. 2016

**WifiDirectDemo**

**Tabla 12-4** Información general de la aplicación WifiDirectDemo

| | |
|---|---|
| **Nombre de la aplicación** | WifiDirectDemo |
| **Versión** | 8 |
| **Máximo de pruebas por componente** | 10 |
| **Clases** | • DeviceDetailFragment.java<br>• DeviceListFragment.java<br>• FileTransferService.java<br>• WifiDirectActivity.java<br>• WifiDirectBroadcastReceiver.java |

**Realizado por:** Villa, H. 2016

A continuación, se observa la interfaz de la aplicación WifiDirectDemo

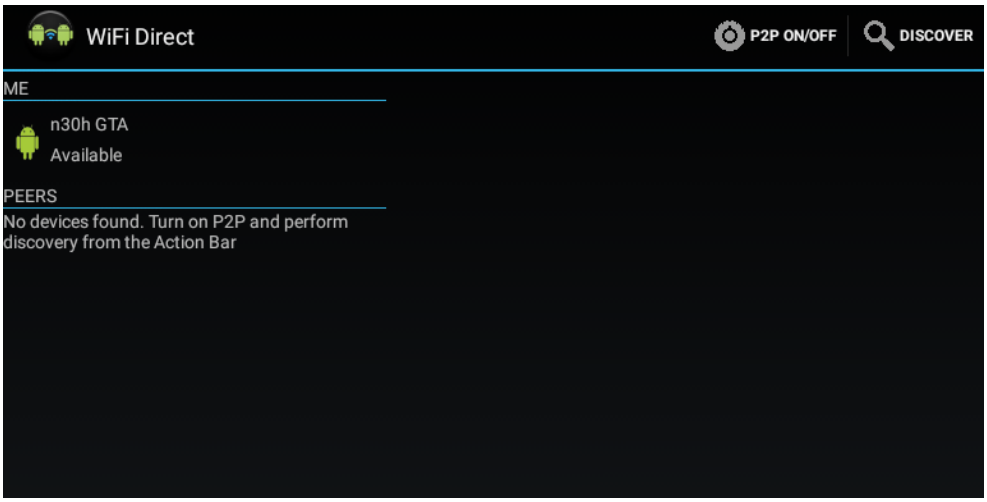



**Gráfico 14-4** Interfaz de la aplicación WifiDirectDemo
**Realizado por:** Villa, H. 2016

Luego de ejecutar el escaneo de la aplicación WifiDirectDemo con el programa desarrollado se obtiene el siguiente resultado

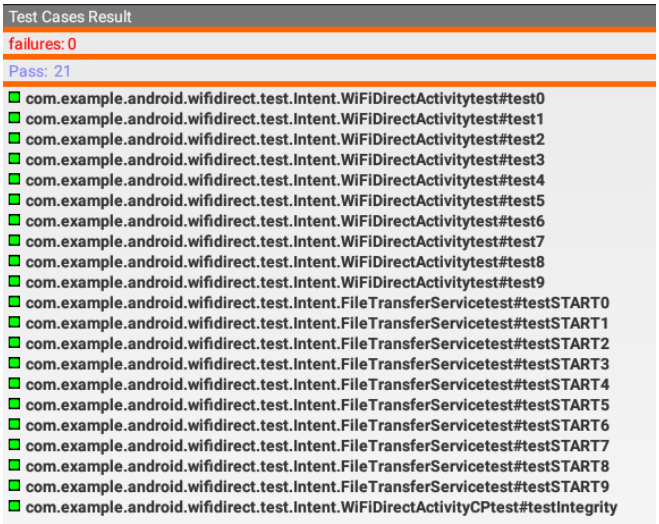



**Gráfico 15-4** Resultados del escaneo de la aplicación WifiDirectDemo
**Realizado por:** Villa, H. 2016

En la tabla a continuación se observa los resultados obtenidos en la aplicación WifiDirectDemo

**Tabla 13-4** Resultados del escaneo de la aplicación WifiDirectDemo

| VUL | NVUL | INCONCLUSIVE |
|---|---|---|
| 0 | 21 | 0 |

**Realizado por:** Villa, H. 2016

La disminución de las vulnerabilidades se produce debido a que se han aplicado las siguientes recomendaciones de la ***"Guía de mejores prácticas para la realización de una programación segura en Android":***

- Limitar el acceso a actividades delicadas: se ha seguido la recomendación y se ha aumentado la línea de *android:exported="false"*, en el archivo de manifiesto de esta manera cualquier aplicación no puede hacer uso no deseado de dicha actividad.

```
<activity
          android:name=".WiFiDirectActivity"
          android:label="@string/app_name"
          android:launchMode="singleTask"
          android:exported="false">
</activity>
```

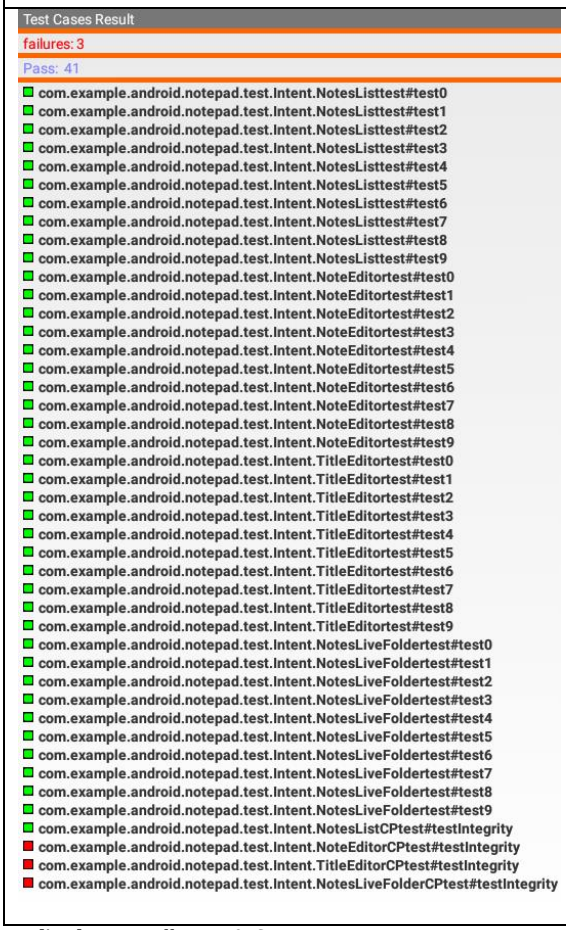
## 4.2. Análisis de resultados

Luego de realizar las pruebas en los escenarios establecidos con los Prototipos desarrollados, se procede a realizar el análisis y comparación de resultados obtenido en cada uno de ellos:

### 4.2.1. Aplicación: Notepad

Se comparan los resultados del escaneo de vulnerabilidades basados en intents utilizando el Prototipo I y el Prototipo II de la aplicación NotePad.

**Tabla 14-4** Comparación de resultados de los prototipos del escaneo de la aplicación NotePad

| NotePad | |
|---|---|
| **Prototipo I** | **Prototipo II** |
|  |  |



**Realizado por:** Villa, H. 2016

**Tabla 15-4** Resultados de los prototipos del escaneo de la aplicación NotePad

| NotePad | | | |
|---|---|---|---|
| **Prototipo** | **VUL** | **NVUL** | **INC** |
| I | 3 | 41 | 0 |
| II | 1 | 43 | 0 |

**Realizado por:** Villa, H. 2016

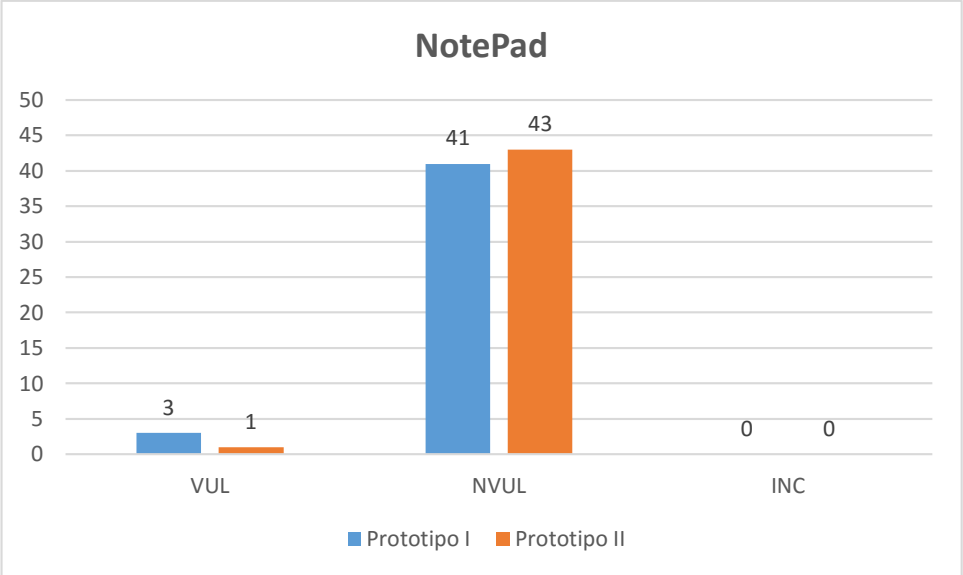



**Gráfico 16-4** Resultados de los prototipos del escaneo de la aplicación NotePad
**Realizado por:** Villa, H. 2016

De acuerdo a los resultados obtenidos al escanear la aplicación con el Prototipo II que incluye el uso de la ***"Guía de mejores prácticas para la realización de una programación segura en Android",*** específicamente aplicando la recomendación de "Limitar la accesibilidad al proveedor de contenidos para no compartir datos confidenciales entre aplicaciones", disminuye las vulnerabilidades basadas en intents en contraste de los resultados obtenidos al escanear la aplicación con el Prototipo I de la aplicación Notepad.

Además se puede observar que existe todavía una vulnerabilidad que no se ha solucionado es debido a que en la guía no se ha considerado la recomendación para solución a dicho problema.

### 4.2.2. Aplicación: Searchable Dictionary

Se comparan los resultados del escaneo de vulnerabilidades basados en intents realizado en los Prototipo I y con el Prototipo II de la aplicación Searchable Dictionary.

**Tabla 16-4** Comparación de resultados de los prototipos del escaneo de la aplicación Searchable Dictionary

| Searchable Dictionary | |
|---|---|
| **Prototipo I** | **Prototipo II** |
| Test Cases Result<br>failures: 4<br>Pass: 18<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test0<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test1<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test2<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test3<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test4<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test5<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test6<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test7<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test8<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test9<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test0<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test1<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test2<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test3<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test4<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test5<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test6<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test7<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test8<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test9<br>com.example.android.searchabledict.test.Intent.SearchableDictionaryCPtest#testIntegr<br>com.example.android.searchabledict.test.Intent.WordActivityCPtest#testIntegrity | Test Cases Result<br>failures: 1<br>Pass: 21<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test0<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test1<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test2<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test3<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test4<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test5<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test6<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test7<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test8<br>com.example.android.searchabledict.test.Intent.SearchableDictionarytest#test9<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test0<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test1<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test2<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test3<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test4<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test5<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test6<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test7<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test8<br>com.example.android.searchabledict.test.Intent.WordActivitytest#test9<br>com.example.android.searchabledict.test.Intent.SearchableDictionaryCPtest#testIntegrity<br>com.example.android.searchabledict.test.Intent.WordActivityCPtest#testIntegrity |

**Realizado por:** Villa, H. 2016

**Tabla 17-4** Resultados de los prototipos del escaneo de la aplicación Searchable Dictionary

| Searchable Dictionary | | | |
|---|---|---|---|
| **Prototipo** | **VUL** | **NVUL** | **INC** |
| I | 4 | 18 | 0 |
| II | 1 | 21 | 0 |

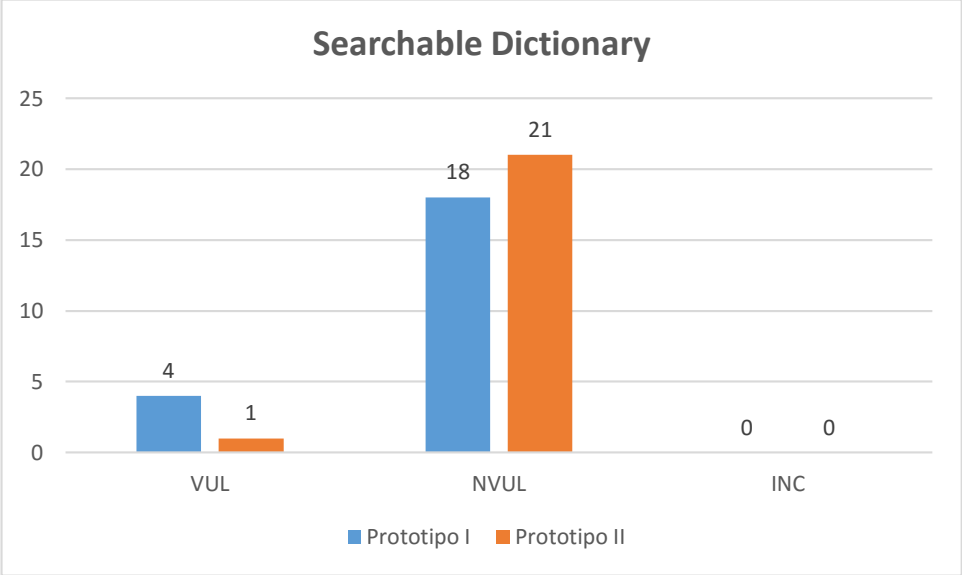



**Gráfico 17-4** Resultados de los prototipos del escaneo de la aplicación Searchable Dictionary

**Realizado por:** Villa, H. 2016

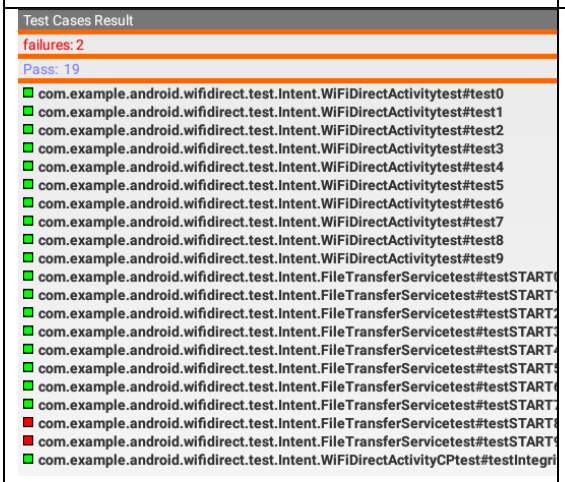
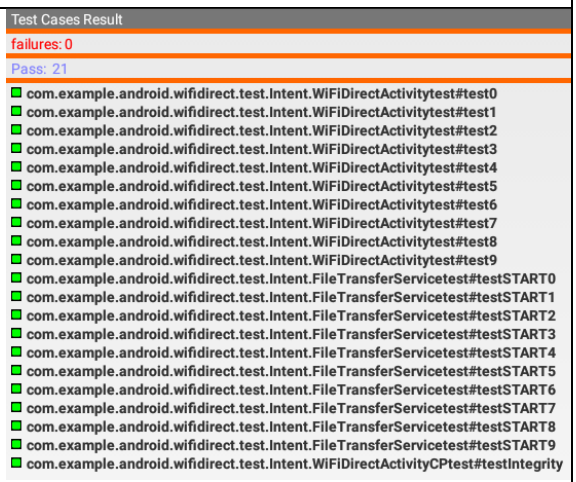
De acuerdo a los resultados obtenidos al escanear la aplicación con el Prototipo II que incluye el uso de la ***"Guía de mejores prácticas para la realización de una programación segura en Android",*** específicamente aplicando la recomendación de "Limitar la accesibilidad al proveedor de contenidos para no compartir datos confidenciales entre aplicaciones", disminuye las vulnerabilidades basadas en intents en contraste de los resultados obtenidos al escanear la aplicación con el Prototipo I de la aplicación Searchable Dictionary.

Además se puede observar que existe todavía una vulnerabilidad que no se ha solucionado es debido a que en la guía no se ha considerado la recomendación para solución a dicho problema.

### 4.2.3. Aplicación: WifiDirectDemo

Se comparan los resultados del escaneo de vulnerabilidades basados en intents realizado en los Prototipo I y con el Prototipo II de la aplicación WifiDirectDemo.

**Tabla 18-4** Comparación de resultados de los prototipos del escaneo de la aplicación WifiDirectDemo

| **WifiDirectDemo** | |
|---|---|
| **Prototipo I** | **Prototipo II** |
|  |  |





**Realizado por:** Villa, H. 2016

**Tabla 19-4** Resultados de los prototipos del escaneo de la aplicación WifiDirectDemo

| WifiDirectDemo | | | |
|---|---|---|---|
| **Prototipo** | **VUL** | **NVUL** | **INC** |
| I | 2 | 19 | 0 |
| II | 0 | 21 | 0 |

**Realizado por:** Villa, H. 2016

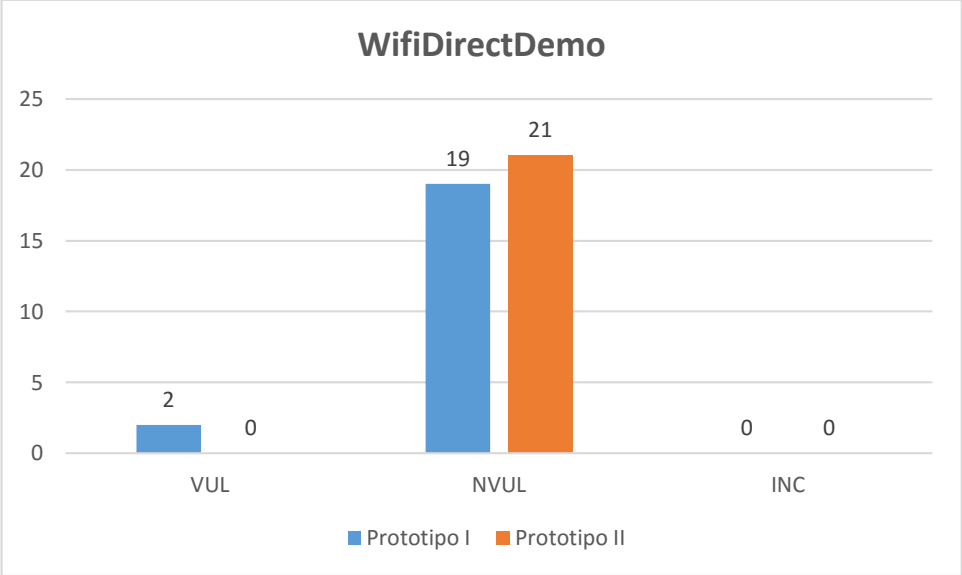



**Gráfico 18-4** Resultados de los prototipos del escaneo de la aplicación WifiDirectDemo

**Realizado por:** Villa, H. 2016

De acuerdo a los resultados obtenidos al escanear la aplicación con el Prototipo II que incluye el uso de la ***"Guía de mejores prácticas para la realización de una programación segura en Android"***, específicamente "Limitar el acceso a actividades delicadas", disminuye las vulnerabilidades basadas en intents en contraste de los resultados obtenidos al escanear la aplicación con el Prototipo I de la aplicación WifiDirectDemo.

A continuación, se muestra un resumen de las pruebas realizadas en los prototipos I y II de las aplicaciones NotePad, Searchable Dictionary y WifiDirectDemo.

**Tabla 20-4** Pruebas realizadas en los prototipos I y II de las aplicaciones escaneadas

| **APLICACIÓN** | **Prototipo I** | | | **Prototipo II** | | |
|---|---|---|---|---|---|---|
| | **VUL** | **NVUL** | **INC** | **VUL** | **NVUL** | **INC** |
| **NotePad** | 3 | 41 | 0 | 1 | 43 | 0 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **Searchable Dictionary** | 4 | 18 | 0 | 1 | 21 | 0 |
| **WifiDirectDemo** | 2 | 19 | 0 | 0 | 21 | 0 |

**Realizado por:** Villa, H. 2016

## 4.3. Prueba de hipótesis

### 4.3.1. Ambiente de pruebas

Para la comprobación de la hipótesis se realizarán 10 casos de prueba por cada componente en las 3 aplicaciones Android, para determinar los siguientes indicadores de la variable dependiente definida:

- Vulnerable: VUL
- No Vulnerable: NVUL
- No se puede llegar a una conclusión: INC

Para medir los indicadores se utilizará los resultados obtenidos con el Prototipo I y el Prototipo II de cada una de las aplicaciones.

Se realizará el escaneo utilizando los IDEs Netbeans y Eclipse:

- En el Prototipo I se analiza la aplicación Android
- En el Prototipo II se analiza la aplicación Android incluyendo las mejores prácticas

### 4.3.2. Escalas de calificación

Para realizar la comparación de los resultados obtenidos se utilizará la escala de Likert para cada uno de los indicadores.

#### 4.3.2.1. Indicador 1: Número de pruebas vulnerables

Para medir el Indicador 1: Número de pruebas vulnerables, se utilizará la escala mostrada en la Tabla 21-4.

**Tabla 21-4** Tabla de escalas para el Indicador 1: Número de pruebas vulnerables

| No. pruebas | Valor |
|---|---|
| 0…1 | 4 |
| 2…3 | 3 |
| 4…6 | 2 |
| >6 | 1 |

**Realizado por:** Villa, H. 2016

La escala definida se basa en que la relación de la seguridad es inversamente proporcional a la cantidad de pruebas con resultados vulnerables de las aplicaciones escaneadas, además se ha tomado en cuenta el número de pruebas que se han realizado a las tres aplicaciones Android relacionados proporcionalmente a los módulos que cada una contiene.

#### 4.3.2.2. Indicador 2: Número de pruebas no vulnerables

Para medir el Indicador 2: Número de pruebas no vulnerables, se utilizará la escala mostrada en la Tabla 22-4.

**Tabla 22-4** Tabla de escalas para el Indicador 2: Número de pruebas no vulnerables

| No. pruebas | Valor |
|---|---|
| >30 | 4 |
| 21…30 | 3 |
| 11…20 | 2 |
| 0…10 | 1 |

**Realizado por:** Villa, H. 2016

La escala definida se basa en que la relación de la seguridad es directamente proporcional a la cantidad de pruebas con resultados no vulnerables de las aplicaciones escaneadas,

tomándose en cuenta el número de pruebas que se han realizado a las tres aplicaciones Android relacionados proporcionalmente a los módulos que cada una contiene.

#### 4.3.2.3. Indicador 3: Número de pruebas indeterminadas

Para medir el Indicador 3: Número de pruebas indeterminadas, se utilizará la escala mostrada en la Tabla 23-4.

**Tabla 23-4** Tabla de escalas para el Indicador 3: Número de pruebas indeterminadas

| No. pruebas | Valor |
|---|---|
| 0 | 4 |
| 1…2 | 3 |
| 3…4 | 2 |
| >4 | 1 |

**Realizado por:** Villa, H. 2016

La escala definida se basa en la naturaleza de las aplicaciones escaneadas, el número de módulos que contienen cada una y el número de pruebas realizadas.

### 4.3.3. Ponderación de indicadores

Los resultados obtenidos en las pruebas para cada Indicador en el punto 4.3.1 son cuantificados con las escalas definidas en el punto 4.3.2.

#### 4.3.3.1. Indicador 1: Número de pruebas vulnerables

Utilizando los valores promedios del Indicador 1: Número de pruebas vulnerables, con cada prototipo se cuantifica los resultados de acuerdo a la escala definida, con lo que se obtiene los valores mostrados en la Tabla 24-4.

**Tabla 24-4** Códigos del Indicador 1: Número de pruebas vulnerables

| No. | Aplicación | Valor | | Código obtenido (de acuerdo a la escala) | |
|---|---|---|---|---|---|
| | | Prototipo I | Prototipo II | Prototipo I | Prototipo II |
| **1** | NotePad | 3 | 1 | 3 | 4 |
| **2** | Searchable Dictionary | 4 | 1 | 2 | 4 |
| **3** | WifiDirectDemo | 2 | 0 | 3 | 4 |

**Realizado por:** Villa, H. 2016

En el Gráfico 19-4 se muestran los códigos obtenidos (de acuerdo a la escala) del Indicador 1: Número de pruebas vulnerables.

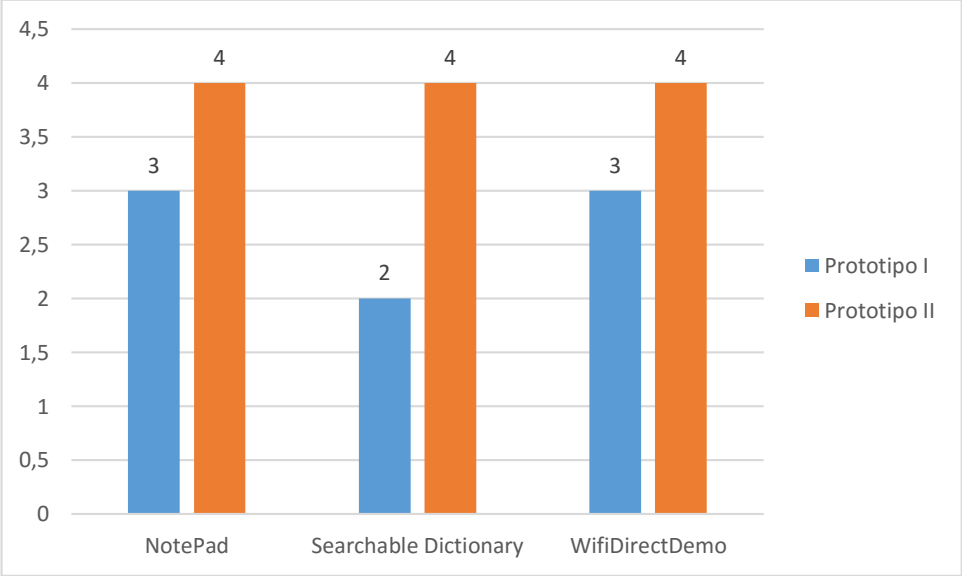



**Gráfico 19-4** Resultados obtenidos (de acuerdo a la escala) del Indicador 1: Número de pruebas vulnerables

**Realizado por:** Villa, H. 2016

### 4.3.3.2. Indicador 2: Número de pruebas no vulnerables

Utilizando los valores promedios del Indicador 2: Número de pruebas no vulnerables, con cada prototipo se cuantifica los resultados de acuerdo a la escala definida, con lo que se obtiene los valores mostrados en la Tabla 25-4.

**Tabla 25-4** Códigos del Indicador 2: Número de pruebas no vulnerables

| No. | Aplicación | Valor | | Código obtenido (de acuerdo a la escala) | |
|---|---|---|---|---|---|
| | | Prototipo I | Prototipo II | Prototipo I | Prototipo II |
| **1** | NotePad | 41 | 43 | 4 | 4 |
| **2** | Searchable Dictionary | 18 | 21 | 2 | 3 |
| **3** | WifiDirectDemo | 19 | 21 | 2 | 3 |

**Realizado por:** Villa, H. 2016

En el Gráfico 20-4 se muestran los códigos obtenidos (de acuerdo a la escala) del Indicador.

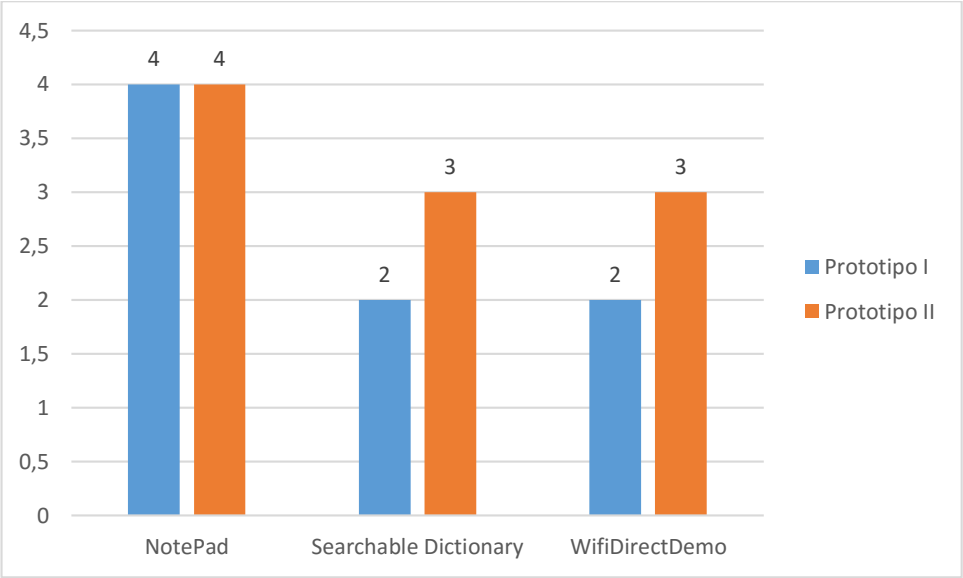



**Gráfico 20-4** Resultados obtenidos (de acuerdo a la escala) del Indicador 2: Número de pruebas no vulnerables

**Realizado por:** Villa Henry, 2015

### 4.3.3.3. Indicador 3: Número de pruebas indeterminadas

Utilizando los valores promedios del Indicador 3: Número de pruebas indeterminadas, con cada prototipo se cuantifica los resultados de acuerdo a la escala definida, con lo que se obtiene los valores mostrados en la Tabla 26-4.

**Tabla 26-4** Códigos del Indicador 3: Número de pruebas indeterminadas

| No. | Aplicación | Valor | | Código obtenido (de acuerdo a la escala) | |
|---|---|---|---|---|---|
| | | Prototipo I | Prototipo II | Prototipo I | Prototipo II |
| 1 | NotePad | 0 | 0 | 4 | 4 |
| 2 | Searchable Dictionary | 0 | 0 | 4 | 4 |
| 3 | WifiDirectDemo | 0 | 0 | 4 | 4 |

**Realizado por:** Villa, H. 2016

En el Gráfico 21-4 se muestran los códigos obtenidos (de acuerdo a la escala) del Indicador.

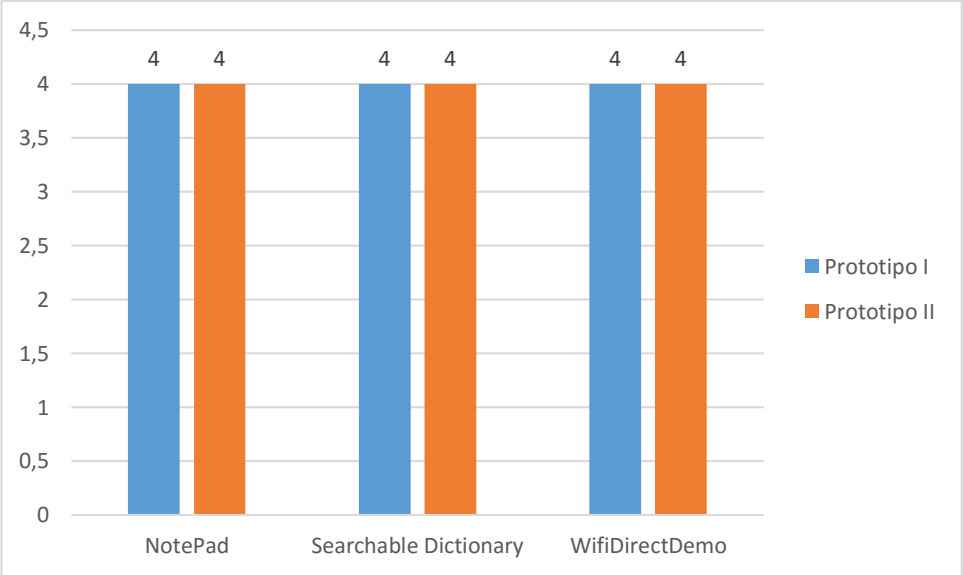



**Gráfico 21-4** Resultados obtenidos (de acuerdo a la escala) del Indicador 3: Número de pruebas indeterminadas

**Realizado por:** Villa, H. 2016

### 4.3.4. Comprobación de hipótesis

La hipótesis definida en el presente trabajo de investigación es "*La propuesta de aplicación para dispositivos móviles Android contribuirá a la detección de vulnerabilidades para mejorar la seguridad en las aplicaciones escaneadas*"

Para la demostración de la hipótesis se utilizará la ***estadística descriptiva*** como punto de partida de la investigación en la que se cuantifican los resultados obtenidos en las pruebas realizadas de cada uno de los indicadores definidos, posteriormente se realizará el proceso de la estadística inferencial.

Una vez obtenido los valores de los indicadores con cada prototipo, se calcula los valores promedios de los resultados de los indicadores como se muestra en la Tabla 27-4

**Tabla 27-4** Resultados de los indicadores con cada prototipo

| APLICACIÓN | Prototipo I | | | Prototipo II | | |
|---|---|---|---|---|---|---|
| | **VUL** | **NVUL** | **INC** | **VUL** | **NVUL** | **INC** |
| **NotePad** | 3 | 4 | 4 | 4 | 4 | 4 |
| **Searchable Dictionary** | 2 | 2 | 4 | 4 | 3 | 4 |
| **WifiDirectDemo** | 3 | 2 | 4 | 4 | 3 | 4 |
| **TOTAL** | **8** | **8** | **12** | **12** | **10** | **12** |

**Realizado por:** Villa, H. 2016

En la se muestran los resultados de la comparación realizada por cada uno de los indicadores

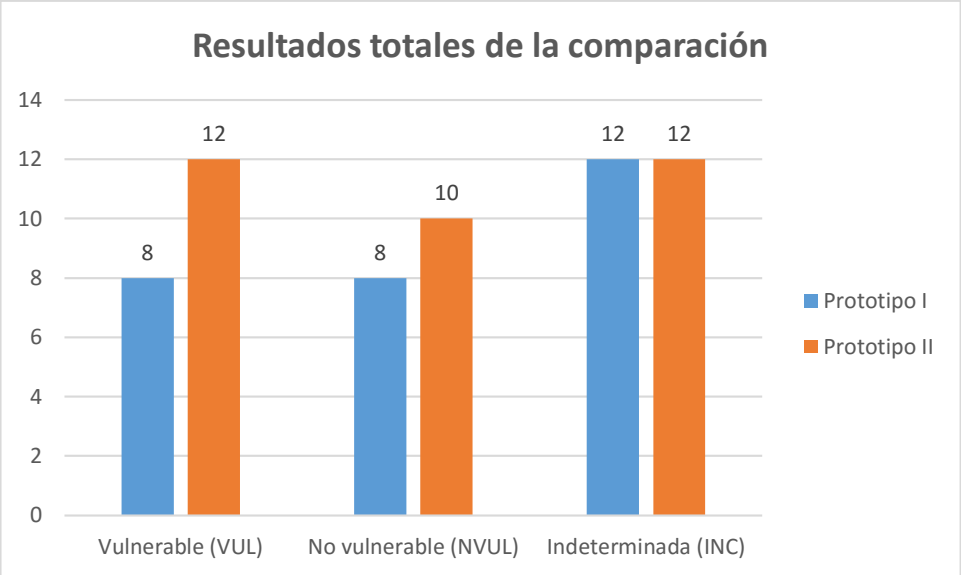



**Gráfico 22-4** Resultados de los indicadores con cada prototipo
**Realizado por:** Villa, H. 2016

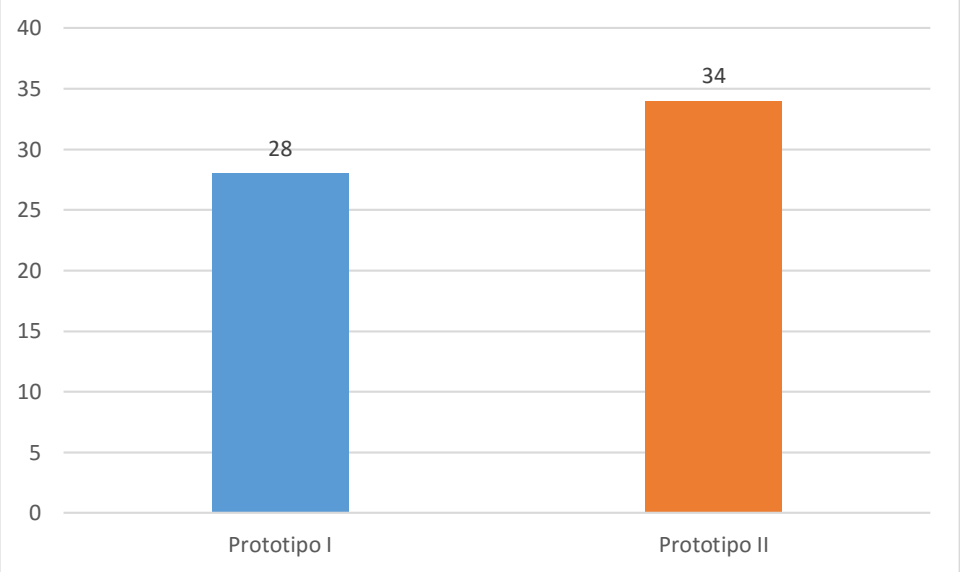



**Gráfico 23-4** Resultados totales de los resultados con cada prototipo
**Realizado por:** Villa, H. 2016

De esta manera se concluye que al utilizar la ***"Guía de mejores prácticas para la realización de una programación segura en Android"*** mejora en un 21,4% la seguridad en las aplicaciones Android escaneadas.

En el proceso de la ***estadística inferencial*** para la comprobación de la hipótesis de investigación se da los siguientes valores a la variable independiente X los siguientes valores:

X = Seguridad

$X_1$ = Mejora la seguridad

$X_2$ = No mejora la seguridad

Los mismos en los que se comprobará el impacto en relación a la variable dependiente que son los algoritmos de seguridad implementados en el Prototipo I y Prototipo II.

Para la prueba de hipótesis planteada se utilizó la prueba chi cuadrado o $X^2$, que es una prueba no paramétrica a través de la cual se mide la relación entre la variable dependiente e independiente.

Además, se considera la hipótesis nula Ho y la hipótesis de investigación Hi.

- **Hi:** *La propuesta de aplicación para dispositivos móviles Android contribuirá a la detección de vulnerabilidades para mejorar la seguridad en las aplicaciones escaneadas.*
- **Ho:** *La propuesta de aplicación para dispositivos móviles Android contribuirá a la detección de vulnerabilidades y no mejora la seguridad en las aplicaciones escaneadas.*

La tabla de contingencia creada para el cálculo de chi cuadrado, se muestra en la Tabla 28-4, en la que se ubican las frecuencias observadas de cada Indicador.

**Tabla 28-4** Tabla de contingencia para el cálculo de chi cuadrado

| V. Independiente / V. Dependiente | Aplicaciones | Prototipo I | Prototipo II | Total |
|---|---|---|---|---|
| **Mejora la seguridad (No vulnerable)** | NotePad | 0 | 4 | **4** |
| | Searchable Dictionary | 0 | 4 | **4** |
| | WifiDirectDemo | 0 | 4 | **4** |
| **No mejora la seguridad (Vulnerable)** | NotePad | 3 | 0 | **3** |
| | Searchable Dictionary | 2 | 0 | **2** |
| | WifiDirectDemo | 3 | 0 | **3** |
| **No se puede determinar** | NotePad | 4 | 4 | **8** |
| | Searchable Dictionary | 4 | 4 | **8** |
| | WifiDirectDemo | 4 | 4 | **8** |
| **TOTAL** | | **20** | **24** | **44** |

**Realizado por:** Villa, H. 2016

La tabla de contingencia de frecuencias esperadas son los valores que se esperaría encontrar si las variables no estuvieran relacionadas. Chi cuadrado parte del supuesto de "no relación entre las variables" y se evaluará si es cierto o no, analizando si sus frecuencias observadas son diferentes de lo que pudiera esperarse en caso de ausencia de correlación.

La frecuencia esperada de cada celda, se calcula mediante la siguiente fórmula aplicada a la tabla de frecuencias observadas.

$$fe = \frac{(total\_fil) * (total\_columna)}{N}$$

**Donde:**

**N:** Número total de frecuencias observadas.

Aplicando la fórmula a los valores de la Tabla 28-4 se obtiene la tabla de contingencia de valores esperados, como se muestra en la Tabla 29-4.

**Tabla 29-4** Tabla de contingencia de frecuencias esperadas

| V. Independiente / V. Dependiente | Aplicaciones | Prototipo I | Prototipo II | Total |
|---|---|---|---|---|
| **Mejora la seguridad (No vulnerable)** | NotePad | 1,82 | 2,18 | **4** |
| | Searchable Dictionary | 1,82 | 2,18 | **4** |
| | WifiDirectDemo | 1,82 | 2,18 | **4** |
| **No mejora la seguridad (Vulnerable)** | NotePad | 1,36 | 1,64 | **3** |
| | Searchable Dictionary | 0,91 | 1,09 | **2** |
| | WifiDirectDemo | 1,36 | 1,64 | **3** |
| **No se puede determinar** | NotePad | 3,64 | 4,36 | **8** |
| | Searchable Dictionary | 3,64 | 4,36 | **8** |
| | WifiDirectDemo | 3,64 | 4,36 | **8** |
| **TOTAL** | | **20** | **24** | **44** |

**Realizado por:** Villa, H. 2016

Una vez obtenida la tabla de frecuencias esperadas, se aplica la siguiente fórmula de chi cuadrado.

$$x^2 = \sum \frac{(o - E)^2}{E}$$

**Donde:**

**O:** Frecuencia observada en cada celda

**E:** Frecuencia esperada en cada celda

En la Tabla 30-4 se calcula el valor de $X^2$

**Tabla 30-4** Cálculo de $X^2$

| | Indicadores | $O$ | $E$ | $O-E$ | $(O-E)^2$ | $\frac{(O-E)^2}{E}$ |
|---|---|---|---|---|---|---|
| **Prototipo I** | Mejora/NotePad | 0 | 1,82 | -1,82 | 3,31 | 1,82 |
| | Mejora/Searchable Dictionary | 0 | 1,82 | -1,82 | 3,31 | 1,82 |
| | Mejora/WifiDirectDemo | 0 | 1,82 | -1,82 | 3,31 | 1,82 |
| **Prototipo II** | Mejora/NotePad | 4 | 2,18 | 1,82 | 3,31 | 1,52 |
| | Mejora/Searchable Dictionary | 4 | 2,18 | 1,82 | 3,31 | 1,52 |
| | Mejora/WifiDirectDemo | 4 | 2,18 | 1,82 | 3,31 | 1,52 |
| **Prototipo I** | No Mejora/NotePad | 3 | 1,36 | 1,64 | 2,68 | 1,96 |
| | No Mejora/Searchable Dictionary | 2 | 0,91 | 1,09 | 1,19 | 1,31 |
| | No Mejora/WifiDirectDemo | 3 | 1,36 | 1,64 | 2,68 | 1,96 |
| **Prototipo II** | No Mejora/NotePad | 0 | 1,64 | -1,64 | 2,68 | 1,64 |
| | No Mejora/Searchable Dictionary | 0 | 1,09 | -1,09 | 1,19 | 1,09 |
| | No Mejora/WifiDirectDemo | 0 | 1,64 | -1,64 | 2,68 | 1,64 |
| **Prototipo I** | Indeterminada/NotePad | 4 | 3,64 | 0,36 | 0,13 | 0,04 |
| | Indeterminada/Searchable Dictionary | 4 | 3,64 | 0,36 | 0,13 | 0,04 |
| | Indeterminada/WifiDirectDemo | 4 | 3,64 | 0,36 | 0,13 | 0,04 |
| **Prototipo II** | Indeterminada/NotePad | 4 | 4,36 | -0,36 | 0,13 | 0,03 |
| | Indeterminada/Searchable Dictionary | 4 | 4,36 | -0,36 | 0,13 | 0,03 |
| | Indeterminada/WifiDirectDemo | 4 | 4,36 | -0,36 | 0,13 | 0,03 |
| | **$X^2$** | | | | | **19.80** |

**Realizado por:** Villa, H. 2016

## Interpretación

Para determinar si el valor de $X^2$ es o no significativo, se debe determinar los grados de libertad mediante la siguiente fórmula.

$$GI = (f-1)(c-1)$$

**Donde:**

**f:** Número de filas de la tabla de continencia

**c:** Número de columnas de la tabla de contingencia

Por lo tanto:

$$GI = (9-1)(2-1) = 8$$

De acuerdo la tabla de distribución $X^2$ que se muestra en la Tabla 31-4 y eligiendo como nivel de significancia de $\alpha = 5\% = 0.05$ para obtener un nivel de confianza del 95%, se obtiene como punto crítico de $X^2$ para 8 grados de libertad $X^2_{Crìtico} = 15.5073$

**Tabla 31-4** Tabla de distribución de $X^2$

P = Probabilidad de encontrar un valor mayor o igual que el chi cuadrado tabulado, ν = Grados de Libertad

| ν/p | 0,001 | 0,0025 | 0,005 | 0,01 | 0,025 | 0,05 | 0,1 | 0,15 | 0,2 | 0,25 | 0,3 | 0,35 | 0,4 | 0,45 | 0,5 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10,8274 | 9,1404 | 7,8794 | 6,6349 | 5,0239 | 3,8415 | 2,7055 | 2,0722 | 1,6424 | 1,3233 | 1,0742 | 0,8735 | 0,7083 | 0,5707 | 0,4549 |
| 2 | 13,8150 | 11,9827 | 10,5965 | 9,2104 | 7,3778 | 5,9915 | 4,6052 | 3,7942 | 3,2189 | 2,7726 | 2,4079 | 2,0996 | 1,8326 | 1,5970 | 1,3863 |
| 3 | 16,2660 | 14,3202 | 12,8381 | 11,3449 | 9,3484 | 7,8147 | 6,2514 | 5,3170 | 4,6416 | 4,1083 | 3,6649 | 3,2831 | 2,9462 | 2,6430 | 2,3660 |
| 4 | 18,4662 | 16,4238 | 14,8602 | 13,2767 | 11,1433 | 9,4877 | 7,7794 | 6,7449 | 5,9886 | 5,3853 | 4,8784 | 4,4377 | 4,0446 | 3,6871 | 3,3567 |
| 5 | 20,5147 | 18,3854 | 16,7496 | 15,0863 | 12,8325 | 11,0705 | 9,2363 | 8,1152 | 7,2893 | 6,6257 | 6,0644 | 5,5731 | 5,1319 | 4,7278 | 4,3515 |
| 6 | 22,4575 | 20,2491 | 18,5475 | 16,8119 | 14,4494 | 12,5916 | 10,6446 | 9,4461 | 8,5581 | 7,8408 | 7,2311 | 6,6948 | 6,2108 | 5,7652 | 5,3481 |
| 7 | 24,3213 | 22,0402 | 20,2777 | 18,4753 | 16,0128 | 14,0671 | 12,0170 | 10,7479 | 9,8032 | 9,0371 | 8,3834 | 7,8061 | 7,2832 | 6,8000 | 6,3458 |
| 8 | 26,1239 | 23,7742 | 21,9549 | 20,0902 | 17,5345 | 15,5073 | 13,3616 | 12,0271 | 11,0301 | 10,2189 | 9,5245 | 8,9094 | 8,3505 | 7,8325 | 7,3441 |
| 9 | 27,8767 | 25,4625 | 23,5893 | 21,6660 | 19,0228 | 16,9190 | 14,6837 | 13,2880 | 12,2421 | 11,3887 | 10,6564 | 10,0060 | 9,4136 | 8,8632 | 8,3428 |
| 10 | 29,5879 | 27,1119 | 25,1881 | 23,2093 | 20,4832 | 18,3070 | 15,9872 | 14,5339 | 13,4420 | 12,5489 | 11,7807 | 11,0971 | 10,4732 | 9,8922 | 9,3418 |
| 11 | 31,2635 | 28,7291 | 26,7569 | 24,7250 | 21,9200 | 19,6752 | 17,2750 | 15,7671 | 14,6314 | 13,7007 | 12,8987 | 12,1836 | 11,5298 | 10,9199 | 10,3410 |
| 12 | 32,9092 | 30,3182 | 28,2997 | 26,2170 | 23,3367 | 21,0261 | 18,5493 | 16,9893 | 15,8120 | 14,8454 | 14,0111 | 13,2661 | 12,5838 | 11,9463 | 11,3403 |
| 13 | 34,5274 | 31,8830 | 29,8193 | 27,6882 | 24,7356 | 22,3620 | 19,8119 | 18,2020 | 16,9848 | 15,9839 | 15,1187 | 14,3451 | 13,6356 | 12,9717 | 12,3398 |
| 14 | 36,1239 | 33,4262 | 31,3194 | 29,1412 | 26,1189 | 23,6848 | 21,0641 | 19,4062 | 18,1508 | 17,1169 | 16,2221 | 15,4209 | 14,6853 | 13,9961 | 13,3393 |
| 15 | 37,6978 | 34,9494 | 32,8015 | 30,5780 | 27,4884 | 24,9958 | 22,3071 | 20,6030 | 19,3107 | 18,2451 | 17,3217 | 16,4940 | 15,7332 | 15,0197 | 14,3389 |
| 16 | 39,2518 | 36,4555 | 34,2671 | 31,9999 | 28,8453 | 26,2962 | 23,5418 | 21,7931 | 20,4651 | 19,3689 | 18,4179 | 17,5646 | 16,7795 | 16,0425 | 15,3385 |
| 17 | 40,7911 | 37,9462 | 35,7184 | 33,4087 | 30,1910 | 27,5871 | 24,7690 | 22,9770 | 21,6146 | 20,4887 | 19,5110 | 18,6330 | 17,8244 | 17,0646 | 16,3382 |
| 18 | 42,3119 | 39,4220 | 37,1564 | 34,8052 | 31,5264 | 28,8693 | 25,9894 | 24,1555 | 22,7595 | 21,6049 | 20,6014 | 19,6993 | 18,8679 | 18,0860 | 17,3379 |
| 19 | 43,8194 | 40,8847 | 38,5821 | 36,1908 | 32,8523 | 30,1435 | 27,2036 | 25,3289 | 23,9004 | 22,7178 | 21,6891 | 20,7638 | 19,9102 | 19,1069 | 18,3376 |
| 20 | 45,3142 | 42,3358 | 39,9969 | 37,5663 | 34,1696 | 31,4104 | 28,4120 | 26,4976 | 25,0375 | 23,8277 | 22,7745 | 21,8265 | 20,9514 | 20,1272 | 19,3374 |
| 21 | 46,7963 | 43,7749 | 41,4009 | 38,9322 | 35,4789 | 32,6706 | 29,6151 | 27,6620 | 26,1711 | 24,9348 | 23,8578 | 22,8876 | 21,9915 | 21,1470 | 20,3372 |
| 22 | 48,2676 | 45,2041 | 42,7957 | 40,2894 | 36,7807 | 33,9245 | 30,8133 | 28,8224 | 27,3015 | 26,0393 | 24,9390 | 23,9473 | 23,0307 | 22,1663 | 21,3370 |
| 23 | 49,7276 | 46,6231 | 44,1814 | 41,6383 | 38,0756 | 35,1725 | 32,0069 | 29,9792 | 28,4288 | 27,1413 | 26,0184 | 25,0055 | 24,0689 | 23,1852 | 22,3369 |
| 24 | 51,1790 | 48,0336 | 45,5584 | 42,9798 | 39,3641 | 36,4150 | 33,1962 | 31,1325 | 29,5533 | 28,2412 | 27,0960 | 26,0625 | 25,1064 | 24,2037 | 23,3367 |
| 25 | 52,6187 | 49,4351 | 46,9280 | 44,3140 | 40,6465 | 37,6525 | 34,3816 | 32,2825 | 30,6752 | 29,3388 | 28,1719 | 27,1183 | 26,1430 | 25,2218 | 24,3366 |
| 26 | 54,0511 | 50,8291 | 48,2898 | 45,6416 | 41,9231 | 38,8851 | 35,5632 | 33,4295 | 31,7946 | 30,4346 | 29,2463 | 28,1730 | 27,1789 | 26,2395 | 25,3365 |
| 27 | 55,4751 | 52,2152 | 49,6450 | 46,9628 | 43,1945 | 40,1133 | 36,7412 | 34,5736 | 32,9117 | 31,5284 | 30,3193 | 29,2266 | 28,2141 | 27,2569 | 26,3363 |
| 28 | 56,8918 | 53,5939 | 50,9936 | 48,2782 | 44,4608 | 41,3372 | 37,9159 | 35,7150 | 34,0266 | 32,6205 | 31,3909 | 30,2791 | 29,2486 | 28,2740 | 27,3362 |
| 29 | 58,3006 | 54,9662 | 52,3355 | 49,5878 | 45,7223 | 42,5569 | 39,0875 | 36,8538 | 35,1394 | 33,7109 | 32,4612 | 31,3308 | 30,2825 | 29,2908 | 28,3361 |

**Fuente:** http://labrad.fisica.edu.uy/docs/tabla_chi_cuadrado.pdf

El valor $X^2$ calculado $X^2_{Calculado}$ en esta investigación es de 19.80 que es superior al valor de la tabla de distribución de 15.5073, como se muestra en el Gráfico 24-4.

$$X^2_{Crítico}(15.5073) < X^2_{Calculado}(19.80)$$

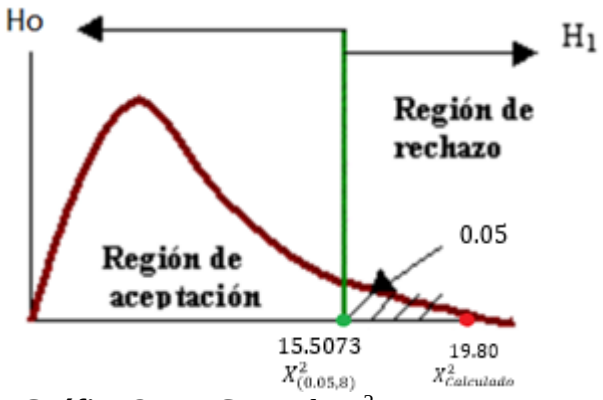



**Gráfico 24-4** Curva de $X^2$
**Realizado por:** Villa, H. 2016

Por lo que el valor calculado de $X^2$ se encuentra en el sector de rechazo de la hipótesis nula $H_0$, y se acepta la hipótesis de investigación que es significativa, con un nivel de significancia de $\alpha = 5\% = 0.05$ para obtener un nivel de confianza del 95%,

## CONCLUSIONES

- Luego del análisis realizado a los diferentes tipos de vulnerabilidades encontrados en Android, se selecciona a las de tipo intent, ya que permite una comunicación entre varios componentes de la aplicación Android y eso lo convierte en un componente sensible para recibir ataques.

- La aplicación desarrollada escanea las vulnerabilidades basadas en intents mediante el uso de patrones que facilitan la especificación de las mismas, para tal fin se utiliza dos componentes importantes que son el generador y el framework de ejecución de los casos de prueba JUnit.

- Se desarrolló una Guía de mejores prácticas para la realización de una programación segura en Android que permite mejorar la seguridad en las aplicaciones

- Luego del análisis realizado entre los prototipos I y II de las 3 aplicaciones, se puede verificar que el prototipo II tiene menos vulnerabilidades debido a que se incorpora las recomendaciones que se brindan en la guía, lo que implica un mejoramiento en la seguridad de la aplicación escaneada.

- El prototipo II es más seguro en un 21,4% debido a que en este prototipo se utiliza las recomendaciones de la Guía de mejores prácticas para la realización de una programación segura en Android

## RECOMENDACIONES

- Al momento de desarrollar software saber qué tipo de intents se va a usar (implícito o explícito), ya que el uso incorrecto provocaría que la aplicación exponga datos sensibles o cree vulnerabilidades que los atacantes pueden aprovechar.

- Siempre tomar en cuenta la documentación oficial que Android mismo nos provee, ya que de esa manera, se puede evitar el uso innecesario de componentes que no se ocupen o que provoquen que la aplicación colapse.

- Dar continuidad al desarrollo de la aplicación APSEBI mediante la creación de más patrones de vulnerabilidad que se pueden establecer en base a nuevos huecos de seguridad que se puedan encontrar.

- Definir los escenarios, ambientes de pruebas e indicadores de las variables dependiente independiente, de forma adecuada con el objetivo de que la ejecución y resultados de las pruebas realizadas sean confiables.

- Realizar las comparaciones en situaciones similares con el objetivo de verificar que las modificaciones realizadas permiten obtener valores significativos de mejora.

- Utilizar la Guía de mejores prácticas elaborada, ya que de esa manera se solventaría en gran parte las vulnerabilidades basadas en intents.

## RECOMENDACIONES PARA TRABAJOS FUTUROS

- Crear más patrones de vulnerabilidad que englobe más fallas de seguridad, basándose en organizaciones que se dedican a ver fallas de seguridad y proponer soluciones como OWASP (Open Web Application Security Project)
- Se implemente APSEBI en todas las plataformas más populares para dispositivos móviles, es decir, Android, iOS, Windows Phone, BlackBerry.

- Actualizar la guía de mejores prácticas en función de las posteriores versiones de Android que sean lanzadas, sin olvidar la documentación oficial que Android mismo provee.

**BIBLIOGRAFÍA**


**BERND KLEINJOHANN, L. K.** (2007). *From Model-Driven Design to Resource Management for Distributed Embedded Systems.* Braga, Portugal: Springer.

**CHIN, E., FELT, A., GREENWOOD, K., & WAGNER, D.** (2011). *Analyzing inter-application communication in android.*, (págs. roceedings of the 9th International Conference on Mobile Systems, Applications, and Services).

**DHAYA, R., DEPT. OF CSE, V. E., & POONGODI, M**. (2014). *Detecting software vulnerabilities in android using static analysis*. Advanced Communication Control and Computing Technologies (ICACCCT), 2014 International Conference on, (págs. 915 - 918). Ramanathapuram.

**HAMANDI, K., BEIRUT, D. O., CHEHAB, A., ELHAJJ, I., & KAYSSI, A.** (2013). *Android SMS Malware: Vulnerability and Mitigation. Advanced Information Networking and Applications Workshops (WAINA)*, 2013 27th International Conference on (págs. 1004 - 1009). Barcelona: IEEE.

**HASSAN, R., & NARAYANA, M.** (2012). *A Secure Software Architecture for Mobile Computing. Information Technology: New Generations (ITNG)*, 2012 Ninth International (págs. 566-571). Las Vegas: IEEE.

**KHODOR, H., ALI, C., IMAD, E., & AYMAN, K.** (2013). *Android SMS Malware: Vulnerability and Mitigation. Advanced Information Networking and Applications Workshops (WAINA)*, 2013 27th International (págs. 1004-1009). Barcelona: IEEE.

**MICROSOFT.** (1 de 2 de 2015). *Microsoft*. Obtenido de http://download.microsoft.com/download/1/A/E/1AE5C1D8-8874-481B-94F8-57B41D4E8965/Microsoft_Security_Intelligence_Report_Volume_17_English.pdf

**PASCAL, U., & CHRISTOPHE, K**. (2012). *A new cooperative architecture for sharing services managed by secure elements controlled by android phones with IP objects.* Collaboration Technologies and Systems (CTS) (págs. 404-409). Denver: IEEE.

**QINGQING, S., TAO, Q., TAN, Y., & YIDONG, C.** (2013). *An Android dynamic data protection model based on light virtualization*. Communication Technology (ICCT), 2013 15th IEEE International Conference (págs. 65-69). Guilin: IEEE.

**SALVA, S., & ZAFIMIHARISOA, S. R.** (2015). *APSET, an Android aPplication SEcurity Testing tool for detecting intent-based vulnerabilities*. International Journal on Software Tools for Technology Transfer (págs. 1433-2779). Springer Berlin Heidelberg.

**SOURABH, G., NIKHIL, C., DEEPAK, S., & AKASH, G.** (2013). *A framework for executing android applications on the cloud*. Advances in Computing, Communications and Informatics (ICACCI), 2013 International (págs. 230-235). Mysore: IEEE.

**STATISTA**. (01 de 02 de 2015). *Statista*. Obtenido de http://www.statista.com/statistics/266970/market-share-forecast-of-smartphone-operating-systems-from-2010-to-2015/

**WU, J., WU, Y., YANG, M., WU, Z., & WANG, Y.** (2013). *Vulnerability Detection of Android System in Fuzzing Cloud.*

**XATAKANDROID**. (01 de 02 de 2015). *Xatakandroid*. Obtenido de http://www.xatakandroid.com/sistema-operativo/que-es-android

# ANEXOS

## Anexo A: Código de vulnerabilidades

Código de vulnerabilidad de integridad relacionada con los servicios

```
digraph Integrity {

l_0->l_1[label="!intent(action,category,type,uri,extra), [in(type,T+RV(String)) x in(uri,U+RV(String)+INJ) x in(extra,Ve)], *"];

l_1->l_2[label="?running(ServiceS), [in(S.name,ApplicationComponent) x S.isEnable=true],*"];

ll_1->pass[label="?Exception, [*], *"];

l_2->l_3[label=" !q,[data>0], *"];

l_3->pass[label="?open(ContentProviderCP), [in(CP.name,ApplicationComponent) x CP.isEnable=true x data=uri.data], *"];

l_3->fail[label="?open(ContentProviderCP), [in(CP.name,ApplicationComponent) x CP.isEnable=true x data#uri.data], *"];

}
```

Código de vulnerabilidad de integridad relacionado con las actividades

```
digraph Integrity {

l_0->l_1[label="!intent(action,category,type,uri,extra), [in(type,T+RV(String)) x in(uri,U+RV(String)+INJ) x in(extra,Ve)], *"];

l_1->l_2[label="?display(ActivityA), [in(A.name,ApplicationComponent) x A.isEnable=true], *"];

l_1->pass[label="?Exception, [*], *"];

l_2->l_3[label=" !q,[data>0], *"];

l_3->pass[label="?open(ContentProviderCP), [in(CP.name,ApplicationComponent) x CP.isEnable=true x data=uri.data], *"];

l_3->fail[label="?open(ContentProviderCP), [in(CP.name,ApplicationComponent) x CP.isEnable=true x data#uri.data], *"];

}
```