Universidad de Deusto

RUBICÓN: UN NUEVO ENFOQUE PARA LA SEGURIDAD EN LAS APLICACIONES DE SMARTPHONES

Tesis doctoral presentada por

BORJA SANZ URQUIJO

Bilbao, diciembre 2012

RUBICÓN: UN NUEVO ENFOQUE PARA LA SEGURIDAD EN LAS APLICACIONES DE SMARTPHONES

Tesis doctoral presentada por
BORJA SANZ URQUIJO

Dentro del programa de
Doctorado en Sistemas de Información

Facultad de Ingeniería - ESIDE
Universidad de Deusto

Dirigida por el Dr. Pablo García Bringas
y por el Dr. Gonzalo Álvarez Marañón

El doctorando | El director | El director

Bilbao, diciembre 2012

Borja Sanz Urquijo: *RUBICÓN: Un nuevo enfoque para la seguridad en las aplicaciones de smartphones*, © diciembre 2012 .

WEBSITE:
http://paginaspersonales.deusto.es/bosanz

E-MAIL:
borja.sanz@deusto.es

A Aitziber.
Por todo lo recorrido, por todo lo que queda por recorrer.

A mis padres, mi hermano, mis abuelos.
Soy lo que soy gracias a vosotros.

«Se nos conoce por nuestros actos.»

B.Wayne, creado por B. Kane.

RESUMEN

El crecimiento del número de teléfonos móviles inteligentes o smarphones ha sido y está siendo extraordinario. Estos equipos, dotados de gran movilidad y hardware dedicado (p.ej., GPS o giroscopio) son gobernados por sistemas operativos cada vez más complejos. Además, la proliferación de las «tiendas de aplicaciones» (es decir, tiendas donde se distribuyen aplicaciones para cada sistema operativo) ha creado una forma sencilla para el usuario de instalar aplicaciones en el terminal.

Desafortunadamente, la gestión de la seguridad de estos dispositivos dista mucho de ser óptima. La proliferación del aplicaciones maliciosas (malware) en este tipo de plataformas, el acceso por parte de las aplicaciones a datos sensibles y su gestión a espaldas de los usuarios, han creado un escenario en el que los dispositivos almacenan una gran cantidad de información de carácter personal (p.ej., contraseñas, agenda de contactos, los mensajes o los correos electrónicos); cuya seguridad no está tan madura como en entornos más asentados, como pueden ser los equipos de escritorio.

La comunidad científica se ha lanzado a buscar soluciones para mitigar esta problemática. Para ello, se han intentado migrar modelos que funcionaban en entornos de escritorio a este tipo de dispositivos con suerte dispar. Se han desarrollado representaciones de las aplicaciones de smartphones, para posteriormente utilizar técnicas de aprendizaje automático para clasificar las aplicaciones, aunque el resultado obtenido no es tan óptimo como en otros entornos.

Con este telón de fondo, el objetivo general es determinar qué amenazas son las más peligrosas a las que está expuesto el terminal y mitigar su impacto sin que sea necesaria la intervención del usuario. Por ello, se formula la siguiente hipótesis fundamental: *«Es posible, mediante el uso de algoritmos supervisados de inteligencia artificial y minería de datos, hacer una gestión inteligente, automática, y efectiva de la seguridad en las aplicaciones de los teléfonos smartphones, tal que libere al usuario de parte de la responsabilidad de la gestión de la seguridad del mismo.»*.

Para validar esta hipótesis, se realiza en primer lugar una evaluación exhaustiva de las soluciones existentes. A continuación,

se desarrolla un nuevo modelado de las amenazas existentes en este tipo de dispositivos. Con el fin de validar este nuevo modelado, se realiza un banco de ataques que define los mayores activos, amenazas, ataques y vulnerabilidades que se dan en estos dispositivos.

Tras evaluar el resultado obtenido, se determina que la mayor amenaza se centra en el software malicioso o *malware*. Posteriormente se fijan los criterios sobre los que se evaluará la solución propuesta para la detección de este tipo de software. A continuación se diseña y desarrolla una solución que mejore esta situación, específicamente sobre la plataforma Android, para, finalmente, evaluarla mediante el uso de métricas aplicadas en el área de la inteligencia artificial y contrastarla en base a los criterios anteriormente seleccionados.

Los smartphones se han convertido en una herramienta importante en el día a día, que almacena cantidad de información sensible.

Mediante esta investigación se busca ampliar el estado del arte en la detección del malware en smartphones, avanzando en la creación de un entorno más seguro para el uso de este tipo de sistemas.

ABSTRACT

The number of smartphones has grown exponentially in recent years. These devices, which have a big mobility and dedicated hardware (i.e., GPS or gyroscope), are guided by complex operating systems. In addition, the proliferation of application stores has generated a new easy way to install tools and games directly on device.

Unfortunately, the security management of these devices is far from optimal. The proliferation of malicious applications (malware) in these platforms, added to the fact that the access of these applications to sensible data is made behind the backs of users, has created a new scene. In this new scenario, these devices store a huge amount of private and sensitive data (e.g., short messages or e-mails), and their security is not as mature as in other enviroments, for example in personal computers.

The scientific community has accepted the challenge and is looking for solutions. To this end, they have tried to migrate models from desktop environment to these devices with mixed fortunes. Some researchers have developed representations of applications and, after that, they apply machine learning techniques with different results.

Against this background, the main goal of this research is to mitigate the threats to which these devices are exposed to, through the surveillance of installed applications, without any user interaction. Therefore, we formulate the following hypothesis: "It is possible, using supervised algorithms of artificial intelligence and data mining, to deploy an intelligent, automatic and effective security layer for smartphones to release the user from the responsibility for managing the safety."

In order to validate this hypothesis, first we made an exhaustive evaluation of the existing solutions. Then, we developed a new threat modeling in these devices. To validate this model, we have developed a new bank of attacks, which defines the assets, threats, attacks and vulnerabilities that occurs in these devices.

After evaluating the results, we concluded that malicious software is the biggest threat facing these devices. Then, we designed and developed a solution which enhanced this situation. We use Android platform to validate this solution and evaluate

this using metrics that are applied in machine learning area and contrast them based on previously selected criteria. Through this investigation we want to advance in the state of the art of malware detection in smartphones, progressing in creation a safe environment for the use of such systems.

AGRADECIMIENTOS

Antes de comenzar a exponer los resultados de esta investigación doctoral, creo que es el momento de agradecer a todas aquellas personas que me han ayudado a lograr este importante hito en mi vida.

Esta tesis doctoral no habría sido posible sin sus directores. Gracias, *Pablo*, por darme la oportunidad de entrar en el S^{3}Lab, por la iniciación en la ciencia y por todos aquellos consejos y opiniones que me han hecho crecer durante estos años. Gracias, *Gonzalo*, sobre todo por enseñarme que la comunicación de lo investigado es casi tan importante como la propia investigación. Gracias a los dos por la guía durante este proceso y por ayudarme a crecer como investigador.

La comunidad universitaria de Deusto ha formado una parte importante de mi vida en estos últimos años. Por ello, quisiera dar las gracias a todo el personal tanto a la Universidad de Deusto como a DeustoTech por todo lo que me han enseñado a lo largo de estos años. En particular, quisiera agradecerle a Josu Solabarrieta por ofrecerme mi primer trabajo en Deusto en los últimos años de la carrera, enseñándome que los informáticos tenemos una forma de pensar algo particular. Gracias *Josu*. También me gustaría agradecer al director del programa de doctorado su ayuda en este proceso final de la tesis. Gracias *Diego*.

No habría sido posible hacer algunos experimentos de esta tesis sin la colaboración de VirusTotal. Un simple mail fue suficiente para que me dieran acceso a su vasta colección de muestras de malware para Android. En particular, quiero agradecer especialmente al receptor de ese mail, *Sergio de los Santos*, por las facilidades dadas para conseguir las muestras.

No se me ocurre ningún lugar mejor donde haber podido desarrollar este trabajo doctoral que en el S^{3}Lab. Eso es así gracias a las personas que lo componen. Quisiera agradecer en especial a *Mikel Salazar Gonzalez* el trabajo realizado en algunas de las ilustraciones de esta tesis. Gracias a *Iker Pastor López* por las largas conversaciones mantenidas en estos últimos años. Ánimo, que dentro de poco estarás escribiendo líneas como éstas en tu propia tesis. A *Patxi Galán García*, a *Xabier Ugarte Pedrero*, a *Félix Brezo Fernández* y a *José Gaviria de la Puerta*, por ser la savia nue-

va que le ha dado nuevos aires al laboratorio, muchas gracias también. Seguid siendo como sois.

He tenido la suerte de tener buenos compañeros de trabajo que me han acompañado en este viaje, pero hay tres en particular que han tenido especial importancia estos años.

El primero de ellos ha sido Igor Santos. Su empuje y decisión nos ha llevado a todo el equipo a obtener grandes resultados y, lo que es más importante, aprender y disfrutar con la ciencia. Gracias *Igor*, por influirme tanto con tu forma de entender la ciencia.

Por otro lado, seguramente haya sido con Carlos Laorden con el que más tiempo he pasado trabajando codo con codo, creciendo y aprendiendo como investigadores. Gracias *Carlos*, por hacerme ver que la satisfacción obtenida del trabajo en equipo es directamente proporcional al valor del equipo en sí.

Finalmente, Javier Nieves ha sido, además de un extraordinario compañero de fatigas, el compañero de travesía en esta etapa final del trabajo doctoral, hasta el punto de hacer juntos el proceso, a veces duro, y otras incomprensibles, que esperemos, nos llevará a ser doctores. Gracias *Javi*, por todos acompañarme en este último sprint.

Un honor y un orgullo trabajar con vosotros.

Por último, no quisiera dejar pasar la oportunidad de agradecer su aportación a personas realmente importantes para mi que, pese a no estar relacionados con la ciencia, son igualmente responsables de la consecución de este trabajo doctoral.

No me gusta nada el término «familia política». Para mí, son simplemente familia. Gracias a *Raquel* y *José* por acogerme desde el minuto cero como si hubiera estado ahí siempre. No lo olvidaré jamás. Gracias *Filo* por esas conversaciones en la cocina de tu casa. Gracias *Leire* por la risa constante, gracias *Gorka* todos estos años de debates, tanto de informática como de la vida. La locura siempre es más divertida en compañía. Gracias *Unate*, por acompañarme todos estos años tanto en la locura como en la cordura, aunque ésta última haya sido efímera. Y lo que nos queda por reírnos.

Tengo claro que soy lo que soy gracias a mi familia. Mis padres, mi hermano y mis abuelos, estén con nosotros o no, me han enseñado todos los valores y principios que me rigen. Esta tesis es sólo uno de los frutos de todas esas cosas buenas de la vida que me habéis enseñado. Gracias por el apoyo, los valores y el ejemplo que habéis supuesto durante toda mi vida.

Esta investigación no habría sido posible sin el sacrificio, el apoyo y los ánimos de *Aitziber*. En los últimos 13 años hemos aprendido juntos un montón de cosas, y las que nos quedan por aprender. Gracias por ser como eres, por la felicidad constante, por estar a mi lado siempre y por darme la fuerza en esos momentos en los que no ando sobrado de ella.

Y por último, gracias de corazón a todos los que habéis hecho posible esta tesis y no estáis en estos agradecimientos. No me olvido de vosotros; simplemente, es hora de empezar a trabajar.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 MOTIVACIÓN

«La perfección no se alcanza cuando no hay nada más que añadir, sino cuando no hay nada más que quitar»

A. de Saint-Exupery
(1900 – 1944)

ÍNDICE

A lo largo de los últimos años, el crecimiento experimentado por los llamados teléfonos inteligentes o *smartphones* ha sido y está siendo extraordinario. Sin embargo, esto ha traído consigo una serie de problemas, especialmente, desde el punto de vista de la seguridad.

Este primer capítulo sirve para introducir la problemática de la seguridad dentro de este tipo de dispositivos. Primeramente, veremos cómo definirlos para, a continuación, centrarnos en Android, que es el sistema operativo elegido para realizar la experimentación de la tesis. Las razones por las que hemos elegido este sistema también están esgrimidas a lo largo de este capítulo.

Al final de este capítulo tendremos una visión clara de la problemática que pretende afrontar esta tesis. También tendremos una visión global de la plataforma elegida para la realización de la misma.

1.1 LA PROLIFERACIÓN DE LOS TELÉFONOS INTELIGENTES

Existen más de 500 millones de terminales gobernados por Android.

En los últimos años, los teléfonos inteligentes o *smartphones* han supuesto una revolución en el panorama tecnológico. Las cifras hablan por si solas. Según un informe de IDC[1], el crecimiento experimentado por este tipo de dispositivos en el último año ha sido de un 54 % más de dispositivos vendidos, siendo especialmente relevante el crecimiento del número de unidades vendidas por Samsung, que ha apostado fuerte por el sistema operativo Android, y que ha vendido un 275 % más de dispositivos, llegando a vender 36 millones de unidades en el último cuatrimestre del año respecto al mismo periodo de tiempo que en el año anterior.

Android ha sido uno de los sistemas operativos que más ha crecido. En sus escasos 5 años de vida, ha conseguido alcanzar un ritmo 1.3 millones de activaciones diarias, rozando los 500 millones de dispositivos gobernados por este sistema operativo[2]

Este tipo de dispositivos también ha hecho proliferar el número de aplicaciones que se desarrollan para estos dispositivos. De nuevo, las cifras son elocuentes. La tienda de aplicaciones de Apple, el AppStore (a la que podríamos considerar la primera tienda de estas características), alberga 500.000 aplicaciones, mientras que la tienda oficial de Android, Google Play, aloja a más de 600.000 aplicaciones[3]. Y la importancia económica de este tipo de tiendas no deja de crecer. Según un informe de Canalys[4], el beneficio directo obtenido de las aplicaciones, bien sea por compra directa, compras dentro de la aplicación o suscripciones, pasará de los 7.300 millones de dólares de 2011 a los 36.000 millones en 2015, lo que supone un incremento cercano al 500 %.

Se estima que en el año 2015 los beneficios de las aplicaciones en terminales móviles serán de 36.000 millones de dólares.

Todos estos datos reflejan el impacto que ha tenido este tipo de tecnología. Sin embargo, dentro de esta vorágine de crecimiento, no se ha tenido en cuenta un aspecto tan importante como la seguridad. Como consecuencia, han surgido en los últimos años nuevas amenazas y nuevos vectores de ataque. En este capítulo, analizaremos más a fondo esta situación.

1 http://www.idc.com/getdoc.jsp?containerId=prUS23299912
2 http://www.engadget.com/2012/09/05/google-ceo-1-3-million-android-activations-a-day/
3 http://news.cnet.com/8301-33620_3-57471636-278/does-an-app-stores-size-matter-if-content-is-the-killer-app/
4 http://www.canalys.com/newsroom/app-stores-direct-revenue-exceed-14-billion-next-year-and-reach-close-37-billion-2015

1.2 DEFINICIÓN FORMAL DE TELÉFONO INTELIGENTE

Un teléfono inteligente es un dispositivo móvil con funcionalidad avanzada, hardware especializado y una alta capacidad de cómputo.

Smartphone es un término inglés que se utiliza para definir a los teléfonos que cumplen una serie de características:

- *Funcionalidad avanzada*: los *smartphones* son aquellos teléfonos cuya funcionalidad va más allá de llamar o recibir mensajes SMS. Este tipo de teléfonos son capaces de realizar tareas más complejas como la gestión del correo personal o la reproducción de contenidos multimedia, entre otros.
- *Hardware especializado*: estos dispositivos cuentan con hardware dedicado necesario para la realización de las tareas avanzadas de las que son capaces. De esta manera, este tipo de dispositivos suele contar con chip GPS (*Global Positioning System*), giroscopio, acelerómetro o procesador gráfico.
- *Alta capacidad de cómputo*: la capacidad de procesamiento de información que poseen estos dispositivos es la que hace posible toda esa funcionalidad avanzada.

Por lo tanto, podemos definir un smartphone de la siguiente forma:

> *Un smartphone es un teléfono móvil que, gracias a una alta capacidad de cómputo y al hardware especializado que dispone, es capaz de realizar funciones avanzadas en movilidad.*

Algunos autores lo definen como la combinación de dispositivos ya existentes, mientras que otros autores describen sus características.

Otros autores definen los *smartphones* desde otros puntos de vista. Cheng et al. [CWYL07] definen los *smartphones* de la siguiente manera:

> «*Un nuevo tipo de dispositivo de comunicación que combina la funcionalidad de los teléfonos móviles tradicionales (por ejemplo, voz y mensajería) con aquellos que son de computación en movilidad como las PDAs (Personal Digital Assistant)*».

Por su parte, Chow *et ál.* [CJ08] definen estos dispositivos de una manera más concreta:

> «*Los teléfonos móviles hoy en día han evolucionado hasta convertirse en una plataforma de comunicación y de*

> *cómputo ubicua donde los usuarios pueden hacer las llamadas como un teléfono tradicional, crear diapositivas de una presentación, editar documentos, escuchar música, ver vídeos, jugar juegos, y acceder a Internet, todo desde un único dispositivo móvil, simplemente conocido como teléfono inteligente.*»

Con estas definiciones podemos ver el término *smartphone* desde distintos puntos de vista, tanto desde la evolución natural de distintos dispositivos existentes como desde la descripción de sus capacidades.

1.3 LA PLATAFORMA ANDROID

Google es una empresa que fue fundada el 4 de septiembre de 1998 en Menlo Park, con el fin de comercializar el motor de búsqueda creado como fruto de la tesis doctoral realizada por dos estudiantes de la Universidad de Standford, Larry Page y Sergei Brin. Este buscador fue el punto de partida para la comercialización de una gran diversidad de productos, algunos de desarrollos propios y otros a través de la adquisición de empresas.

Google compró Android Inc. en el año 2005.

Dentro de este último grupo se encuentra Android. En junio de 2005, Google compró una pequeña compañía cuya finalidad era el desarrollo de aplicaciones para dispositivos móviles que se llamaba Android Inc. Con el paso del tiempo, uno de los cofundadores de aquella pequeña compañía, Andy Rubin, pasó a ser el director de la división de plataformas móviles de Google, que es una de las máximas responsables del desarrollo del sistema operativo Android.

El desarrollo de Android se hace desde la Open Handset Alliance, conformada por varias empresas, entre ellas Google.

Android es un sistema operativo orientado a dispositivos móviles basado en una versión modificada del sistema operativo GNU y el núcleo Linux. El desarrollo corre a cargo de la Open Handset Alliance[5], que es un conglomerado de fabricantes de software y hardware, compuesto entre otros por Google, Intel, Qualcomm, Texas Instruments, Nvidia, Dell, HP o Motorola.

Los componentes principales de este sistema operativo son:

- *Aplicaciones*, escritas principalmente en el lenguaje de programación Java.
- *Framework de aplicaciones*, diseñado con el objetivo de simplificar la reutilización de componentes.

5 http://www.openhandsetalliance.com/

- *Bibliotecas*, escritas en C/C++, usadas por el sistema y expuestas a los desarrolladores a través del *framework*.

Está desarrollado en C/C++, pero las aplicaciones se ejecutan en una máquina virtual de Java.

- *Runtime*, que incluye la máquina virtual Dalvik donde se ejecutan las aplicaciones. Cada vez que una aplicación se ejecuta se crea una nueva instancia de la maquina virtual. Estas máquinas virtuales están optimizadas para consumir la mínima memoria, así como para la ejecución en paralelo de las distintas tareas.

- *Núcleo de Linux*, para los servicios base del sistema, como son la seguridad, la gestión de procesos, etc.

Las características más reseñables desde el punto de vista del desarrollador de este sistema operativo son:

La plataforma tiene un serio problema de fragmentación de versiones.

- *Importantes herramientas de desarrollo*, incluyendo un emulador de dispositivo, herramientas para la depuración de proyectos, perfiles de memoria y rendimiento y un complemento para el IDE Eclipse[6].

- *Android Market*, que permite que, siguiendo el modelo desarrollado por Apple, los desarrolladores pongan sus aplicaciones, bien sean gratuitas o de pago, en el mercado a través de esta plataforma.

El desarrollo de Android ha sido vertiginoso. En los últimos años, la variedad de socios, unido a la posibilidad de personalización que ofrece el sistema operativo, ha conseguido que crezca a un ritmo vertiginoso. Sin embargo, se empiezan a plantear una serie de problemas, provocados en parte por esta libertad.

Por un lado, existe una gran fragmentación de versiones del sistema operativo, hasta el punto que, como se puede ver en la Figura 1.1 (obtenida de los propios datos que proporciona Android[7]), ahora mismo conviven 5 versiones distintas del sistema operativo, cada cual con su cuota de mercado. Esto supone un problema a la hora de desarrollar las aplicaciones, ya que el uso de ciertas características de la API limita la versión del sistema operativo que se puede utilizar.

Uno de los grandes problemas de Android es la fragmentación tanto a nivel de hardware como de software.

Además de la fragmentación de software, también se produce otra importante en el hardware, ya que el sistema operativo puede estar soportado por una gran variedad de *smartphones* con

6 http://www.eclipse.org/

7 http://developer.android.com/resources/dashboard/platform-versions.html

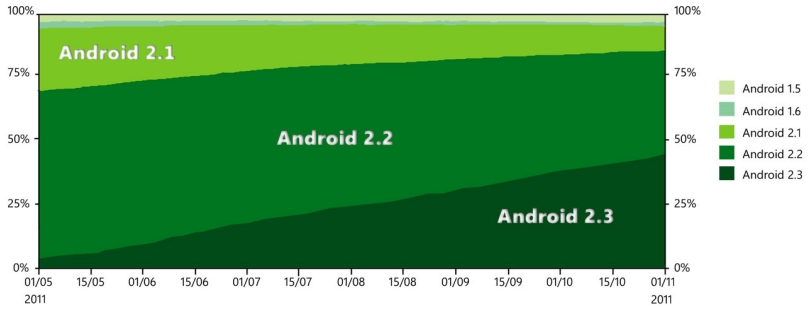


Figura 1.1: Distribución histórica de las versiones de los sistemas operativos de Android, obtenida de la página de desarrolladores de Android, a 1 de Noviembre de 2011.

diversas características. Esto dificulta enormemente la labor de los desarrolladores, ya que no tienen una idea clara de cuáles son los recursos a los que pueden acceder para desarrollar sus aplicaciones (p.ej., si dispone de GPS o la cantidad de memoria disponible).

1.3.1 ¿Por qué Android?

Existen varias razones por las que hemos elegido esta plataforma. La primera de ellas, y la principal para el curso de la investigación, es que se trata de un sistema operativo de fuentes abiertas. Esto implica que es posible obtener el código fuente del mismo para poder examinarlo. Múltiples partes del sistema son también de código abierto, por lo que es posible acceder a él. Por ejemplo, el núcleo del sistema es una versión adaptada a procesadores de ARM del núcleo Linux. Este núcleo es un software muy documentado, cuyos problemas de seguridad también son bien conocidos.

Se optó por Android por tratarse de una plataforma abierta, lo que ha provocado el interés de la comunidad científica.

Por otro lado, la representatividad que Android está adquiriendo en el mercado es otro de los puntos a tener en cuenta. Se trata de un sistema operativo que está desarrollando un gran crecimiento en los últimos tiempos, especialmente en Europa. Según los datos ofrecidos por IDC [Idc11] el número de teléfonos vendidos con este sistema operativo ha crecido un 1.580 % en un año (de 470.000 unidades en 2009 a 7,9 millones en 2010). Este crecimiento lo hace atractivo para la nueva industria del *malware* (locución inglesa que proviene de la combinación de las palabras «malicious» y «software»). Prueba de ello son las muestras

de *malware* para esta plataforma aparecidas recientemente, que se analizan en el punto 1.4.1.

Por último, el interés de la comunidad investigadora se ha centrado principalmente en los últimos tiempos en este sistema operativo. Si bien buena parte de la investigación sobre móviles existente se ha realizado sobre la plataforma Symbian [Mor07], debido a su caída en el mercado, a lo largo de los últimos años han aparecido varios artículos centrados en aspectos de seguridad de la plataforma Android [OME09b, SSB+09, EGC+10a]. Ambos sistemas operativos poseen varios puntos en común (por ejemplo, los dos de código abierto y con una gran presencia en el mercado) que hacen aumentar el interés de la comunidad científica en desarrollar e investigar sobre ellas. Android parece que es el heredero natural de Symbian, y de la misma forma que en el mercado, dentro de la comunidad científica también ha ocupado el puesto vacante dejado por éste.

Muchas investigaciones se han realizado sobre esta plataforma en los últimos años.

Por todo ello, consideramos que Android es la plataforma adecuada para sustentar los experimentos de este trabajo doctoral.

1.4 EL PROBLEMA DE LA SEGURIDAD EN LOS TELÉFONOS INTELIGENTES ANDROID

Con el crecimiento de estos últimos años en el uso de estos dispositivos, también ha crecido el número de peligros que deben encarar. Así, existen distintas amenazas sobre este tipo de dispositivos. Una de las que más ha evolucionado en los últimos tiempos es el desarrollo de aplicaciones maliciosas o *malware* sobre este tipo de plataformas.

Para el desarrollo del modelo de seguridad de la plataforma se basaron en la experiencia adquirida durante los últimos años por parte de Google en materia de seguridad. Este modelo está detallado en la sección 4.1, en el que se analizarán las hipótesis de partida y el desarrollo del mismo.

Uno de los mayores problemas de seguridad a los que se enfrenta Android es el malware.

A continuación, analizaremos el estado de esta problemática y, más concretamente, la que se da en la plataforma Android.

1.4.1 Situación del malware sobre Android

En este apartado vamos a ver el estado en el que se encuentra el *malware* en *smartphones*. Con ese fin, hemos hecho un estudio a partir de los datos obtenidos de la empresa especialista en seguri-

dad sobre dispositivos móviles NetQuin[8] y, más concretamente, de su base de datos[9].

A continuación se muestran los datos obtenidos, separados en varias categorías. Primero se analizará el objetivo que se esconde detrás del *malware*. Posteriormente, se clasificarán las muestras analizadas en función de su peligrosidad. Finalmente, se analizará el patrón de crecimiento que está experimentando el *malware* sobre esta plataforma.

Hemos clasificado los objetivos del malware en 5 categorías distintas.

En primer lugar, en la Figura 1.2 podemos observar cuál es el objetivo del *malware* aparecido en esta plataforma. Estos objetivos se clasifican en cinco categorías distintas:

- *Robo de Privacidad*: sustrae datos de carácter privado de las víctimas o roba información de los *smartphones*.
- *Industria del malware*: los creadores de malware pueden obtener beneficio económico a través de la vente en el mercado negro de la información y los recursos sustraídos de las víctimas.
- *Cuota de Consumo*: consumo sin permiso expreso de servicios de tarificación especial.
- *Puerta Trasera*: se ejecutan en segundo plano y abren algún puerto para acceder o controlar los *smartphones*. De esta manera los atacantes obtienen beneficio sustrayendo información del terminal y vendiéndola en el mercado negro o alquilando la capacidad de cómputo del terminal.
- *Badware*: aplicaciones difíciles de desinstalar y que dificultan el uso del teléfono por parte de los usuarios.

La mayor parte del malware existente para esta plataforma busca el robo de información personal.

En la Figura 1.2 podemos observar cuál es el objetivo de la mayoría de las muestras de *malware* analizadas: el robo de información personal. Cabe resaltar que estos dispositivos almacenan una gran cantidad de información personal (p.ej. información de nuestros contactos, nuestras fotos, números de teléfono de los conocidos o de nuestras próximas citas). También almacenan datos de nuestra identidad digital, como son usuarios y contraseñas de cuentas de correo electrónico o de redes sociales. Toda esta información adquiere sentido gracias a la existencia de un mercado negro, en el que se negocia con ella. Según el informe de Panda

8 http://www.netqin.com/en/
9 http://virus.netqin.com/en/

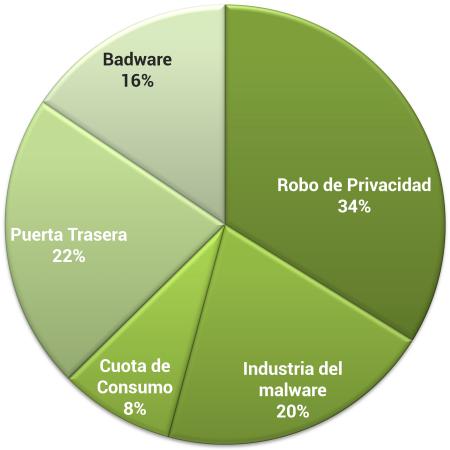


Figura 1.2: Gráfica que representa la misión que tiene el *malware* dentro de la plataforma Android.

Security sobre este mercado negro [Lab11], el precio de una tarjeta de crédito en el mercado negro va desde los 2$ hasta los 90$, mientras que las cuentas de tiendas on-line y pasarelas de pago con saldo verificado oscilan entre los 80$ y los 1.500$. Ya se ha publicado algún artículo de divulgación que han profundizado en este tipo de economía [Alv11], con el fin de concienciar a la gente sobre este tipo de actividades.

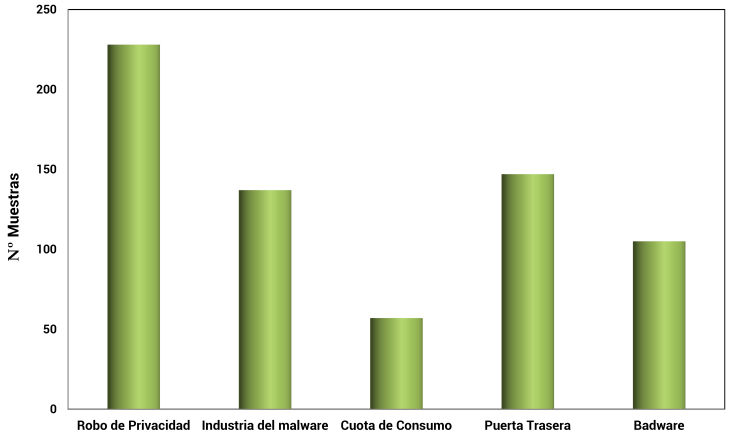


Figura 1.3: Representación el número de muestras de *malware* clasificadas por objetivo.

En la Figura 1.3 tenemos una visión distinta de los mismos datos. Esta gráfica representa el número de muestras que existen

en función del objetivo. Se debe tener en cuenta que las categorías son disjuntas, por lo que cada muestra de *malware* puede pertenecer a más de una categoría, por lo que no suman el total de muestras. Así, también se puede apreciar de forma mucho más clara la intencionalidad del malware: sobre un total de 280 muestras identificadas, 228 buscan obtener la información personal. Estos datos revelan cuál es el principal peligro del que se debe proteger a los *smartphones*: el robo de datos privados.

El malware para Android ha crecido un 687 % en 3 meses.

La Figura 1.4 representa el crecimiento del *malware* en los últimos meses. Cabe resaltar el incremento que se ha producido desde las navidades de 2010 hasta julio de 2011. En los últimos meses se ha producido un crecimiento del 2.000 % en el número de muestras. En esta figura también podemos observar la explosión de muestras de *malware* aparecidas durante el mes de marzo, así como el repunte exhibido a partir de mayo, tras un mes sin un crecimiento.

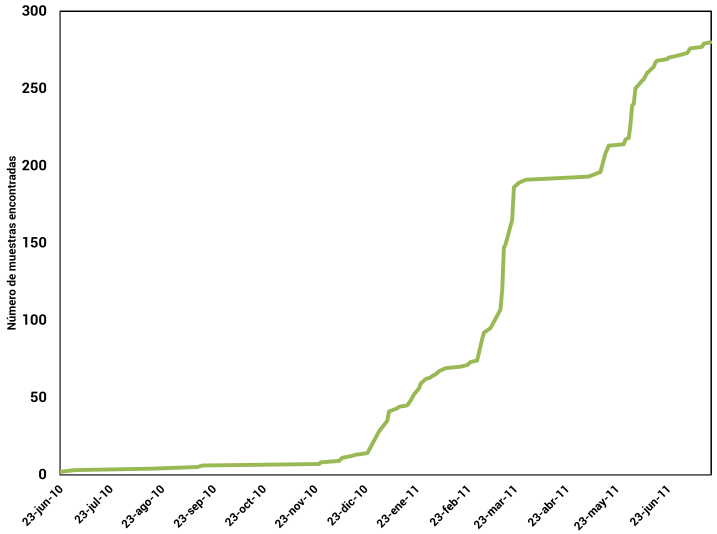


Figura 1.4: Crecimiento del *malware* en la plataforma Android.

El crecimiento del software malicios en la plataforma Android está siendo exponencial.

De todos estos datos podemos extraer varias conclusiones. El crecimiento en los últimos meses es extraordinario, y la tendencia no parece sino acrecentarse. Por otro lado, la peligrosidad del *malware* y su sofisticación está incrementándose, dificultando las tareas de detección y limpieza. Finalmente, el objetivo principal que buscan estas piezas de software es, principalmente y como se ha apuntado con anterioridad, el de obtener los datos privados del usuario almacenados en el dispositivo móvil, a fin de obtener rédito económico vendiéndolo en el mercado negro.

1.5 HIPÓTESIS Y OBJETIVOS

Frente a esta situación, y con el problema claramente identificado, hemos planteado la siguiente hipótesis fundamental:

> *«Es posible, mediante el uso de algoritmos supervisados de inteligencia artificial y minería de datos, hacer una gestión inteligente, automática, y efectiva de la seguridad en las aplicaciones de los teléfonos smartphones, tal que libere al usuario de parte de la responsabilidad de la gestión de la seguridad del mismo.»*

La intención que subyace bajo esta hipótesis es la de demostrar que es posible desarrollar un mecanismo de seguridad que permita mantener la funcionalidad de este tipo de dispositivos sin por ello renunciar a la defensa frente a las constantes amenazas de seguridad a las que deben enfrentarse. Para ello, hemos desarrollado una serie de objetivos principales que son en los que desglosan la hipótesis.

1. Conocer y categorizar la información que se almacena en los dispositivos móviles inteligentes o *smartphones*.
2. Identificar las amenazas de seguridad que se ciernen sobre estos dispositivos.
3. Desarrollar metodologías y procedimientos que sirvan para proteger estos dispositivos sin necesidad de la intervención del usuario.

Por otro lado, estos objetivos principales se desarrollan a través de una serie de objetivos operativos, que son los siguientes:

- *Desarrollo de un modelado de amenazas en teléfonos inteligentes*. Este modelado de amenazas sirve para obtener una visión más amplia de cuáles son las problemáticas que, desde el punto de vista de la seguridad, deben afrontar estos dispositivos.
- *Identificación de las características más relevantes a la hora de analizar aplicaciones*. Esto servirá para centrarse en las características más representativas a la hora de analizar las aplicaciones.
- *Desarrollo de nuevas técnicas de detección de malware sobre la plataforma*. En este punto se buscará desarrollar nuevas técnicas para la detección de *malware* sobre estos dispositivos.

1.6 METODOLOGÍA DE LA INVESTIGACIÓN

Figura 1.5: Metodología de la investigación.

La metodología busca una continua retroalimentación, para mejorar el resultado.

Dada la rama de la ciencia en la que se encuentra ubicada esta tesis, cuya evolución avanza a un ritmo trepidante, ha sido necesaria la utilización de una metodología de investigación que se adecúe a esta velocidad. Por ello, es necesario un ciclo continuo de revisión del estado de la técnica de la materia, tanto desde el punto de vista científico, como de la realidad que, en materia de seguridad, afrontan estos dispositivos.

Para todo ello, hemos desarrollado una metodología cíclica basada en una interacción constante con todos los elementos basada en los siguientes puntos:

1. Actualización de nuestro conocimiento sobre la materia, basado en las publicaciones de la comunidad científica en este ámbito, de igual manera que en los incidentes de seguridad que van surgiendo, así como en un constante contacto con expertos en el ámbito de la seguridad en teléfonos inteligentes.
2. Diseño de la solución que solvente los problemas en los que nos hemos centrado.
3. Experimentación y evaluación, para medir de forma cuantitativa el resultado obtenido.
4. Validación con la comunidad científica, de tal forma que se determine la relevancia de la investigación.

5. Adaptación con la retroalimentación obtenida por la comunidad científica.

6. Validación de los resultados obtenidos con muestras reales, para obtener una validación real de la investigación.

7. Transferencia del conocimiento obtenido durante el proceso a la sociedad, a través de comunicaciones en conferencias y revista internacionales.

La Figura 1.5 representa esta metodología cíclica, la cual se alimenta de todo el *feedback* obtenido por la sociedad y la comunidad científica, y devuelve el trabajo resultante.

1.7 ESTRUCTURA DE LA TESIS

Esta tesis doctoral esta organizada de la siguiente manera.

El *capítulo 1* ha introducido la problemática que se quiere abordar en este trabajo doctoral, abarcando tanto la descripción de lo que se considera teléfono inteligente como la situación actual de este tipo de dispositivos. También incluye la hipótesis de partida de esta tesis doctoral, así como la metodología utilizada para intentar refutarla.

El *capítulo 2* analiza el estado de la técnica en el ámbito de la seguridad. En concreto, en un primer lugar se centra en el ámbito de los modelados de amenazas, y en cuáles han sido los avances más significativos en los últimos años. Posteriormente, se centra en aspectos relacionados con la seguridad en dispositivos móviles, desde dos puntos de vista distintos. También se analiza la seguridad de este tipo de terminales desde el punto de vista del sistema operativo. Tras ello, se hace un breve repaso a las investigaciones más relevantes dentro del ámbito de la seguridad en este tipo de dispositivos. Posteriormente, se hace un repaso a la literatura existente en la detección de *malware* desde tres enfoques distintos: el análisis estático, el dinámico y el análisis híbrido. Finalmente, se hace un repaso a las recientes investigaciones centradas en el ámbito de la seguridad sobre la plataforma Android.

El *capítulo 3* desarrolla el modelado de amenazas en *smartphones* desarrollado. Para ello, en primer lugar se hace un modelado de la seguridad para, posteriormente, explicar el grafo de ataques y el concepto de caminos de ataques. En este punto se plantea el desarrollo del modelo. También se presenta la primera

biblioteca de ataques sobre este tipo de terminales. En ella se recogen todos aquellos aspectos relacionados con la seguridad que se han identificado tras un estudio de las diferentes situaciones acaecidas en los últimos tiempos sobre este tipo de dispositivos. Finalmente, se aplica la biblioteca de ataques al modelo generado y se extraen conclusiones de los resultados obtenidos.

El *capítulo 4* analiza en profundidad el sistema operativo Android. Para ello, en primer lugar, se hace un análisis del árbol de directorios y de cómo funciona a nivel interno. Asimismo se hace un análisis exhaustivo de las aplicaciones de Android, de su estructura y de están organizadas internamente. Posteriormente se realiza un breve repaso al desarrollo de aplicaciones de la plataforma, desde el punto de vista de la seguridad. Tras ello, se detalla el modelo de instalación y ejecución de las aplicaciones en Android. Para finalizar, se analiza el modo en que se comunican las aplicaciones entre sí.

El *capítulo 5* presenta dos nuevas aproximaciones a la detección de aplicaciones maliciosas dentro de la plataforma Android. En primer lugar, se conforma el conjunto de datos necesario para llevar a cabo la validación de los sistemas planteados. Una vez obtenidas y categorizadas las muestras, se determina qué características dentro de los ejecutables son elegidas como predictores a la hora de determinar si una muestra es maliciosa o no lo es. En segundo lugar, se presenta el análisis estático basado en anomalías. Este análisis determina qué predictores son correctos para que, todas aquellas muestras que se distancien de la normalidad, sean clasificadas como maliciosas. Posteriormente, se aplicará otro enfoque, basado en técnicas de aprendizaje automático, para la detección de software malicioso, usando como base los predictores extraídos con anterioridad. Por último, se analizarán los resultados obtenidos.

Finalmente, el *capítulo 6* expondrá las conclusiones extraídas de este trabajo doctoral. Posteriormente, se realizará una evaluación de las limitaciones que hemos detectado en el proceso investigador. Para concluir, definiremos las líneas de trabajo futuro en las que consideramos que se debe trabajar en este ámbito.

2 ESTADO DE LA TÉCNICA

«La ciencia más útil es aquella cuyo fruto es el más comunicable.»

L. Da Vinci
(1452 – 1519)

ÍNDICE

Inspirados por la investigación que dentro del área de la seguridad, se ha llevado a cabo en el área de los ordenadores personales la comunidad científica dota cada día de más importancia a la investigación en los dispositivos móviles. Si bien es cierto que se trata de una etapa temprana dentro de la investigación, la experiencia adquirida en otro tipo de plataformas sirve para extrapolarla a estos nuevos dispositivos. Sin embargo, estos dispositivos poseen una serie de características que los hacen especiales.

En este capítulo vamos a repasar las investigaciones que se han llevado a cabo en los últimos años en el ámbito de la seguridad, tanto desde una perspectiva general como, más detalladamente, en el ámbito de los dispositivos móviles inteligentes.

Al finalizar el capítulo, tendremos una visión clara de cuáles son los aspectos que han sido objeto de investigación dentro del área, así como de los resultados obtenidos en las mismas. También veremos cuál ha sido el recorrido que se ha seguido en otros dispositivos más comunes, como los PCs (de la locución inglesa *Personal Computer*), para intentar vislumbrar el camino que la investigación puede seguir en los *smartphones*.

2.1 MODELADOS DE AMENAZAS

En la literatura se han dado varios modelos que pretenden caracterizar las amenazas, muchos de los cuales se han centrado en el sector de la informática.

La aproximación de Microsoft se desarrolló como herramienta para ayudar a los administradores de seguridad.

Una de las primeras aproximaciones es la desarrollada por Microsoft, denominada STRIDE/DREAD [HLOF03]. Esta herramienta ayuda a los administradores de seguridad a identificar qué elementos de la lógica del negocio pueden verse afectados por distintos ataques. En concreto, se reflejan los siguientes ataques: suplantación (*spoofing*), alteración (*tampering*), repudio, divulgación de la información, denegación de servicio y elevación de privilegios. Este estudio se hace a través del análisis del flujo de datos y sus posteriores cálculos de riesgos. Comparado con T-MAP [CB07], este método enfoca el modelado de amenazas en el software desde un punto de vista más general que el de los sistemas basados en el coste.

Por otro lado, Trike[1] es una aproximación *open source* que se define como «un marco conceptual unificado para la auditoría de seguridad desde una perspectiva de gestión del riesgo a través de la generación de modelos de amenazas de una manera confiable y repetible.». Para ello, provee de un lenguaje común que permite «de forma completa y precisa describir las características de seguridad de un sistema, tanto de su arquitectura de alto nivel a sus detalles de implementación de bajo nivel». Este modelado se hace a través de los árboles de ataques, para lo cual se aprovecha de un conjunto de herramientas desarrolladas a tal efecto.

Otra aproximación al modelado de amenazas es la que plantearon Chen *et ál.* [CBS07], basada en T-MAP, el cual cuantifica las amenazas calculando el peso total de los ataques relevantes para sistemas COSTS (de la locución inglesa *Commercial Off The Shelf*)[McK99]

2.2 SEGURIDAD EN DISPOSITIVOS MÓVILES

La seguridad en los dispositivos móviles ha sido objeto de investigación en los últimos años, pero cuando han despuntado en las capacidades y las ventas de estos dispositivos, la investigación en esta área ha sido más destacada. En este sentido, la investi-

1 http://octotrike.org/

gación sobre la seguridad de estos dispositivos ha cubierto una gran variedad de áreas.

La definición de lo que se denomina como «dispositivo móvil» incluye una gran variedad de dispositivos. Si bien actualmente esta definición suele relacionarse con los *tablets* y los *smartphones*, son muchos los dispositivos que, aún hoy, pueden acogerse a esta definición. Así, los ordenadores portátiles o las *Personal Digital Assistants* (PDAs), también podría entrar en esta definición.

2.2.1 Seguridad del sistema operativo

En los años 70 Salter enumeró las 9 áreas principales de investigación en seguridad.

La seguridad en los sistemas operativos es uno de los campos más amplios y complejos de las ciencias de la computación. Algunos de los conceptos de base fueron definidos en los años 60 y 70, cuando se diseñaron ordenadores *mainframes* y era necesario un sistema operativo que los gobernara. En aquella época, Salter [Sal74] enumeró las 9 áreas principales de investigación de la seguridad de los sistemas operativos, que incluían:

- Ejercicios de penetración de sistemas.
- Estudio de las interfaces de usuario.
- Prueba de correctitud.
- Modelos matemáticos de protección del núcleo.
- Mecanismos de protección.
- Seguridad en la comunicación de los datos.
- Recursos de base de datos.
- Mecanismos de autenticación.
- Problemas operacionales del Departamento de Defensa.

Si bien el último elemento de la lista demuestra que se trata de una informática de otra época, no es menos cierto que muchos de los problemas que se reflejan siguen estando vigentes en la seguridad de los sistemas operativos de hoy en día. El trabajo de Jaeger [Jae08] ofrece una perspectiva histórica de la evolución de estos problemas desde la época de los *mainframes* hasta la plataformas más actuales, como pueden ser toda la familia UNIX o Microsoft Windows.

En los sistemas operativos tradicionales, la protección se representa como una Matriz de Control de Acceso.

En los sistemas operativos tradicionales, la protección puede ser representada como una *Matriz de Control de Acceso* (de la locución inglesa *Access Control Matrix* (ACM)) [Lam74]. En él, se considera que cada aplicación se ejecuta como un proceso, y que cada proceso tiene definida la información y los recursos a los que puede acceder, creando de esta manera un dominio de protección. Podemos ver un ejemplo de esta matriz en la tabla 2.1.

	f	g	p	q
p	rwo	r	rwx	r
q	—	w	w	rwxo

Tabla 2.1: Ejemplo de matriz de acceso de control. En este caso tenemos 2 sujetos (p y q), cuatro objetos (f, g, p y q) y cuatro permisos de acceso (r, w, x y o).

La matriz está construida por un conjunto de sujetos, por ejemplo los procesos, y de objetos, como los ficheros, con la matriz de elementos poblada por acciones (p. ej. leer o escribir). Debido a que en multitud de ocasiones esta matriz está ampliamente poblada, la matriz se convierte en una «lista de control de acceso» o *Access Control List* (ACL), en la que a cada objeto le corresponde una ACL.

El control de acceso discrecional es comúnmente conocido como una política gestionada por el usuario.

El «control de acceso discrecional» (de la locución inglesa *Discrectional Access Control* (DAC)) determina la forma de acceso a los recursos en base a los propietarios y a los grupos a los que pertenecen esos objetos. Se denomina discrecional porque un sujeto con permisos sobre un objeto puede transmitirlos a otro sujeto. Por lo tanto, la política de control de acceso discrecional se describe comúnmente como una política gestionada por el usuario. Si las transiciones políticas no se limitan cuidadosamente, los objetivos de seguridad pueden ser eludidos. La capacidad de decisión por parte del sujeto hace que la posibilidad de que una política discrecional pueda llegar a un estado inseguro sea desconocida. Esta condición se conoce como el *problema de la seguridad* [HRU76].

Por otro lado, el «control de acceso obligatorio» (de la locución inglesa *Mandatory Access Control* (MAC)) se basa en un conjunto de reglas de autorización, denominadas «políticas». Éstas determinan si una operación sobre un objeto realizada por un sujeto está permitida o no. La gestión de estas políticas se restringe a

una entidad administrativa, por lo que el *problema de seguridad* no puede darse. Existe una gran cantidad de literatura centrada en MAC y la forma en la que puede ofrecer garantías de seguridad probadas [FKC+92, SCFY96, EOM09].

La aplicación de políticas MAC es demasiado estricta para ejecutarlas de forma práctica.

La base para la aplicación de políticas MAC es un monitor de referencia [And72]. Desde una perspectiva purista, un monitor de referencia requiere: (i) la mediación completa de todas las operaciones de seguridad sensibles, (ii) garantía de seguridad del mecanismo de aplicación, y (iii) verificabilidad de la mediación completa y garantía de seguridad. Sin embargo, realmente pocos sistemas logran o intentan alcanzar esta verificabilidad rigurosa debido a las limitaciones prácticas. Por lo tanto, los errores de programación que llevan a una escalada de privilegios o a su circunvención a menudo pasan desapercibidos hasta que es demasiado tarde.

2.3 PRIVACIDAD EN DISPOSITIVOS MÓVILES

El flujo de la información se puede representar como un grafo.

Históricamente, el acceso de control obligatorio ha sido sinónimo de «flujo de control de información» (*Information Flow Control* (IFC)). De manera similar a la matriz de control de acceso de Lampson [Lam74], los modelos de IFC cuentan con un conjunto de sujetos $\mathcal{S}$ y de objetos $\mathcal{O}$, siendo representadas las acciones de lectura y escritura como flujos ($\rightarrow$). Es decir, que si el objeto $o \in \mathcal{O}$ escribe un objeto $s \in \mathcal{S}$, se denota $s \rightarrow o$, de la misma forma que si lo lee se denota $s \leftarrow o$. De esta manera, se conforma un un grafo de flujo de información $G = (\mathcal{V}, \mathcal{E})$, donde los vértices $\mathcal{V}$ son la unión de los sujetos y los objetos ($\mathcal{V} = \mathcal{S} \cup \mathcal{O}$), y $\mathcal{E}$ son los flujos.

Los modelos de seguridad basados en flujos de la información etiquetan cada objeto y cada sujeto con una «clase» de seguridad. De cara a fijar objetivos verificables de seguridad, Denning [Den76] organizó las clases en forma de red. Esta red define los flujos permitidos entre las clases de seguridad. Bell y LaPadula [BL73], definen la «seguridad multi-nivel» (Multi-Level Security (MLS)), que define las redes de confidencialidad. En MLS, las redes codifican las políticas de seguridad simple (no lectura) y la «-seguridad» (no escritura) mediante el uso de clases de seguridad jerárquicas (p. ej., alto secreto).

Por otro lado, Biba [Bib77] propone la doble integridad, la cual está basada en codificar la integridad simple (no lectura) y la «$*$-integridad» (no escritura) dentro de la red. Mientras que la

confidencialidad y la integridad de los modelos de flujo de información se pueden aplicar al mismo tiempo, se utilizan conjuntos disjuntos de etiquetas de seguridad. Por otro lado, los sujetos sólo pueden leer y escribir en sus propias clases de seguridad. Por ejemplo, mientras que la MLS utiliza las clases de seguridad militar, un modelo de integridad puede utilizar clases de seguridad tales como: confianza, sistema o usuario de la aplicación.

A diferencia de este enfoque más teórico, otros investigadores han optado por un enfoque más práctico, centrándose en la confiabilidad del proceso. El modelo de integridad de Clark-Wilson [CW89] trata de proveer de verificabilidad a los procesos a través de la certificación de todos los procesos confiables. Sin embargo, este modelo sigue siendo demasiado estricto para mejorar su aplicabilidad, por lo que surgieron otros enfoques que relajan estas condiciones. La versión ligera del modelo de Clark-Wilson, CW-lite [SJS06] no exige la certificación de todo el proceso, sino que se basa en las interfaces definidas para el filtrado a la hora de actualizar o descartar peticiones de lectura de los procesos que requieren un nivel de integridad menor. Otro modelo que se centra en la usabilidad del mismo es el *Usable Mandatory Integrity Protection* (UMIP)[LMC07], definiendo la confiabilidad en términos de tipos de flujos de información, manejando las excepciones a través de reglas. En esta misma línea, el modelo de integridad práctica proactiva (PPI) [SSPK08] utiliza una combinación de etiquetas de integridad y políticas para garantizar la integridad del sistema, sin perder la flexibilidad del mismo.

Los flujos de control descentralizados buscan superar las limitaciones de los modelos de procesos confiables.

Existen otras aproximaciones para sortear las limitaciones de estos modelos de procesos confiables. En esta línea se enmarcan los flujos de control descentralizados (*Decentralized information flow control* (DIFC)) [EKV+05, ZBWKM06, KYB+07], el cual está basado en el modelo de etiquetas descentralizado (*Decentralized Label Model* (DLM)) [ML00]. Esta aproximación fragmenta la confianza en distintas clases no comparables entre sí, y considera que el desarrollador de aplicaciones es el que tiene el contexto adecuado para administrar la política de protección. Las clases, denominadas «tags», son creadas y administradas por las aplicaciones. Se define la red de seguridades como un conjunto de etiquetas sujeto y objeto ordenadas parcialmente. Se dice que un sujeto «posee» una etiqueta si se puede agregar y quitar permisos a la misma. Por otra parte, los sujetos pueden delegar, añadir y eliminar las capacidades de etiquetas para gestionar la confianza.

Todos estos enfoques buscan asegurar los flujos de información del sistema, garantizando que la información de la que disponen las aplicaciones sea la adecuada. Hay otros investigadores que han propuesto modelos donde son los propios usuarios los que identifican una acción como no confiable. TRON [BBS95] define «capacidades» para cada uno de los ficheros y directorios, las cuales pueden ser desplazadas a procesos clonados para limitar el dominio de protección de las aplicaciones no confiables. Otro enfoque busca la mitigación del daño mediante el uso de diferentes identidades. Este concepto de identidades a nivel de aplicación apareció inicialmente en PACL's [WCO+90], donde las lista de control de acceso de los programas eran definidas a nivel de fichero. Polaris [SKY+06] usa perfiles predefinidos que ejecutan automáticamente las aplicaciones con una cuenta de usuario diferentes. Existen otras propuestas similares, como FileMonster [SHG02] o PinUP [EMJ08, ERS+07], en las que todas las aplicaciones corren bajo distintos usuarios. Android sigue también este modelo, ejecutando cada aplicación con un único usuario, como se explica en el punto 4.5.1.

2.4 DETECCIÓN DE MALWARE

La palabra *malware* proviene de la combinación de dos locuciones inglesas: *malicious* y *software*. Existen varias definiciones para este tipo de software. Primeramente, Rutkowska [Rut06] lo definió de la siguiente manera:

> *«Malware es un trozo de código que cambia el comportamiento del núcleo del sistema operativo o de algunas aplicaciones de seguridad sensibles, sin permiso del usuario y de tal manera que es imposible de detectar dichos cambios, utilizando características documentadas del sistema operativo o de las aplicaciones (por ejemplo, API).»*.

McGraw y Morriset [MM00], por su parte, centran su definición en la forma de comportarse de este tipo de software:

La palabra malware proviene de la combinación de dos locuciones inglesas: malicious y software.

> *«Cualquier trozo de código añadido, cambiado, o borrado de un sistema software para causar intencionalmente daño o subvertir la función del sistema.»*.

Otra definición ofrecida por Christodorescu y Jha [CJS+05] se centra en definir el objetivo del software:

> *«Un programa cuyo objetivo es malévolo».*

Con todo ello, nuestra definición de *malware* es la siguiente:

> *Malware es aquel programa o trozo de código que ejecutado, añadido, cambiado o borrado de un sistema de software cambia el comportamiento del mismo y cuyo objetivo final es causar daño intencionalmente.*

La detección de *malware* ha sido un área de investigación relevante en los últimos años. Si bien esta investigación se centró principalmente en los ordenadores de escritorio [IM07], en los últimos tiempos, con la mejora en las capacidades de los teléfonos móviles, éstos han sido objeto de varias investigaciones [BHSP08, YE05, SPL+09]. En concreto, Symbian[2], dada la amplia penetración en el mercado que tuvo en su día, varias investigaciones en este ámbito se han centrado sobre esta plataforma. [SCCA09, VH08, LYZC09].

El malware se puede categorizar en virus, gusanos y caballos de Troya.

En cuanto a la categorización del *malware*, podemos dividirlo en tres categorías importantes: *virus*, *gusanos* y *caballos de Troya*. Los virus [CAJ07] son aquellas aplicaciones que se replican a sí mismas, normalmente tras una acción concreta (p. ej., el usuario ejecuta un programa). Los gusanos [DP07], por su parte, tienen el mismo objetivo que el virus, replicarse a sí mismo, sin embargo, utiliza la Red para enviar las copias. Por último, los caballos de Troya o *troyanos* [DP07] son programas escondidos dentro de programas legítimos que ejecutan fragmentos de código maliciosos.

Además, dos conceptos más deben ser definidos. En primer lugar definimos *vector de infección* como aquellas técnicas que se usan para distribuir la aplicación maliciosa (p. ej., explotación de una vulnerabilidad del navegador o recepción del ejecutable por correo electrónico). Por otro lado, la *carga de la infección* (en inglés *infection payload*) representa el contenido real que se utiliza para dañar la máquina de las víctimas (p. ej., troyano introducido en un juego).

De cara a la detección de las muestras maliciosas, se han desarrollado dos enfoques principales. El primero es el enfoque estático, que busca recuperar características y propiedades de los programas para, así, detectar los comportamientos maliciosos en base a ellos. El segundo es el enfoque dinámico, el cual monitoriza la ejecución del programa para determinar si su comportamiento es malicioso o no. Las ventajas y desventajas de cada uno de los enfoques se pueden ver en la tabla 2.2

2 http://www.symbian.org/index.php

A continuación detallaremos algunas investigaciones relevantes que han sido llevadas a cabo en el área. Moser et ál. [MKK07] presentaron un esquema de ofuscación de binario que profundizaba en la idea de constantes opacas, es decir, primitivas que cargan una constante en el registro. A partir de ese registro una herramienta de análisis no es capaz de determinar su valor, por lo que el software es capaz, entre otras cosas, de ocultar el flujo de control, así como ocultar el acceso a variables locales o globales. De esta manera, se demuestra que, en caso de *malware* muy evolucionado y complejo, el análisis estático puede no ser suficiente para identificar por sí sólo el *malware*. Por ello, existe un tercer enfoque menos habitual, que es el híbrido, que combina los dos enfoques anteriores [RKLC03].

Moser et ál. profundizaron en la idea de constantes opacas.

A continuación se detallan algunas de las investigaciones más relevantes utilizando el enfoque estático (en el punto 2.4.1) y el enfoque dinámico (en el punto 2.4.2), centrado en el área de los *smartphones*, que es el objeto de esta investigación doctoral.

	Estático	Dinámico
Ventajas	• Permite revelar el comportamiento del programa bajo condiciones inusuales. • En general, es más rápido que el dinámico. • Sirve para hacerse una idea general aproximada del comportamiento de la muestra software.	• Permite el estudio durante la ejecución del comportamiento del software. • Permite conocer a fondo el comportamiento del software.
Desventajas	• Es imposible predecir el comportamiento general del programa.	• Requiere de muchos más recursos. • Infecta la maquina en cada ejecución. • No se recorren todos los posibles caminos de la ejecución

Tabla 2.2: Ventajas y desventajas de los análisis estático y dinámico de *malware*.

2.4.1 Análisis estático aplicado a teléfonos móviles

Varias investigaciones se centraron en el estudio de los datos de consumo de energía.

Varias publicaciones utilizan el poder de los datos que contiene el ejecutable para detectar ataques en smartphones. Uno de los enfoques dentro del análisis estático más prometedor es el que se centra en el estudio de los datos de consumo de energía [KSS08, JD04, BNC+08, NMHH05] a fin de determinar si se está produciendo un ataque. Andromaly [SKE+12] implementa un sistema de detección de anomalías sobre Android que se basa en distintos parámetros extraídos en tiempo de ejecución, como son la carga de la CPU, los eventos de entrada, y el consumo de energía. En la misma línea, Nash *et ál.* [NMHH05] plantean los posibles algoritmos para detectar ataques contra la batería del *smartphone*. Otras aproximaciones, en cambio, utilizan vectores de características y técnicas basadas en firmas [VH08, CWYL07, SM97, MHH06, BNC+08, SPL+09].

Kruegel et ál. crearon la definición del desensamblado estático.

El análisis estático de ejecutables es una técnica bien explorada. La definición del desensamblado estático de un ejecutable la llevaron a cabo Kruegel *et ál.* [KRVV04]. Por otro lado, Wang *et ál.* [WWH08] presentaron métodos de minería de datos para discriminar entre ejecutables benignos y malignos. Para ello, utilizaron máquinas de soporte vectorial (de la locución ingles *Support Vector Machine* (SVM)) (este algoritmo se explica en detalle en el punto 5.5.4). Schultz et ál. [SEZS01] también aplicaron distintas técnicas de aprendizaje automático contra un conjunto de datos con aplicaciones maliciosas.

Christodorescu *et ál.* [CJ06] desarrollaron la herramienta SAFE (*Static Analyzer For Executables*), que utilizaba el análisis estático para la creación de autómatas basados en el código en ensamblador del programa con el fin detectar la actividad maliciosa. Concretamente, trataba de solventar el problema de la ofuscación. Sin embargo, la aplicabilidad de este enfoque a terminales móviles presenta una gran dificultad, ya que es necesario una gran capacidad de cómputo para su aplicación, lo que puede causar una gran merma en el consumo de energía del terminal.

Esta misma problemática se da en el enfoque que presentaron Bergeron *et ál.* [BDD+01] Estos investigadores propusieron un enfoque semántico de análisis del código binario. Su propuesta se separa en 3 etapas distintas:

1. Creación de una representación intermedia.

2. Análisis de comportamiento basado en el flujo de ejecución

3. Verificación estática de los comportamientos críticos contra políticas de seguridad.

Otro enfoque cuya aplicabilidad a los *smartphones* es complicada es la que presentaron Krugel *et ál.* [KRV04]. En su investigación, utilizan el análisis estático de binarios para detectar *rootkits*, es decir, aplicaciones maliciosas con acceso de administrador continuo que se mantiene oculto y que corrompe el funcionamiento normal del sistema operativo o de otras aplicaciones, a nivel de núcleo. Para ello, los investigadores se fijaron en las secuencias de instrucciones antes de que los módulos correspondientes sean cargados en el núcleo. Según reflejan sus resultados, no se produjo ningún falso positivo con el prototipo que desarrollaron a la hora de detectar todos los *rootkits* que conformaban el conjunto de datos. Sin embargo, como reflejan en su investigación, el problema de su aplicación viene dado por «la explosión exponencial de posibles caminos que deben ser seguidos», por lo que su aplicabilidad a dispositivos con recursos tan limitados como los *smartphones* se antoja complicada. Estos caminos son creados por el análisis de control de flujo, que genera los estados de las máquinas que está siendo investigada.

Santos *et ál.* presentaron NOA [SBUP+], un algoritmo que se basa en sistemas de recuperación de la información y utiliza un modelo de espacio vectorial. El ejecutable se representa con secuencias de códigos operacionales, denominados *opcodes*: su frecuencia de aparición y la relevancia de los *opcodes* que componen la secuencia, formando un vector para representarlo en el espacio vectorial. Posteriormente, se utilizaron métodos de aprendizaje automático para clasificar las muestras. Sin embargo, la complejidad de este sistema hace inviable migrarlo a los *smartphones*.

Santos et. ál. aplicaron un modelo de recuperación de la información a la detección de malware.

Por otro lado, Ugarte-Pedrero *et ál.* [UPSB11] presentaron un método para pre-filtrar las aplicaciones que han sido empaquetadas de las que no. El empaquetado de las aplicaciones es un método muy utilizado tanto por aplicaciones con fines benévolos como aplicaciones con fines más oscuros. Sin embargo, el empaquetado de las aplicaciones dificulta enormemente el análisis estático de las aplicaciones, por lo que es importante distinguir las aplicaciones que están empaquetadas de las que no. Este tipo de técnicas no han sido muy explotadas en el entorno de los *smartphones*.

Otros investigadores optaron por analizar la secuencia de llamadas al sistema. Warrender *et ál.* [WFP99] utilizaron este enfoque a fin de definir comportamientos anómalos. Sus pruebas se

realizaron utilizando 4 métodos distintos: *sequence time-delay embedding* (STIDE) [TM02], *stide with frequency threshold* (T-STIDE) [WFP99], *repeated incremental pruning to produce Error Reduction* (RIPPER) [Coh95], *hidden markov models* (HMMs) [Edd96].

Mutz *et ál.* [MVVK06] afirman que aquellas investigaciones que utilizan un enfoque de llamadas al sistema no suelen considerar los argumentos de la llamada, lo que permite a los atacantes crear métodos para evadir la detección. Ellos proponen dos mejoras principales sobre el sistema existente:

Mutz et ál. identificó una debilidad en los enfoques de llamadas.

- La primera mejora se centra en aplicar varios modelos de detección a los argumentos de llamadas del sistema, a fin de analizarlos.
- La segunda mejora describe un método sofisticado para la agregación de los resultados de todos los modelos de detección aplicados.

Este método se basa en la utilización de redes bayesianas para la clasificación, y obtiene una mejora en la precisión de la detección y una mayor resistencia a los métodos que intentan evitarlos.

Por otro lado, otros enfoques buscan analizar el código fuente. Wagner *et ál.* [WD01] presentan un enfoque de extracción de autómata no deterministas finitos (de la locución inglesa *Non-Deterministic Finite Automaton* (NDFA)) que son generados a partir del código fuente de la aplicación. Entonces, las llamadas al sistema de la aplicación se elevan para el cumplimiento del modelo creado en tiempo de ejecución. Los resultados presentados son todavía preliminares, siendo la sobrecarga al sistema para la detección de ataques alta.

Por su parte, Liu *et ál.* [LBV05] propone un sistema que enriquece la detección estática, usando la información de la pila de llamadas para capturar el control de flujo del programa. En su investigación, utilizan el análisis estático para crear un modelo base, utilizando un aprendizaje dinámico para las llamadas de las funciones. Esta combinación demuestra ser más efectiva que utilizar únicamente el análisis estático y con una tasa menor de falsos positivos.

Liu et ál. proponen enriquecer la detección estática con la información de la pila de llamadas.

Gran parte de estos métodos, si bien se han demostrado efectivos, adolecen de las limitaciones impuestas por el sistema sobre el que se deben desplegar. Por ello, muchos de los sistemas que se han implementado a día de hoy sobre terminales móviles están basados en sistemas de firmas. En ellos, se identifica un patrón o una serie de características dentro de la muestra en función de los cuáles, se genera una firma que la identifica de forma

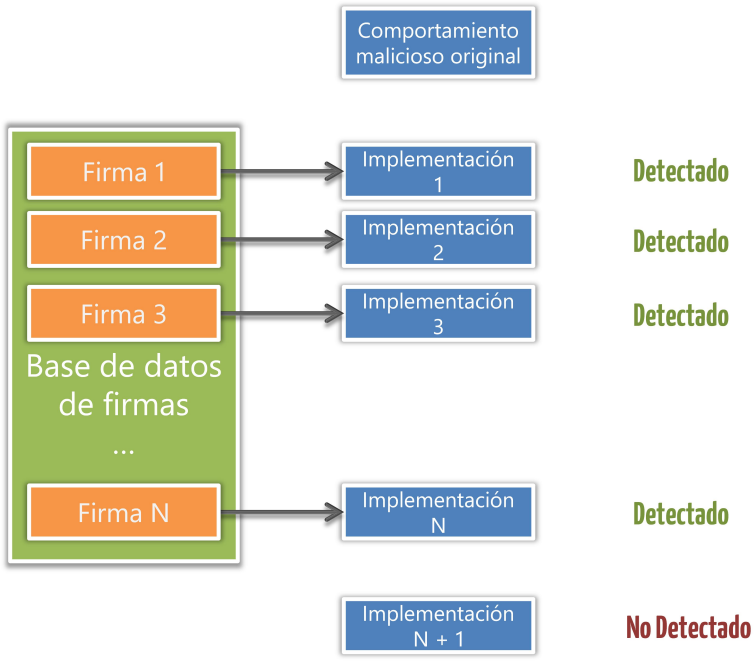


Figura 2.1: Los métodos de firmas se vuelven ineficaces a la hora detectar nuevas implementaciones de comportamientos maliciosos originales.

unívoca. Posteriormente, se actualiza una base de datos con esa firma en de tal manera que, al analizar una nueva muestra, se identifique con algunas de las muestras de la base de firmas. Estos sistemas tienen como ventaja que son realmente rápidos y apenas consumen recursos, ya que requiere de operaciones de una complejidad controlada. Además, la tasa de falsos positivos que genera es realmente baja. Un esquema del funcionamiento de estos métodos se puede ver en la figura 2.1.

Sin embargo, las nuevas variantes de cada muestra de *malware* requieren de una nueva firma para ser detectadas. En el entorno de ordenador de escritorio, y según el informe anual de Panda[3] correspondiente al año 2011, durante ese año se recibieron en sus oficinas 73.000 nuevas muestras cada día, para alcanzar la cifra de 26.000.000 de muestras en todo el año, lo que supone la cifra de muestras recogidas más altas de su historia. Estas cifras dan idea del trabajo que supone generar una nueva firma por cada una de las muestras que aparecen a diario.

Según un informe de Panda, el año pasado registraron más de 26 millones de nuevas muestras.

3 http://prensa.pandasecurity.com/wp-content/uploads/2012/01/Informe-Anual-PandaLabs-2011.pdf

En los entornos de *smartphones*, estas cifras están lejanas a día de hoy. Sin embargo, es necesario plantearse que nos dirigimos a un escenario similar al que se vive hoy en día en los entornos de escritorio, por lo que, si bien un sistema basado en firmas puede ser suficiente a día de hoy, cada vez está más cerca el día en que este sistema no sea el adecuado para este tipo entornos.

En un trabajo previo [SSL+12], utilizamos distintas características de aplicaciones de Android , como por ejemplo los permisos que requiere la aplicación, para posteriormente, clasificar las muestras en distintas categorías. En él generamos modelos con las características extraídas a fin de clasificar las aplicaciones entre un conjunto de 7 categorías distintas. Esta misma metodología puede ser aplicada para generar un clasificador que distinga las aplicaciones entre maliciosas o no.

2.4.2 Análisis Dinámico

En la misma línea, se han planteado distintos enfoques para el análisis dinámico, teniendo en cuenta las limitaciones con las que cuentan estos dispositivos. Son varios los enfoques que se han aplicado, siendo los sistemas de detección de intrusiones los más comunes [SM97, NMHH05, MHH06]. También se ha tenido en cuenta el consumo de energía a la hora de detectar intrusiones [KSS08, JMD06, JD04, BNC+08].

Por otro lado, la detección de comportamiento es otro de los enfoques en los que se ha avanzado en los últimos años [FHSL96, BHSP08, AL09, CBS05, SPL+09].

El enfoque de Cheng et ál monitorizaba los ficheros de log del terminal para detectar comportamientos maliciosos.

Cheng *et ál* [CWYL07] desarrollaron un sistema de monitorización del estado del sistema a través de los ficheros de log a fin de detectar comportamientos maliciosos. Su prototipo, instalado en la plataforma Windows Mobile 5, observaba distintos aspectos, como los logs de llamadas y SMSs o el identificativo de la celda a la que estaba conectado el teléfono. El proceso se realizaba a través del análisis estadístico y de la detección de anomalías de los datos monitorizados.

Dentro del análisis dinámico, es muy común encontrar, y especialmente en el ámbito de los *smartphones*, enfoques que implementan un «sistema de detección de intrusiones» (*Intrusion Detection System* (IDS)). Samfat y Molva [SM97] buscaron generar perfiles, a través de algoritmos de aprendizaje automático, que se utilizaban de un modo similar a las firmas que se utilizaban en el análisis estático. Con estos perfiles crearon un sistema

de detección de intrusiones distribuido que detectaba comportamientos abusivos.

Nash *et ál.* [NMHH05] diseñaron un IDS que se centraba únicamente en detectar ataques de denegación de servicio contra la batería. Para ello, tras recoger diversos parámetros de cada uno de los procesos (p. ej., acceso a los discos o carga de la CPU) detectaba aquellos procesos que eran candidatos a ser causantes de un ataque de este tipo, mediante el uso de la regresión lineal [MPV07]. Por otro lado, Miettinen *et ál.* [MHH06] combinaban la información proveniente del dispositivo y de la red, la cual recopilaban a través de distintos sensores distribuidos por ella. Utilizaban un motor de correlación a fin de combinar la información obtenida de ambas fuentes, preservando de esta forma la salud de toda la red.

Varios enfoques buscan detectar anomalías que sucenden en el funcionamiento del terminal como consecuencia del ataque.

Por otro lado, los IDS que se centran en la monitorización del uso de la batería han sido frecuentes en los últimos años. Buennenmeyer *et ál.* [BNC+08] presentaron un sistema similar al de Jacboy *et ál.* [JD04]. Ambas investigaciones buscaban monitorizar los cambios que se producen en el dispositivo a fin de detectar las anomalías que suceden en el funcionamiento del mismo como consecuencia de un ataque. Los datos que se obtienen se envían a un servidor que se encarga de realizar un perfil del dispositivo, en base al cual se detectan las anomalías. Por su parte, Jacboy *et ál.* [JD04] presentaron HIDE: *Host based Intrusion Detection Engine*. En primer lugar, observaba el comportamiento de la batería, intentado identificar comportamientos anómalos. En caso de detectarlo, mandaba al teléfono al modo suspensión, tras lo cual avisaba al usuario y al servidor proxy más cercano, guardando los logs con las causas del incremento en el uso de la batería.

Estos enfoques de detección de anomalías también se han dado en otras investigaciones. Por ejemplo, Forrester *et ál.* [FHSL96] propusieron un modelo de detección de anomalías donde la normalidad se define como un pequeño conjunto de correlaciones en las llamadas al sistema que hace el proceso. En sus experimentos demostraron que una secuencia corta de llamadas al sistema genera una firma estable de un comportamiento normal, ya que demostraron que tenía una baja varianza en condiciones normales y que era muy específica de cada uno de los procesos.

Asimismo, Bose *et ál.* [BHSP08] representaron el comportamiento malicioso en la observación del orden lógico de las acciones de la aplicación durante el tiempo. También propone una técnica de mapeo de dos etapas que construye firmas de com-

Bose et ál. utlizaron un enfoque basado en el orden lógico de las acciones.

portamientos maliciosos en tiempo de ejecución, a través de la monitorización de los eventos del sistema y las llamadas a la API. A fin de clasificar los eventos, se utiliza un clasificador basado en máquinas de soporte vectorial. A la hora de validar el modelo propuesto, obtuvieron una precisión en la detección de más del 96 %, realizando los experimentos sobre el sistema operativo Symbian. Este resultado lo obtuvieron combinando muestras de 25 familias distintas de *malware* con otras muestras sintéticas generadas a tal efecto.

La aproximación de Kang *et ál.* [KFH05] consideraba las secuencias de las llamadas del sistema como un problema de clasificación en una bolsa de llamadas del sistema. En esta bolsa, la frecuencia de las llamadas al sistema se almacena sin tener en cuenta el orden de las mismas. Los resultados experimentales, realizados con conjuntos de datos públicos, muestran que la información que da la frecuencia es lo suficientemente eficaz para discriminar entre secuencias normales y anormales.

Chaturvedi *et ál.* [CBS05] proporcionan una definición formal del flujo de datos del comportamiento del terminal, el cual estaba compuesto por el top de las trazas de las llamadas del sistema. Posteriormente crearon algoritmos para poder utilizar los modelos desarrollados. Este enfoque, si bien se demostró eficaz incluso con ataques muy sofisticados, necesita datos de entrenamiento de 1,4 a 4 millones de llamadas del sistema para cada traza.

Schmidt et ál. enviaban la información a un servidor para salvar las limitaciones computacionales de los terminales.

Schmidt *et ál.* [SPL+09] desarrollaron un sistema que, tras obtener distintas características del dispositivo, eran enviadas a un servidor remoto para realizar los cálculos necesarios, dadas las limitaciones que poseen estos dispositivos a nivel de *hardware*. Para la validación obtuvieron las 10 aplicaciones más utilizadas (según en un estudio de 2005) por los usuarios, extrayendo las características de los mismos.

Lee *et ál.* [LSS00] utilizaron una aproximación de minería de datos para descubrir patrones del sistema que describan el comportamiento del usuario. Para determinar comportamientos maliciosos, utilizaron la traza de las llamadas del sistema de «sendmail», que permite el envío de correos desde la línea de comandos, así como de «tcpdump», que realiza un volcado del tráfico de red que realiza el terminal.

Por su parte, Wang *et ál.* [WWH08] utilizaron máquinas de soporte vectorial para generar los modelos que permitan la detección del comportamiento de los usuarios. Para generar estos modelos, observaron el uso de las bibliotecas y de los métodos de la

API. Los enfoques basados en el comportamiento normalmente sufren de una alta tasa de falsos positivos; a la par que necesitan una cantidad significativa de potencia de procesamiento, el almacenamiento y la memoria. Los autores afirman que, basándose en su experimentación, su método obtiene una precisión del 99 %, pero no presentan la tasa detección correspondiente, lo que evita que se pueda realizar una evaluación de la calidad de los resultados.

2.4.3 Análisis Híbrido

La técnica descrita por Kirda *et ál.* [KKB+06] está centrada en entornos de escritorio, y se basa en una caracterización abstracta del comportamiento de una clase popular de los programas de *spyware*, y se aplica una combinación de análisis estático y dinámico de objetos binarios para determinar si un componente monitoriza las acciones de los usuarios e informa de sus conclusiones a una entidad externa.

Kirda et ál. caracterizaron el comportamiento de una clase popular de malware.

Su caracterización es resistente a la ofuscación, independiente de la imagen binaria particular, por lo tanto puede ser utilizado para identificar programas de software espía nunca antes vistos. Ya que es necesaria la interacción con el sistema operativo para activar uno de los componentes del software espía, se analizan las llamadas a las API de un componente que se puede utilizar para filtrar información del proceso actual, especialmente aquellos que se realizan en respuesta a eventos.

Por esa razón, utilizan un enfoque dinámico para supervisar la interacción del componente con el navegador y registrar todas las funciones que se invocan en respuesta a los acontecimientos, con el fin de determinar las regiones de código que son responsables de la gestión de eventos. Luego, utilizan el análisis estático para examinar en estas regiones la aparición de las llamadas del sistema, en particular aquellas relativas a la creación de hilos o temporizadores. Se basan en suponer que cualquier función de la API que existe en esa región puede ser invocada en respuesta a un evento. Por último, se han generado automáticamente una lista negra de llamadas a la API a partir de datos extraídos mediante el análisis de frecuencia.

2.5 SUMARIO Y CONCLUSIONES

En este capítulo hemos visto primeramente los distintos enfoques presentados para la gestión de la privacidad en los sistemas. Posteriormente, hemos analizado las investigaciones más relevantes enfocadas a uno de los aspectos que más está afectando a la privacidad y a la seguridad en los últimos tiempos: el *malware*. Nos hemos centrado especialmente en su aplicabilidad a los entornos de movilidad, como son los *smartphones*.

En concreto, hemos analizado las distintas aportaciones que se han realizado desde 3 aproximaciones distintas a este problema. En primer lugar, el enfoque estático, que busca analizar características de las aplicaciones para determinar la intencionalidad de las mismas.

Por otro lado, también hemos analizado las propuestas de análisis dinámico que se han aportado en la literatura. Este tipo de enfoque analiza la ejecución de las mismas, para determinar si el fin es malicioso o no.

Finalmente, nos hemos centrado en las seguridad en dispositivos móviles y, más concretamente, en la plataforma Android, ya que es la elegida para llevar a cabo la validación de esta investigación doctoral. En concreto, se han analizado las investigaciones más relevantes aparecidas en los últimos años en esta plataforma, desde el punto de vista de la seguridad y la privacidad.

La investigación en el área de la seguridad aplicada a tecnología móvil lleva, como hemos podido ver, un largo recorrido, especialmente en lo relativo a la detección de malware. Sin embargo, en los últimos años, desde la aparición de esta nueva generación de smartphones, este área de investigación ha sufrido un importante crecimiento. En la sección 4.3 podremos ver un extenso análisis de estas, centradas en el sistema operativo Android, que es el que se ha usado durante esta investigación doctoral.

En este capítulo hemos podido ver como la investigación en el campo de los teléfonos móviles inteligentes, desde el punto de vista de la seguridad, ha evolucionado en función de las características de los terminales. Si bien al principio se utilizaban enfoques similares a los que se daban en entornos de PC, hemos podido observar como estas investigaciones se han ido adecuando al entorno, analizando variables como la batería del terminal. Por ello, consideramos importante analizar esta evolución en la

seguridad de estos dispositivos para prever las áreas en las que centrar la investigación.

La seguridad en los sistemas operativos, así como el acceso a los recursos, ha sido ampliamente estudiada, y se encuentra en un estado muy estable. Estas metodologías de control, heredados de los sistemas operativos tradicionales, han sido exportados con éxito a los terminales móviles.

Sin embargo, la gestión de la privacidad es un ámbito que ha si que han producido investigaciones a lo largo de estos años, buscando encontrar un modelo lo suficientemente maduro y escalable para poder utilizar en todo tipo de dispositivos. Sin embargo, ese meta-modelo no ha llegado, desarrollándose una gran variedad de modelos en el camino.

En el ámbito de la detección de malware, las investigaciones han venido dadas por la escasa repercusión que este tipo de software ha tenido en dispositivos móviles. Sin embargo, cuando empezó a usarse el móvil para otras tareas, como por ejemplo para verificar al usuario de la cuenta bancaria, empezó a aparecer malware que buscaba controlar este entorno. El mercado de dispositivos móviles estaba muy fragmentado, y Symbian era el sistema operativo dominador del mercado, por lo que era el objetivo predilecto de los atacantes. Esto hizo que se los investigadores prestaran más atención a este tipo de dispositivos.

El primer enfoque fue el estático. Este tipo de enfoques son muy utilizados ya que, en general, son más eficientes a nivel de consumo de recursos, que es una de las principales limitaciones de este tipo dispositivos. Por ello, este enfoque es el utilizado en esta tesis doctoral. En este apartado se han presentado tanto las investigaciones realizadas en el ámbito de la movilidad como el que que se ha dado en otros entornos, estudiando su aplicabilidad a los *smartphones*.

Por contra, el análisis dinámico permite analizar comportamiento de los programas una vez ejecutados. De esta forma, se consigue una conocimiento más exhaustivo de la muestra, a costa de una mayor necesidad de recursos. Este tipo de análisis también permiten inspeccionar el uso de otros recursos del terminal, como por ejemplo el tráfico de red.

Por ello, este de técnicas se utilizan en una menor medida, y se han dedicado sobre todo al análisis de recursos del sistema, como puede ser por ejemplo el uso de batería por parte del terminal para detectar comportamientos anómalos.

En general, en este capítulo podemos observar que, si bien la evolución en los últimos años de este tipo de terminales han

supuesto una autentica revolución y sus características distan mucho de ser similares a las que tenían cuando se realizaron estas investigaciones, muchas de ellas pueden ser portadas a este nuevo escenario. También permiten ver la evolución que, en lo que respecta a este tipo de dispositivos móviles, han experimentado las investigaciones que se han desarrollado en los últimos años.

3 MODELADO DE AMENAZAS EN SMARTPHONES

«Esencialmente, todos los modelos son erróneos, pero algunos son útiles.»

George Box
(1919 –)

ÍNDICE

Antes de afrontar los problemas que se dan en el ámbito de la seguridad en los *smartphones*, es necesario conocer a fondo estos problemas. En este capítulo desarrollaremos un modelado de las amenazas en este tipo de dispositivos, ayudándonos así, a adquirir este conocimiento.

En primer lugar desarrollaremos este modelo, viendo las ventajas que aporta sobre otros modelos similares. Posteriormente, a fin de probar el modelo, detallaremos una serie de amenazas, vulnerabilidades y ataques que aplicaremos sobre él, a las que hemos denominado banco de ataques. En el desarrollo de este banco se han incluido todas aquellas de las que hemos tenido constancia a día de hoy, así como puntos vulnerables que podrían ser explotados en el futuro.

Finalmente, evaluaremos los resultados obtenidos, los cuales nos ofrecerán una visión clara de cuáles son los aspectos más débiles de la seguridad en este tipo de dispositivos a día de hoy.

3.1 MODELADO DE LA SEGURIDAD

El modelado se basa en 4 elementos claves: activos, amenazas, vulnerabilidades y ataques.

El primer paso es identificar todos aquellos elementos que entran en juego dentro del ámbito de la seguridad en los *smartphones*. Para ello hemos utilizado una categorización basada en el modelado de redes sociales [SLAB10, LSAGB10] desarrollado con anterioridad, ya que en este caso, se trata de un modelo que se ajusta a nuestras necesidades.

En este modelado de amenazas, hay 4 elementos claves sobre los cuales hemos desarrollado el modelo: los activos, las amenazas, los ataques y las vulnerabilidades. Es por ello que, la definición precisa y clara de cada uno de estos elementos es el punto de partida de este capítulo.

A continuación, pasamos a definir cada uno de estos elementos. En primer lugar definimos lo que son los activos que deben ser protegidos:

> *Definimos activos como el conjunto de todos los bienes y derechos con valor tal que son susceptibles de ser objetivo de amenazas, y cuya pérdida o deterioro puede suponer una merma en el valor de los mismos.*

Definimos amenaza de la siguiente forma:

> *Un evento que puede desencadenar un incidente, bien sean organizaciones (dispositivos pertenecientes a la empresa) usuarios particulares o elementos naturales, produciendo daños materiales o pérdidas inmateriales en sus activos.*

Por otro lado, definimos vulnerabilidad de la siguiente manera:

> *La posibilidad de que se materialice una amenaza sobre un activo.*

Finalmente, definimos ataque de la siguiente manera:

> *Es un evento, exitoso o no, que atenta sobre el buen funcionamiento del sistema que busca perjudicar al usuario o a alguno de sus activos, a través de la explotación de alguna vulnerabilidad.*

A modo de resumen, podemos decir que los ataques buscan perjudicar al usuario o a alguno de los activos, los cuales están comprometidos por distintas amenazas, a través del aprovechamiento de las vulnerabilidades existentes en el sistema.

El objetivo de este modelo es el de evaluar el riesgo de este tipo de dispositivos gracias al modelado de las amenazas que pueden sufrir, definiendo el riesgo de la siguiente manera:

> *La expectativa de pérdida expresada como la probabilidad de que una amenaza concreta explote una vulnerabilidad particular dando lugar un resultado perjudicial.*

Existe un sexto elemento, el cual no va a ser tratado en esta investigación, que son las contramedidas. Podemos definir contramedidas como *el sistema o las medidas diseñadas para prevenir que un ataque consiga su objetivo*. Si bien es cierto que es un punto vital desde el punto de vista de la seguridad, las contramedidas se pueden convertir en una lista de consejos y buenas prácticas, quedando lejos de los objetivos de esta investigación.

El modelado desarrollado deja fuera de ámbito de estudio las contramedidas.

Todos estos elementos, así como sus relaciones, han sido materializadas en el denominado «Círculo del Riesgo». Éste es una representación gráfica de las relaciones que se producen entre los elementos (ver en la figura 3.1).

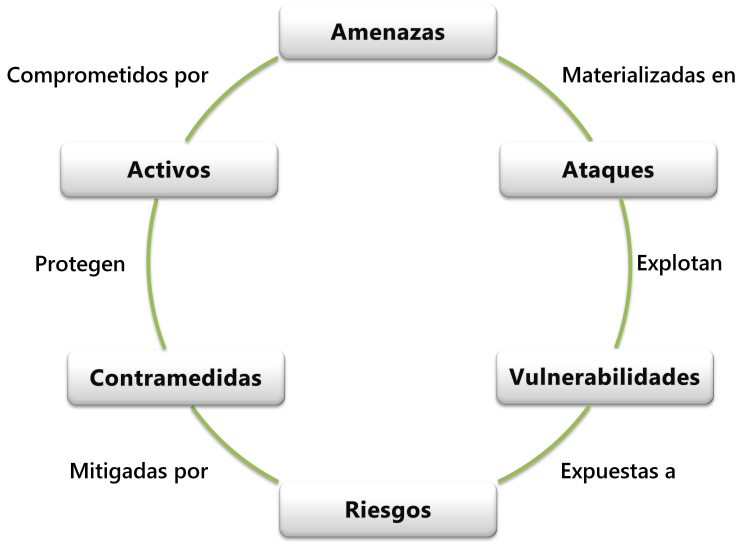


Figura 3.1: Círculo del Riesgo

A continuación veremos el modelo de amenazas desarrollado, fundamentado en los caminos de ataque [CBS07], permitiendo

categorizar las amenazas, y por extensión las vulnerabilidades y los ataques relacionados, en función de la importancia que se le otorgue a los activos que se desean proteger.

3.2 EL GRAFO DE ATAQUES

Basandonos en un trabajo previo, hemos adaptado el concepto de caminos de ataque a los smartphones.

Para realizar el modelado de las amenazas nos basamos en un trabajo previo realizado por Chen *et ál.* [CBS07]. En esta investigación, se define un método de modelado de amenazas cuantitativo que clasifica las amenazas de seguridad en base a la suma de los pesos de los caminos de ataque que son relevantes.

Basándonos en esa idea inicial de los caminos de ataque, hemos desarrollado un método que permite clasificar las amenazas en función de la importancia que se le dé a cada uno de los activos. Para ello, utilizaremos la propagación del peso de cada uno de los elementos por los distintos caminos existentes, siendo estos valores que se le otorga a cada uno de los activos, uno de los elementos a tener en cuenta a la hora de evaluar las amenazas.

A continuación, definiremos cómo es el camino, así como las partes que lo componen y la forma de calcular cada uno de los valores en cada una de las etapas.

Los pasos a seguir para generar un grafo de ataque son los siguientes:

1. *Definimos el banco de ataques.* Este banco está compuesto por todos aquellos accesos, ataques, vulnerabilidades, amenazas y activos identificados en el área en concreto, así como su relación entre los mismos. De esta manera, se define una parte de los valores que se utilizarán más adelante para calcular el valor de cada uno de los elementos dentro del modelado de amenazas.

2. *Generamos el grafo de ataque.* Una vez completado el banco de ataques, se genera el grafo correspondiente. Este grafo está compuesto por todos los elementos definidos en el banco de ataques con sus relaciones entre los mismos

3. *Propagamos los valores por el grafo.* Con el grafo completado, calculamos los valores de cada uno de los nodos del grafo. Posteriormente, evaluamos la importancia que tiene cada uno de ellos en la propagación de los mismos.

4. *Otorgamos el peso de cada uno de los valores*. Durante esta etapa se determina el peso que cada uno de los valores debe

tener en la valoración de cada uno de las amenazas. Para ello, el usuario introduce el peso que se le quiere otorgar a cada uno de los valores. Éstos son elementos relacionados con la seguridad (p. ej., privacidad o seguridad física) los cuales el usuario debe puntuar, dándole mayor importancia a los que considere más relevante. Están íntimamente relacionados con los activos y es, a través de estos valores, la forma en la que los usuarios dotan de valor a los nodos de los activos. De esta forma, se puede ajustar el resultado final del modelado de amenazas. Si por ejemplo, la privacidad es un elemento sobre el que queremos un especial énfasis, se le otorgará un mayor peso, consiguiendo de esta forma que las amenazas que más afectan a este valor sean los que tengan una posición más relevante dentro de la clasificación final.

El modo de otorgar y propagar el peso es uno de los pilares fundamentales del modelado.

Para calcular los valores de los atributos de varios de los elementos, hemos usado como base la guía de *Common Vulnerability Scoring System* (CVSS) [MSR07]. Ésta es un estándar de la industria para asignar importancia a las vulnerabilidades que se producen en los sistemas, estableciendo una métrica para compararlos. Por ello, se utilizará esta guía para evaluar el peso en varias de los nodos.

3.2.1 Definición formal del modelo

Los caminos de ataque se utilizan para determinar escenarios que dan lugar a situaciones que comprometen o dañan activos. Con ellos se pretende determinar cuáles son las vías por las que el atacante consigue el acceso a la víctima, qué vulnerabilidades se explotan para llevarlo a cabo y cuál es el impacto en los activos de dichos ataques. Para lograrlo, se trazan los distintos caminos que un atacante puede recorrer hasta llegar a su víctima. En cada paso, el atacante tiene múltiples opciones. Por ejemplo, el atacante que accede a los datos de un contacto en un smartphone, puede borrarlos o modificarlos. Cada uno de estos ataques genera un escenario distinto. Mediante esta metodología se busca generar un mapa con los posibles ataques, así como la relación existente entre ellos.

Los caminos de ataque se definen desde el atacante hasta la víctima, pasando por un nodo de cada clase.

En la figura 3.2 podemos ver de forma gráfica estos caminos. Se parte desde el nodo atacante, el cual busca acceder al sistema a través de algunos de los accesos que están disponibles. Una vez consigue el acceso, lleva a cabo el ataque, el cual explota

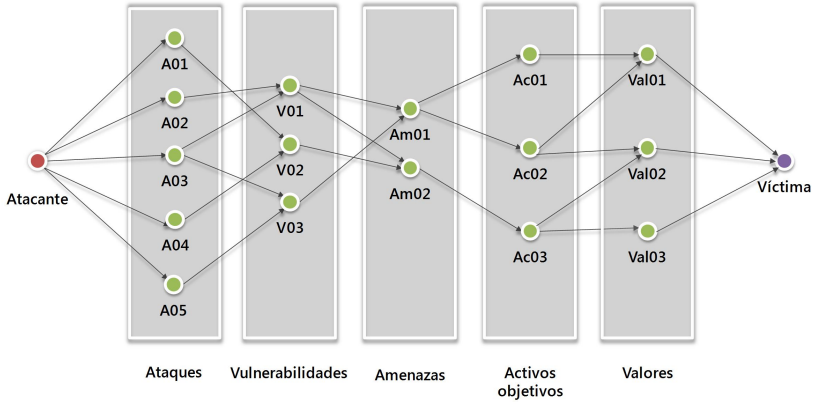


Figura 3.2: Grafo estructurado de ataque.

alguna vulnerabilidad existente. Estas vulnerabilidades son representadas por amenazas concretas a los distintos activos de la víctima del ataque.

El modelo explota la información proveniente del grafo generado.

Sin embargo, viendo el gráfico de ejemplo, podemos observar que no todos los nodos tienen el mismo peso. Así, por ejemplo, se puede ver que el ataque A03 explota dos vulnerabilidades distintas, mientras que el A02 solamente 1. Mediante este modelo, se explota esa información obtenida del grafo generado para clasificar las amenazas existentes.

De manera formal, definimos un grafo estructurado de ataque G como (ecuación 1),

$$G =< V_A, V_V, V_{Am}, V_{Ac}, E_{AcA}, E_{AV}, E_{VAm}, E_{AmAc} > \quad (1)$$

consiste en 5 conjuntos de vértices no vacíos, V_A, V_V, V_{Am}, V_{Ac} con una serie de conjuntos E_{AV},E_{VAm},E_{AmAc} y E_{AC} de pares ordenados de vértices distintos de G. V_A es el conjunto de nodos de ataques, V_V el conjunto de nodos de vulnerabilidades, V_{Am} el conjunto de nodos de amenazas y V_{Ac} el conjunto de nodos de activos. Por otro lado, $E_{AV} =< c, d >$ representa los pares ordenados de vértices donde $c \in V_A$ y $d \in V_v$; $E_{VAm} =< e, f >$ representa los pares ordenados de vértices donde $e \in V_v$ y $f \in V_{Am}$, y $E_{AmAc} =< g, h >$ donde $g \in V_{Am}$ y $h \in V_{Ac}$.

Por otro lado, dado un grafo de ataque G, un camino de ataque CA se define como (ecuación 2),

$$CA =< A, V, Am, Ac > \quad (2)$$

donde $A \in V_A$, $V \in V_V$, $Am \in V_{Am}$ y $Ac \in V_{Ac}$. De esta manera CA caracteriza un escenario de ataque.

Una vez caracterizado el camino de ataque, el siguiente paso sería definir el modelo de ponderación del grafo. Como se ha comentado en el punto 3.2, cada uno de los tipos de nodos de la red contienen un peso concreto. Este peso se otorga en base a dos criterios distintos. El primero de ellos es en base a los valores del atributo. La guía CVSS [MSR07] nos permite calcular los valores para distintos atributos a través de una metodología ampliamente utilizada en la industria.

Usamos CVSS como guía para dotar de peso a los nodos.

Para obtener los valores de esta guía se realizó de la siguiente metodología[SM09]. Un comité de expertos identificaron las métricas de entrada, compuestas por 6 métricas distintas con 729 posibilidades. Éstas se dividieron en las relacionadas con la explotación de la vulnerabilidad y con el impacto de la misma, perteneciendo 3 métricas a cada tipo. Con esta premisa inicial, y con la participación de expertos en la materia, se crearon las tablas de explotación y de impactos, cada una de ellas compuestas por 27 entradas. Posteriormente se combinaron ambas tablas otorgándoles un peso de un 40 % a la relacionadas con la explotación de la vulnerabilidad y un 60 % a las del impacto. Finalmente, se crearon las ecuaciones a fin de aproximar los resultados a las tablas obtenidas. Esta aproximación se realizó mediante la inclusión de pesos a los valores de las métricas y la desviación de las tablas de búsqueda para ajustar hacia arriba puntuación de distribución. Con todo ello, se conforma la guía con sus correspondientes valores.

Mediante la aplicación de formularios, se obtienen los valores de cada uno los atributos. Esto se explicará en el punto 3.2.2.

Por otro lado, aprovechando la naturaleza de grafo del camino, se explota la información del mismo grafo. Para ello, se utilizará una modificación del algoritmo de *PageRank* [PBMW99]. Este algoritmo permite clasificar los nodos de una red en función de la relación entre los mismos. En esencia, clasifica cada uno de los nodos de una red en función de su relevancia, la cual es calculada a partir del número de nodos que enlazan al nodo que se analiza, así como la relevancia de los mismos. La adaptación realizada a este algoritmo se explica en el punto 3.2.3.

Para explotar la información del grafo se utiliza el algoritmo PageRank.

3.2.2 Cálculo del peso de los nodos en función de sus atributos

En este apartado vamos a valorar cómo hemos realizado la ponderación de cada uno de los nodos. Dada la distinta naturaleza de cada uno de los nodos, se aplica una métrica diferente en función de sus características. Para determinar el valor de la mé-

trica, se ha usado como referencia la guía de CVSS[MSR07] para aquellos campos en la que la relación entre ambas métricas era directa. También se ha utilizado como referencia el trabajo realizado por Chen *et ál*. [CBS07], que se basa, a su vez, en la guía de CVSS para realizar el modelado de la amenazas.

Tambien se ha utilizado el trabajo de Chen et ál para el cálculo del peso de los nodos.

A continuación, detallaremos la descripción de los atributos de cada uno de los tipos de nodos, así como la manera de calcular el peso correspondiente de cada uno de ellos.

Ataque

El primer punto que evaluamos es el acceso al activo. Por otro lado, también medimos la complejidad del ataque, en función de las condiciones en las que esto se realiza. También se evalúa la reproducibilidad, es decir, la facilidad para volver a producir el ataque. Finalmente, se puntúa la ventana de ataque, es decir, el espacio de tiempo en el que el ataque puede producirse, que es necesaria para poder llevar a cabo el mismo.

A la hora de evaluar el acceso, existen dos elementos principales. El primero de ellos determina cuál es el vector de acceso del ataque, es decir, el medio por el que el atacante intenta entrar en el sistema. Por otro lado, se evalúa también la necesidad de autenticación o de validación por parte del usuario para llevar a cabo el ataque.

Para calcular el valor que se le da a esta primera característica, se aplica la siguiente fórmula:

$$\text{Acceso} = 20 \cdot VA \cdot Au \tag{3}$$

El acceso en el nodo de tipo ataque combina el vector para llevarlo a cabo y la autenticación.

donde VA representa el vector de acceso a activo y Au representa la autenticación necesario para acceder. Esta fórmula está extraída de la guía CVSS.

En primer lugar, nos centraremos en el vector de acceso necesario. Éste determina la forma en la que un atacante accede a los activos, es decir, el lugar donde debe estar el atacante para poder llevarlo a cabo. En este caso, lo hemos separado en tres posibles vectores: en local, en redes adyacentes, en la misma red. Los valores para evaluarlo son los siguientes:

- Local (L): el atacante requiere tener acceso físico a la máquina o a una cuenta local del sistema para poder realizar el ataque.

- Redes adyacentes (A): el atacante debe tener acceso a un dominio de *broadcast* o a un dominio de colisión para poder llevar a cabo el ataque. Ejemplos de estas redes son las redes IEEE 802.11 (WiFi) o las de IEEE 802.15 (Bluethoot).

- Red (N): el atacante no requiere el acceso local o encontrarse en la misma red que la víctima. Estos ataques también se conocen como «explotables remotamente».

A modo de resumen, podemos ver los valores que adquieren cada uno de estos elementos, así como una breve descripción de la misma en la tabla 3.1.

Valor métrica	*Puntuación*	*Descripción*
Local (L)	0,395	Requiere acceso físico al dispositivo.
Red Adyacente (A)	0,646	Explota redes adyacentes, como pueden ser WiFi, Bluethoot, etc.
Redes (N)	1	El atacante no necesita estar con acceso local a la red local.

Tabla 3.1: Evaluación de los marcadores de vector de acceso.

Por otro lado, evaluamos la necesidad de autenticación para llevar a cabo el ataque. Esta métrica mide el número de veces que un atacante debe autenticarse o pedir el permiso expreso de los usuarios para poder explotar la vulnerabilidad, y no la complejidad del sistema de autenticación utilizado. Cuantas menos veces se requiera la autenticación por parte del atacante, mayor será el valor que se le otorga a este atributo, debido a que el ataque tiene una menor complejidad. Es decir, aquellos ataques que no requieren de autenticación para ser llevados a cabo poseen un valor alto, ya que la facilidad de llevar a cabo el ataque es mayor.

La necesidad de autenticación es otro de los puntos que se evalúan.

Los posibles valores de este atributo son los siguientes:

- Múltiples (M): el atacante requiere autenticarse dos o más veces, aun con las mismas credenciales.

- Única (S): únicamente es necesario autenticarse una sóla vez para llevar cabo el ataque.
- Ninguna (N): no es necesario ningún tipo de autenticación por parte del atacante.

La puntuación que se concede a cada uno de los posibles valores de este atributo se pueden ver en la tabla 3.2.

Valor métrica	*Puntuación*	*Descripción*
Múltiples (M)	0,45	Requiere autenticarse dos o más veces.
Única (S)	0,56	Requiere autenticarse una única vez.
Ninguna (N)	0,704	No requiere autenticación.

Tabla 3.2: Evaluación de los marcadores de la necesidad de autenticación acceso.

La complejidad del ataque mide qué tan difícil es llevar a cabo el ataque.

A continuación definimos los valores del atributo que mide la complejidad del ataque. Este atributo determina, en función de las necesidades, la dificultad que tiene el atacante para llevar a cabo su acción.

Los valores que puede adquirir este atributo son los siguientes:

- Alto (H): este tipo de ataques requiere que se den unas condiciones especiales para poderse llevar a cabo. Por ejemplo, es necesario modificar otros sistemas o requiere privilegios elevados. También se aplica esta calificación cuando es necesario utilizar métodos de ingeniería social que son fácilmente detectados por gente con conocimiento en la materia.
- Medio (M): se requieren algunas condiciones especiales para poder llevar a cabo el ataque. Algunas situaciones dentro de este valor pueden ser que sea necesario obtener alguna información con anterioridad a poder realizar el ataque, la configuración del sistema no es la que se da por defecto o no es una configuración común.

- Bajo (L): todos aquellos ataques que no requieren ningún tipo de condiciones especiales se engloban dentro de este valor.

La puntuación que se concede a cada uno de los posibles valores de este atributo se pueden ver en la tabla 3.3.

Valor métrica	*Puntuación*	*Descripción*
Alto (H)	0,35	Requiere condiciones especiales para llevarse a cabo.
Medio (M)	0,61	Requiere alguna condición especial para poder ejecutarse.
Bajo (L)	0,71	No requiere de condiciones especiales.

Tabla 3.3: Evaluación de los marcadores de la complejidad del acceso.

La ventana de ataque mide la oportunidad de llevar a cabo el ataque.

También se mide la ventana de ataque. Esta ventana determina el tiempo en el que es posible realizar el ataque y, por consiguiente, la dificultad de llevarlo a cabo. Por ejemplo, únicamente es posible realizar el ataque cuando el *smartphone* se conecta al servidor de actualizaciones.

Los valores que puede adquirir este atributo son los siguientes.

- Alta (H): para poder llevar a cabo el ataque se tienen que dar unas serie de circunstancias que están disponibles durante de forma poco frecuente, como por ejemplo durante un proceso de actualización o a unas horas determinadas.
- Media (M): se requieren que se den algunas circunstancias para llevar a cabo el ataque, cuya frecuencia es bastante elevada.
- Baja (L): los ataques que requieran unas condiciones que se dan con mucha frecuencia o siempre entran dentro de esta categoría.

La puntuación de cada uno de los posibles valores vienen dados en la tabla 3.4.

Valor métrica	*Puntuación*	*Descripción*
Alta (H)	0,35	Requiere condiciones poco frecuentes.
Media (M)	0,61	Requiere condiciones que se dan con frecuencia.
Baja (L)	0,71	Requiere de condiciones que se dan habitualmente.

Tabla 3.4: Evaluación de los marcadores de la complejidad del acceso.

El peso del nodo de ataque combina tanto la complejidad como la ventana, así como el acceso.

El peso de un nodo de este tipo se calcula aplicando la siguiente fórmula:

$$N_A = Acceso \cdot Complejidad \cdot Ventana \tag{4}$$

Vulnerabilidad

Varios de los atributos se miden en este tipo de nodos. La evaluación de este tipo de nodos se basa en dos atributos distintos: el impacto y la popularidad de la vulnerabilidad.

El impacto está compuesto, a su vez, por la medición de tres elementos distintos. Esta medición se realiza de la misma manera que se determina en la guía de CVSS. La forma de evaluar este impacto se ha mantenido en la guía en las últimas versiones sin sufrir modificación, por lo que podemos afirmar que se trata de un método de evaluación consolidado. Para medir el impacto, se tiene en cuenta este factor en tres aspectos distintos: la confidencialidad, la integridad y la disponibilidad. La combinación de estos impactos se realiza de la siguiente manera:

$$Impacto = 10,41 \cdot (1 - (1 - IC) \cdot (1 - II) \cdot (1 - ID)) \tag{5}$$

Se mide el impacto de la confidencialidad, de la integridad y de la disponibilidad.

siendo IC el impacto que tiene en la confidencialidad, II el impacto que tiene en la integridad e ID el impacto que tiene el ataque en la disponibilidad. Veamos cómo se miden cada uno de los ellos.

El primero, el impacto de la confidencialidad (IC), hace referencia al acceso a la información limitada únicamente a los

usuarios autorizados, así cómo la prevención del acceso a esa información por parte de los usuarios que no están autorizados.

Los valores que puede adquirir este atributo son los siguientes:

- Ninguno (N): no tiene impacto en la confidencialidad del sistema.
- Parcial (P): tiene un impacto considerable sobre la información del sistema. Es decir, los atacantes pueden acceder a una parte de la información pero no pueden controlar qué parte obtener o está limitado el alcance de la pérdida.
- Completa (C): existe una divulgación completa de la información. El atacante tiene acceso a toda la información.

A modo de resumen, podemos ver los valores en la tabla 3.5.

Valor métrica	*Puntuación*	*Descripción*
Ninguno (N)	0	No tiene impacto en la confidencialidad.
Parcial (P)	0,275	El atacante tiene un acceso limitado a la información.
Completa (C)	0,66	El atacante tiene un acceso completo a la información.

Tabla 3.5: Evaluación de los marcadores del impacto sobre la confidencialidad.

Por otro lado, II mide el impacto sobre la integridad de la información tras producirse el evento. Puede adquirir tres valores:

- Ninguno (N): no tiene impacto sobre la integridad.
- Parcial (P): tiene un impacto considerable sobre la integridad del mismo.
- Completa (C): existe la posibilidad de modificación completa de la información.

Los tres impactos distintos pueden adquirir los mismos valores, con el mismo peso.

A modo de resumen, podemos ver los valores en la tabla 3.6.

Valor métrica	*Puntuación*	*Descripción*
Ninguno (N)	0	No tiene impacto en la integridad de la información.
Parcial (P)	0,275	El atacante tiene un acceso limitado sobre la integridad. de la información.
Completa (C)	0,66	El atacante tiene el control sobre la integridad.

Tabla 3.6: Evaluación de los marcadores del impacto sobre la integridad.

Finalmente, el atributo que mide el impacto de la disponibilidad, es decir, a la capacidad de acceso a la información durante el ataque o con posterioridad a él. Los posibles valores que puede adquirir este atributo son los siguientes:

- Ninguno (N): no tiene impacto en la disponibilidad del sistema.
- Parcial (P): tiene un impacto considerable sobre la disponibilidad del sistema. Se reduce el rendimiento del sistema, o se producen interrupciones en su disponibilidad.
- Completa (C): el recurso deja de estar disponible durante el ataque o con posterioridad a él.

Se pueden ver los valores que adquieren estos atributos, así como los pesos que se le asigna en la tabla 3.7.

La popularidad se mide en base a la frecuencia en la que se explota la vulnerabilidad.

Por otro lado, medimos la popularidad de la vulnerabilidad que ha sido explotada.

Los posibles valores que adquiere este item son los siguientes:

- Frecuentemente explotada (F): se trata de una vulnerabilidad conocida que se explota con frecuencia.

Valor métrica	*Puntuación*	*Descripción*
Ninguno (N)	0	No tiene impacto en la disponibilidad del recurso.
Parcial (P)	0,275	Existe una merma en la disponibilidad del recurso.
Completa (C)	0,66	El atacante tiene un acceso completo a la información.

Tabla 3.7: Evaluación de los marcadores del impacto sobre la disponibilidad.

- Explotada, pero no frecuentemente (E): se trata de una vulnerabilidad conocida pero, por la complejidad técnica o por otro tipo de limitaciones, no tiende a explotarse con frecuencia. Este tipo de vulnerabilidades tienden a ser más difíciles de controlar, motivo por el cual tienen una puntuación más alta dentro que las que son más frecuentemente explotadas.
- No explotada(N): No se conocen casos en los que haya sido explotada.

La puntuación de estos valores puede verse en la tabla 3.8.

Valor métrica	*Puntuación*	*Descripción*
Frecuentemente explotada (F)	0,35	Vulnerabilidad conocida y explotada de forma habitual.
Explotada, pero no frecuentemente (E)	0,61	No es explotada con frecuencia.
No explotada(N)	0,71	No se tiene constancia de que haya sido explotada.

Tabla 3.8: Evaluación de los marcadores sobre la popularidad de la vulnerabilidad.

El peso de un nodo de este tipo se calcula aplicando la siguiente fórmula:

$$N_V = Impacto \cdot Ventana \tag{6}$$

Amenaza

El peso de la amenaza es, en este caso, el valor a calcular. Todas aquellas amenazas identificadas serán ordenadas en función del valor que se obtenga del cálculo de este modelado.

Activo

Los activos adquieren su peso de la evaluación de un experto y de su PageRank.

El valor del activo viene dado principalmente, por el peso que el usuario ha otorgado a cada uno de los valores con los que están relacionados. Por otra parte, el peso del nodo también viene determinado por el valor que el grafo resultante le otorgue a través del algoritmo de *PageRank* [PBMW99].

Valor

El valor es el elemento que aquel que está utilizando el modelado define como importante a la ahora de clasificar las amenazas. El rango de valores posibles para estos valores es $1 \leqslant \alpha \leqslant 10$. El peso de este valor es el que se propaga sobre los activos para, de esta forma, calcular cuáles son los activos con mayor importancia para ponderar las amenazas que entrañan un mayor riesgo para ellos.

Los valores que pueden adquirir son: privacidad, seguridad financiera y física, disponibilidad y reputación.

La lista de valores a considerar son los siguientes:

- *Privacidad* (Va01): es uno de los elementos que aparecen referenciados de forma constante en el banco de ataques, descrito en la sección 3.3. Es uno de los valores que más se está poniendo en entredicho con el auge de los *smartphones* y su combinación con las redes sociales.
- *Seguridad financiera* (Va02): los smartpones cada día están más involucrados en operaciones financieras, bien sea por el uso de tarjetas bancarias a través de distintos servicios Web o por el pago mediante el mismo terminal gracias a las características monedero electrónico que pueden incorporar.
- *Seguridad física* (Va03): las posibilidades de localización a través de los sensores que posee el terminal hacen que la seguridad en el mundo virtual tenga repercusión en el mundo real, pudiendo dar lugar a situaciones comprometidas.

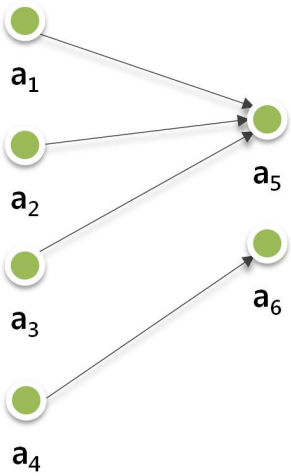


Figura 3.3: Grafo de ejemplo.

- *Disponibilidad* (Va04): el uso del terminal móvil en un amplio conjunto de actividades de la vida cotidiana hace que su disponibilidad sea un valor importante.

- *Reputación* (Va05): como integrador de muchos formas de comunicación de los usuarios (por ejemplo correo electrónico, mensajería instantanea o SMS), un ataque sobre el mismo puede hacer que la reputación del usuario se vea mellada.

3.2.3 Propagación de los pesos

Explotamos la información del grafo a través del algoritmo PageRank.

La propagación de los pesos se realiza mediante el algoritmo de PageRank [PBMW99]. Supongamos que tenemos un grafo como el que se ve en la figura 3.3, en el cual se representa el conjunto de vértices $V = a_i \quad \forall 1 \leqslant i \leqslant 6$ y $E = (a_i, a_j) \quad \forall 1 \leqslant i \leqslant 6 \wedge i \neq j$, siendo A_i el conjunto de nodos a los que apunta el nodo $i \in V$.

De manera formal, consideremos un conjunto de nodos A_n, formado por los nodos a los que apunta el nodo n y B_n, formado por el conjunto de nodos que apuntan al nodo n. Siendo $N_n = |A_n|$, el número total de vértices que apunta el nodo n, definimos el ranking del nodo como (ecuación 7),

$$PR(n) = (1 - d) + d \cdot \sum_{m \in B_n} \left(\frac{PR(m)}{N_n(m)} \right) \cdot P(m) \tag{7}$$

siendo $P(m)$ el peso del nodo calculado en base a los atributos que contiene, el cual se calcula con la metodología expresada en el punto 3.2.2. La variable d determina un valor de amortiguación, configurado con un valor de 0,85. Este valor se introduce para que aquellos nodos que no tengan enlaces no salgan beneficiados. Para ello, resta algo de peso a todos los nodos. Mediante este algoritmo, se potencian aquellos nodos que tienen un gran número de nodos de entrada.

3.2.4 Cálculo de la métrica

Calculamos el valor del camino de ataque en base al peso de cada uno de los nodos, ponderados por la importancia de cada etapa.

A continuación vamos a detallar el cálculo de la métrica que utilizaremos para valorar cada uno de las amenazas a las que se enfrenta el sistema.

Para ello, tomamos como referencia el cálculo base que se realiza en la medición de las vulnerabilidades dentro del sistema CVSS [MSR07], para posteriormente, aplicarle a esta métrica los valores de los nodos que hemos calculado.

Así, en caso de querer evaluar a la amenaza en un camino concreto, es posible hacerlo aplicando la ecuación 8

$$\begin{aligned} |Nam_i| &= (0,3 \cdot Na_i + 0,15 \cdot Nv_i - 1,5) \cdot f(Impacto) + \\ &\quad (0,05 \cdot Nac_i + 0,15 \cdot Nva_i) \end{aligned} \tag{8}$$

donde cada uno de los correspondientes nodos es el valor obtenido de multiplicar el valor del nodo por el peso que tiene en la red, esto es, el valor de su PageRank.

Para categorizar los nodos de amenazas, combinamos los pesos de los nodos con los que están relacionados.

Esta fórmula nos permite hacer un cálculo del peso de cada uno de los elementos que conforman el camino de ataque.

Sin embargo, para determinar el peso de las amenazas en global, hemos optado por otro enfoque. El enfoque de los caminos de ataque tiene una serie de limitaciones. La más importante de ellas, es que únicamente puede haber un nodo de cada clase. Sin embargo, puede darse que haya situaciones en las que un ataque explote más de una vulnerabilidad. Por ello, hemos optado por el uso de todo el grafo para modelar las amenazas.

Formalmente, definimos el grafo dirigido $G = (N_i, E_{ij})$, donde N_i es el conjunto de nodos, que puede ser de cualquiera de las clases que se ha explicado anteriormente y $E_{i,j}$ es el conjunto de vértices que conecta los nodos n_i y n_j, teniendo en cuenta que, al ser un grafo dirigido existe la restricción de $(n_i, n_j) \neq (n_j, n_i)$.

A fin de hacer la clasificación, propagamos el peso de los valores introducidos por el usuario hacia los nodos de activos para, posteriormente, calcular el peso de cada una de las amenazas en función del valor que tienen todos aquellos nodos con los que están conectadas.

Formalmente, primero propagamos el valor del nodo Va hacia los nodos Ac que están conectados, de la siguiente manera:

$$|NAc| = |Nac| + NVa \quad \forall\, Nva \mid Nva \in E < NAc, Nva > \quad (9)$$

Posteriormente, calculamos el valor del nodo Am en función de los nodos a los que esta conectado, de la siguiente manera:

$$|NAm| = \sum N_i \quad \forall\, N_i \mid N_i \in E < NAm, N_i > \quad (10)$$

Con ello, obtenemos el valor de cada una de las amenazas dentro del grafo. Este valor nos permite clasificar las amenazas de mayor a menor importancia en función de la importancia que el analista conceda a cada uno de los valores.

3.3 BIBLIOTECA DE ATAQUES

A continuación detallaremos la colección de ataques, amenazas y vulnerabilidades que hemos identificado en el entorno de los *smartphones*.

Para probar el modelo generado, hemos desarrollado la primera biblioteca de ataques sobre smartphones.

Esta colección ha sido desarrollada tras un profundo estudio de los distintos eventos e investigaciones que se han dado en los últimos tiempos para este tipo de dispositivos. En este apartado se identifican todos los activos que deben protegerse, así como las amenazas a las que se enfrentan, y las vulnerabilidades encontradas.

Todos estos elementos están agrupados para facilitar su comprensión. Sin embargo, dentro del camino de ataque cada uno de ellos corresponde a un único nodo de la red.

Esta biblioteca de ataques es, hasta donde llega nuestro conocimiento, el primer estudio que se ha realizado sobre la materia en el entorno de los *smartphones*.

A continuación empezamos detallando los activos identificados. Posteriormente, clasificaremos las amenazas en tres categorías distintas de amenazas relacionadas con: (i) las aplicaciones, (ii) las redes de comunicaciones y (iii) aspectos físicos.

Al lado de cada uno de ellos aparece un código que identifica de forma unívoca a cada uno de los elementos. En concreto, los activos son identificados con el prefijo *Ac*, las amenazas se identifican con el prefijo *Am*, las vulnerabilidades con el prefijo *V* y los ataques con *A*.

3.3.1 Activos identificados

El primer paso es identificar los activos que deben ser protegidos.

Los activos identificados los hemos agrupados en tres categorías distintas. Por un lado, tenemos los datos que se almacenan en el terminal. Estos datos son de distinta naturaleza (p. ej. multimedia o texto). Por otro lado tenemos el propio software que se ejecuta en los terminales. Finalmente, el hardware que conforma el terminal, con los sensores instalados para las distintas tareas, es uno de los activos más importantes con los que cuenta el smartphone. A continuación, detallamos cada uno de estos activos.

Datos (Ac01)

El activo principal que almacena este tipo de dispositivos son los datos. En un terminal se almacena una gran variedad de información de distinta naturaleza y con distinto grado de importancia, desde el punto de vista de la seguridad.

A continuación se detallan algunos de estos tipos clasificados según su naturaleza:

- *Datos de organización personal*, como por ejemplo el calendario del usuario, listas de tareas, o listas de proyectos.
- *Agenda de contactos de teléfono.*
- *Información textual*, como la información contenida en SMS, en emails, o en aplicaciones de productividad como Evernote.
- *Material Multimedia*, algunos ejemplos son:
  - Música
  - Fotografías
  - Vídeos
  - Partidas guardadas de videojuegos
  - Creaciones artísticas

- *Historial de navegación*, donde se guardan las páginas web que ha visitado el usuario.
- *Datos del GPS*, de donde se pueden obtener los últimos lugares visitados, lugares en los que ha estado, entre otros.
- *Información de redes sociales*. Las redes sociales almacenan una gran cantidad de información de los usuarios, y los teléfonos se han convertido en un elemento imprescindible en su crecimiento, gracias a la inmediatez que ofrece a la hora de actualizar el estado de los usuarios.
- *Información que almacena el operador móvil*, tanto de los datos del mismos (modelo, IMEI, por ejemplo) como de patrones de uso (p. ej. horas a las que se realizan llamadas o conexión a tarifas de datos). Un ejemplo del valor de esta información es el caso de CarrierIQ[1] en Estados Unidos. Los operadores también son capaces de conocer otro tipo de información, como los lugares que has visitado, las llamadas.
- *Credenciales almacenadas en el dispositivo* (p. ej. contraseñas de los servicios). La forma de almacenar las credenciales suele depender de cada una de las aplicaciones, que, en muchas ocasiones, no tienen en cuenta aspectos de seguridad.

Los datos son uno de los activos con más valor dentro del terminal.

Software (Ac02)

El software es uno de los elementos vitales del terminal, ya que dominan y gestiona todo el hardware. Sin embargo, su complejidad y su sofisticación añaden distintos problemas a la seguridad del conjunto.

La complejidad del software presente en los terminales provoca que los problemas también sean más complejos.

- *Vulnerabilidades del sistema operativo instalado*: la complejidad de los sistemas operativos en este tipo de dispositivos ha crecido enormemente, de forma paralela a como han crecido sus capacidades. Como consecuencia de esta complejidad, el numero y la gravedad de las vulnerabilidades presentes en estos sistemas operativos han experimentado un importante crecimiento.
- *ROMs personalizadas*: las ROMs son versiones modificadas del sistema operativo que el usuario puede instalar en su

1 http://androidsecuritytest.com/features/logs-and-services/loggers/carrieriq/

terminal. Se dan en sobre todo en sistemas operativo abiertos, como es Android. Existe una gran variedad de ellas, cada una de las cuales nace con un objetivo distinto, como por ejemplo mejorar el rendimiento, hacer cambios en la interfaz gráfica o fortificar aspectos relacionados con la seguridad. Cabe diferenciar entre dos tipos de ROMs, las que están basadas en AOSP (*Android Open Source Project*, el código fuente ofrecido por Google) y las demás. Existen algunos ejemplos de la primera, por ejemplo Blandroid[2] o CyanogenMOD[3], mientras que otras pretenden dotar de una funcionalidad especial a la ROM, como por ejemplo Taintdroid [EGC+10b], que añade un sistema de monitorización de la privacidad en el dispositivo.

- *Programas instalados*: la libertad que aporta Android hace que el número de fuentes de las que se puede instalar una aplicación sea diverso. Además de la tienda oficial de Android, el Android Market, existen otras tiendas con mayor o menor éxito, como por ejemplo la de Amazon (Amazon Store) o GetJar. Por otro lado, también existe la posibilidad de instalar una aplicación descargada directamente de Internet sin necesidad de pasar por la tienda de aplicaciones. Esta última vía es usada para aportar versiones con las protecciones de las aplicaciones saltadas, algunas de ellas modificadas para albergar *malware*. En el caso de Apple, se han dado casos de aplicaciones con comportamiento malicioso subidas a la tienda. Bien es cierto que estos casos han sido expuestos como prueba de concepto y no se han notificado casos de *malware* que se hayan subido a su tienda.

La falta de control a al hora de instalar aplicaciones en los terminales pueden provocar problemas de seguridad.

Hardware (Ac03)

Los componentes hardware de los dispositivos les ofrecen buena parte de su potencia. Sin embargo, también ofrecen nuevas vías de ataque a los mismos. Algunos de estos componentes susceptibles de ser utilizados en un ataque son:

- *WiFi*: este tipo de hardware, mediante ataques de «Man-in-the-middle» [Sta03], comunes en este medio, puede permitir al atacante interceptar las comunicación del terminal.
- *Batería*: se han documentado ataques a la batería de los terminales [NMHH05, BNC+08, JD04], provocando que no

2 http://blandroid.org/
3 http://www.cyanogenmod.com/

se puede utilizar el terminal. A este tipo de ataques se les conoce como ataques de denegación de servicio.

Elementos del hardware como la conexión WiFi o el la cámara de fotos puede ser utilizada por los atacantes para obtener información crítica del usuario.

- *Procesador*: el «cerebro» del terminal es uno de los activos más valiosos. Si bien es cierto que de momento no se han registrado una gran cantidad de ataques al procesador, nos encontramos ante uno de los elementos más valiosos de los dispositivos. El tipo de *malware* que empieza a explotar este activo son las *botnets* en *smartphones* [XBL+11], las cuales esta proliferando en lo últimos años [BSGBdV11].

- *Interfaz táctil*: la captura de los eventos que se producen en la interfaz táctil de este tipo de dispositivos pueden llevar a la realización de ataques[4]. Existen varias pruebas de concepto de *malware* que, a través de la explotación de la pantalla táctil y de los eventos que genera, pueden causar perjuicio al usuario.

- *Puerto USB*: en general, cualquier interfaz de entrada es susceptible de ser utilizada por un atacante para obtener información del terminal.

- *GPS*: este hardware ofrece datos de la posición en la que se encuentra el terminal, posibilitando al atacante conocer la ubicación física del mismo.

- *NFC*: esta tecnología permite el intercambio de información entre dos dispositivos que se encuentran a muy poca distancia. Ya se han iniciado investigaciones para analizar la seguridad de esta tecnología [HB06], ya que empiezan a aparecer sistemas de pago a través de esta tecnología.

- *Cámara de fotos*: la cámara de fotos ha sido uno de los elementos hardware que más ha evolucionado dentro del terminal móvil. Sin embargo, este hardware permite que un atacante tenga una cámara en directo desde el bolsillo de la víctima.

- *Micrófono*: un atacante con acceso al micrófono es capaz de grabar todas las conversaciones y los sonidos de la víctima, por lo que puede extraer información muy valiosa.

- Conexión GSM y GPRS (teléfono y datos): de la misma forma que las redes WiFi, el ataque sobre este tipo de hard-

4 http://blog.mylookout.com/blog/2010/12/09/android-touch-event-hijacking/

ware permite obtener los datos y las conversaciones de las victimas.

3.3.2 Clasificación de las amenazas

Identificamos las amenazas, las vulnerabilidades y los ataques.

En este apartado se detallan todas aquellas amenazas identificadas en el transcurso de este trabajo doctoral. Estas amenazas han sido clasificadas en 3 categorías distintas. En primer lugar, se agrupan todas aquellas amenazas vinculadas a las aplicaciones del terminal. Por otro lado, todas aquellas amenazas relacionadas con las comunicaciones de los dispositivos tienen su clasificación correspondiente. Estas amenazas engloban todas aquellas acciones que requieren el uso de alguno de los canales de comunicación Finalmente, todas aquellas amenazas basadas en distintos aspectos físicos que involucran el uso de estos terminales tienen su reflejo también en esta clasificación.

Amenazas relacionadas con las aplicaciones

Las aplicaciones de los *smartphones* son uno de los puntos que dotan al terminal de mayor versatilidad. Sin embargo, esta versatilidad también es fuente de múltiples vulnerabilidades. A continuación se detallan las vulnerabilidades localizadas relacionadas con este aspecto.

- *Instalación de aplicaciones* (Am01): la instalación de las aplicaciones se ha demostrado[5] como uno de los principales problemas desde el punto de vista de la seguridad se puede ver comprometida. Hay que tener en cuenta la fuente de la que se obtiene la aplicación, la funcionalidad que tiene, entre otros diversos factores.

  La falta de conocimiento de los usuarios a la hora de instalar aplicaciones puede provocar situaciones de riesgo para la integridad del terminal.

  - Control de las aplicaciones subidas a las tiendas (V01): la falta de control y análisis en las aplicaciones subidas a la tienda de aplicaciones. Algunas de ellas realizan un exhaustivo control sobre las aplicaciones que se suben antes de publicarlas. Sin embargo, otras tiendas no realizan ese control, permitiendo subir aplicaciones maliciosas.
  - Falta de conocimiento de los usuarios a la hora de instalar aplicaciones (V02): en general, los usuarios no

5 http://www.zdnet.com/blog/security/malware-charges-users-for-free-android-apps-on-google-play/12245

son conscientes de las distintas partes de sus datos o del hardware a las que accede una aplicación. En particular, en Android, la pereza o la falta de conocimiento sobre las consecuencias que tiene la instalación de algunas aplicaciones hace que, en muchas ocasiones, el usuario no lee los permisos que solicita la aplicación a la hora de instalarse.

El malware es uno de los mayores problemas de seguridad a los que se enfrenta la plataforma Android.

  * Aplicaciones Maliciosas (A01): han aparecido multitud de muestras de software con intenciones maliciosas, especialmente en el sistema operativo Android [ZJ12a].
  * Robo de información del dispositivo (A02): es posible, a través de las distintas aplicaciones del terminal, que éstas sustraigan información del propio dispositivo y sea enviada a los atacantes.
  * Ataques de denegación de servicio (A03): existen aplicaciones que provocan denegación de servicio, bien sea a través de un consumo excesivo de alguno de ellos, bien estar programada de forma defectuosa, bien porque se ha programado con ese fin.
  * Botnets para móviles (A04): existen algunas redes de *smartphones* zombis, también conocidas como botnets [BSBV11], que han aparecido para diversas plataformas, como iOS [PSY10] o Android [XBL+11].

- *Transmisión de los datos sin conocimiento del usuario* (Am02)
  - Escaso conocimiento de las tecnologías por parte del usuario (V03): en general, hay usuarios que no son conscientes de las capacidades de sus terminales y, por consiguiente, de los peligros que entrañan.

La tecnología NFC cada día está mas extendida entre los smartphones.

    * Ataques a comunicaciones por NFC (A05): la tecnología NFC permite, mediante el contacto de dos dispositivos, el intercambio de datos entre ellos. Con el nuevo método presentado por Android, es posible mandar URLs, aplicaciones, o contactos a otro teléfono sin el consentimiento expreso del usuario receptor.
    * Ataques a comunicaciones por Bluetooth (A06): esta tecnología, que está disponible desde hace

varios años en los terminales móviles, ha sido explotada como medio para la obtención de datos de otros terminales. Un ejemplo de ello es BlueFTP, que permite acceder a los datos de otro teléfono e incluso realizar llamadas.

  * Aplicaciones Maliciosas (A01): las aplicaciones maliciosas pueden enviar los datos del usuario al atacante, a través de los distintos medios por los que puede enviar información una aplicación.

Los sistemas abiertos, como Android, permiten a los usuarios saltarse medidas de seguridad implantadas desde el diseño.

- *Modificación de los datos del teléfono* (Am03): el teléfono es un elemento en constante cambio, especialmente los datos que contiene. A continuación detallamos algunas vulnerabilidades y ataques relacionadas con esta amenaza
  - Modificación o sustitución del sistema operativo del smartphone (V04): a través de Internet existen distintos procedimientos que permiten a los usuarios modificar o sustituir el sistema operativo del terminal, bien sea instalando ROMs, instalando aplicaciones o modificando las existentes; lo que lleva a exponer ciertas parte del mismo.
    * Recolección de datos de log (A07): obtención de datos a través del log del smartphone o a través de aplicaciones de terceros que hacen la recolección de los mismos. Un ejemplo de ello es el software instalado por parte de distintos operadores móviles, conocido como CarrierIQ[6].
    * Instalación de software malicioso (A01): las ROMs personalizadas (es decir, las versiones del sistema operativo creadas por otros usuarios) pueden tener aplicaciones instaladas cuyo comportamiento sea malicioso.
    * Montaje de la partición del sistema desde un comando exterior con permisos de escritura (A08): las ROMs pueden permitir el montar la partición del sistema con permisos de escritura, lo que permitiría al atacante modificar ficheros del sistema.
    * Utilización de herramientas de conexión con el terminal en modo superusuario (A09): en las ROMs

6 http://androidsecuritytest.com/features/logs-and-services/loggers/carrieriq/

personalizadas es posible acceder al sistema con el usuario con mayores privilegios, también conocido como modo administrador o superusuario a través de las herramientas que proveen para desarrollo, como puede ser el puente de conexión con Android (de la locución inglesa *Android Bridge Connection* (ADB)). Existen herramientas que facilitan la conectividad entre un PC y el smartphone. Este tipo de herramientas han sido creadas principalmente para tareas de desarrollo. En Android, por ejemplo, en un terminal modificado, esta herramienta permite ejecutar aplicaciones con permisos de superusuario, lo que haría que un atacante sea capaz de ejecutar aplicaciones con el máximo nivel de privilegios.

Las herramientas de desarrollo mal configuradas también pueden ocasionar situaciones de riesgo.

  - Herramienta de conexión (ADB) sobre WiFi (A10): una mala configuración del la ROM permite acceder al sistema de análisis del sistema en fase de desarrollo sin necesidad de estar conectado físicamente con el dispositivo, sino que únicamente es necesario estar conectado a la misma red inalámbrica.
  - Acceso a la configuración y al sistema como superusuario (A11): En el caso de Android, esto evade uno de los principios de diseño sobre los que se basó su sistema de seguridad, que es el de mínimos privilegios posibles [SKFT09].

- Acceso de datos por parte de las aplicaciones (Am04)
  - URL maliciosa a través del navegador (A12): muchos atacantes están infectado páginas web con el objetivo de perjudicar a los usuarios que se conectan a las mismas. En algunas ocasiones, estos ataques son consentidos por el usuario. Un buen ejemplo de esto es la página web jailbreakme[7]. Esta web explota vulnerabilidades de las distintas versiones del iPhone para poder liberar el teléfono y que el usuario pueda utilizar tiendas alternativas o incluso tarjetas SIM de otras operadoras.
  - Distribución de malware a través de códigos QR (A13). Los códigos QR o BIDI [CLP+11] son códigos de ba-

7 http://www.jailbreakme.com

rra bidimensionales que permiten cargar contenidos al usuario. Ya han aparecido distintos ataques a través de este medio[8].

– Simulación de interfaz de usuario para obtener los datos (A14): algunas páginas web simulan la interfaz del navegador para capturar toda aquella acción que realiza el usuario. A través de este método también es posible modificar las páginas que visita el usuario, pudiendo capturar información privada, como puden ser unas credenciales de acceso.

Amenazas relacionadas con las redes de comunicaciones

- *Conexión a WiFi no confiable* (Amo5). La proliferación de puntos de acceso WiFi abiertos en lugares públicos ha servido para mejorar la conectividad, pero también supone un grave problema desde el punto de vista de la seguridad.

Los ataques en medios abiertos, como las redes WiFi, suponen un problema para la privacidad de las comunicaciones.

  – Redes WiFi sin medidas de seguridad apropiadas (Vo5):
    * Cifrado de la conexión débil o inexistente (Vo6): muchas de estas redes no utilizan ningún tipo de cifrado. Al tratarse de un medio abierto, cualquier atacante es capaz de conectarse a él y vigilar todo aquello que los usuarios conectados a la red envíen o reciban.
    * Alcance de la red inalámbrica superior al necesario (Vo7): el alcance de las redes inalámbricas es, en muchas ocasiones, superior al necesario, lo que puede hacer que un atacante desde la calle pueda tener acceso a las redes inalámbricas.
      · Ataques sobre la capa física (A15): en esta capa es donde se pueden realizar ataques de denegación de servicio a la red. Estos, conocidos como ataques de *jamming* [XMTZo6], consisten en saturar el medio a través de una señal de ruido más potente que la emisión original.
      · Ataques sobre la capa MAC (A16): existen varios ataques conocidos como mal comportamiento sobre MAC, que engloban distintas acciones perniciosas para el buen funcionamiento de la red [GKFo2].

8 http://www.securelist.com/en/blog/208193145/Malicious_QR_Codes_Pushing_Android_Malware

        - Ataques «Man in the Middle» (A17): es un ataque en el que el enemigo adquiere la capacidad de leer, insertar y modificar a voluntad, los mensajes entre dos partes sin que ninguna de ellas conozca que el enlace entre ellos ha sido violado [Sta03, ANN05]. En los terminales móviles, este ataque se puede realizar por todas las vías de comunicación que utilice. El medio más común para realizar este tipo de ataques es sobre las redes inalámbricas.

- Amenazas relacionadas con la infraestructura de red telefónica (Am06): existen una serie de amenazas inherentes a la utilización de la red telefónica, que pueden ser explotadas por los atacantes.

  La infraestructura telefónica también es vulnerable a ataques.

  - Infraestructura de red insegura (V08): la infraestructura de la red de comunicación ha sido objeto de distintos ataques. Éstos buscan obtener ventajas atacando el canal de comunicación.
    - Criptografía de la red GSM (Am07): la red de comunicación GSM (*Global System for Mobile Communications*) [Hei99] es un estándar europeo de comunicación, el más utilizado a día de hoy a nivel global.
      - Ataque al sistema A5 (A18): el sistema de cifrado A5, que es el que utiliza la red GSM, se ha mostrado vulnerable, y ya ha sido quebrado [DKS10].
    - Fragilidad de la infraestructura de mensajes cortos (Am08): la red SMS nació con el objetivo de enviar órdenes de control a los terminales móviles. Sin embargo, su éxito llegó cuando se utilizó para el envío de mensajes cortos entre los usuarios. Posteriormente, evolucionó a mensajes multimedia, ampliando su capacidad.
      - Ataques al sistema SMS (A19): la infraestructura del SMS es vulnerable y puede causar ataques de denegación de servicio en toda la infraestructura de toda la red [CO07, ETMLP05].
    - Debilidades en los sistemas de *backend* (Am09)
      - Ataques contra el sistema *Home Location Register* (HLR) (A20): Traynor et al. [TMP08] han

demostrado que el punto débil de la infraestructura de la red móvil es el HLR. Este elemento es una base de datos central que contiene los detalles de cada usuario de telefonía móvil autorizado a usar la red GSM.

· Ataque a la infraestructura de GPRS (A21): la infraestructura básica que soporta la comunicación de datos ha sido puesta a prueba en varias ocasiones y, a día de hoy, se le puede calificar como insegura [RLPS+oo].

· Ataques a la infraestructura de MMS (A22): los mensajes multimedia siguen otro canal distinto al de los SMS tradicionales, el cual también ha sido objeto de distintos ataques [Xeno6].

Amenazas basadas en aspectos físicos

- *Localización física* (Am10): Una localización física más precisa de los usuarios ha sido posible, en gran medida, gracias al uso de los terminales inteligentes. Esta información puede ser utilizada por el atacante con diversos motivos, desde el chantaje hasta la creación de perfiles de un usuario.
  - Material multimedia con información excesiva: La proliferación en la creación de material multimedia, con todos los beneficios que ello conlleva, puede provocar pérdidas de información considerables.

Los dispositivos GPS de los terminales pueden informar a los atacantes de la posición de la víctima.

    * Coordenadas GPS (Am11): los dispositivos móviles hoy en día cuentan, en su mayoría con un sistema de localización en el que se añaden las coordenadas a las mismas. Sin embargo, esta información sensible, que aporta mayor valor añadido, muchas veces no es usada correctamente por los usuarios.

      · Obtención de datos GPS (Am12): obtención de los datos del chip GPS disponible en la mayoría de los *smartphones*.

    * Extracción de información en las fotografías y vídeos (Am13)

      · Geolocalización de fotos (A23): en los metadatos de las fotos es posible encontrar la latitud y la longitud en la que se ha obtenido la foto.

      - Publicación explícita de su ubicación por parte del usuario (A24): con el auge de las redes sociales, en muchas ocasiones son los propios usuarios los que publican la información sobre su localización a traves de estas redes.
      - Lugares reconocibles mediante técnicas CBIR (*Content-based Image Retrieval*) (A25): existen técnicas de reconocimiento automática de imágenes [SWS+00] que permiten, de forma automática, reconocer elementos de una imagen, lo que facilitaría la labor del atacante.

La miniaturización de los dispositivos favorece su pérdida o su robo.

- Perdida o extravío del dispositivo (Am14):
  - Miniaturización de los dispositivos (V09). La reducción constante de tamaño de los dispositivos favorece su pérdida, extravío o sustracción.
    - Extracción de datos sensibles no protegidos (A26): es posible extraer los datos de un teléfono robado o perdido, debido a la utilización de insuficientes medidas de seguridad para protegerlos.
    - Extracción de datos mediante técnicas forenses (A27): el atacante hace uso de técnicas forenses para obtener datos del terminal que han sido protegidos por el usuario o por el sistema operativo.
- Acceso al dispositivo de otras personas (Am15):
  - Sistemas de validación de acceso (PIN, patrones, reconocimiento facial) inseguros (V10).
    - Robo de información del dispositivo (A28): con acceso al dispositivo físico es posible enviar a otro dispositivo números de teléfono, datos de la agenda, fotos, etc.
  - No pedir contraseña antes de instalar ninguna aplicación (V11):
    - Instalación de aplicaciones sin permiso del usuario (A29): bien sea descargándolas de Internet, bien sea aprovechando vulnerabilidades de las aplicaciones del terminal, como por ejemplo el navegador. Para hacerlo, un atacante puede instalar aplicaciones sin conocimiento del usuario si no se exige un consentimiento expreso del deseo de instalarla.

- Modificación de los datos del teléfono (Am16):
  - Modificación o sustitución del sistema operativo del *smartphones* (V04):
    - Liberación por hardware, a través de *smartcard* (TurboSIM) (A30): es posible interceptar la comunicación entre la *smartcard* y el dispositivo con distintos objetivos (por ejemplo, liberar el dispositivo).
    - *Joint Test Action Group* (JTAG) (A31): es un estándar para la prueba y depuración de hardware. A pesar de que esta funcionalidad de depuración ya no es necesaria en los dispositivos móviles que se venden a los usuarios finales, la funcionalidad de JTAG a veces es todavía accesible. Esta funcionalidad permite inspeccionar el dispositivo a un nivel profundo, pudiendo dar lugar a vulnerabilidades explotables.
    - Análisis forense (A32): el análisis forense de los dispositivos permite extraer datos del teléfono, evitando de esta manera los sistemas de seguridad de los *smartphones*.

Existen varios métodos hardware para saltarse las limitaciones impuestas por los operadores o los fabricantes.

3.3.3 Análisis del grafo resultante

Una vez vistos todos los elementos que componen el grafo de la biblioteca de ataques desarrollada, vamos a analizar distintas características del mismo. El objetivo no es el de realizar un estudio exhaustivo sobre las características de este tipo de redes. Sin embargo, consideramos que un análisis del mismo puede revelar aspectos importantes de la naturaleza del propio grafo.

El grafo resultante se puede ver en el la figura 3.4. Éste representa todas las relaciones expuestas anteriormente. Como se puede apreciar en él, todas las amenazas, vulnerabilidades y ataques están íntimamente relacionados con los activos a los que amenazan. También se puede apreciar que el activo que más interacciones tiene es Ac01, que corresponde a los datos que almacena el terminal.

El grafo resultante tiene un valor de grado medio de 2, 162 y un tamaño de red de 3.

Este grafo está compuesto por 68 nodos y 147 aristas. En primer lugar, vamos a medir el grado medio (en ingles, *average degree*) del grafo [LM08]. El grado $d^o(v)$ de un nodo v determina el numero de enlaces que posee ese nodo. El grado medio del

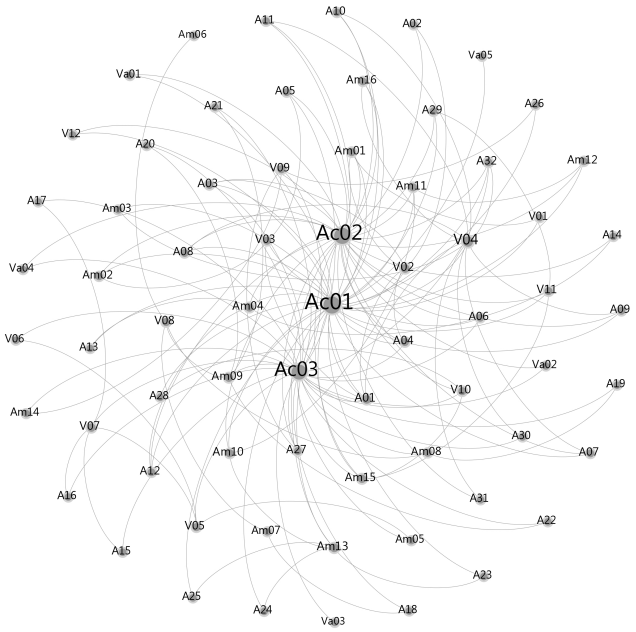


Figura 3.4: Grafo que representa la biblioteca de ataques aquí desarrollada.

grafo se calcula como la media del grado de todos los nodos que componen el grafo, como se puede ver en la ecuación 11.

$$d^o = \frac{1}{n} \sum_{v} d^o(v) \tag{11}$$

En este caso, el grafo resultante tiene como valor de grado medio 2,162.

El grado medio, el diámetro o la modularidad permiten caracterizar un grafo.

Por otro lado, el diámetro de la red es la distancia más larga entro dos nodos cualquiera de la red. En el caso de la red que hemos generado, el diámetro de la misma es de 3. El tamaño medio de la distancia entre los nodos, es decir, la media del número de saltos entre todos los nodos de la red, es de 2,71577.

La modularidad de la red [BGLL08] determina si existen conjuntos de nodos muy interconectados. Aplicando este algoritmo, podemos determinar que existen 5 conjuntos de nodos agrupados, destacando principalmente los que se encuentran alrededor de los activos a analizar.

Por último, analizamos la densidad del grafo. Esto permite medir qué tan completo es el grafo. Se denomina grafo completo a aquel grafo cuyos nodos están conectados todos entre sí. En este caso, obtenemos un valor de 0,032, lo que denota que se trata de un grafo poco concentrado.

Con todos estos datos, podemos concluir que el grafo generado por el banco de ataques esta centrado en torno a los activos, lo que indica la importancia de los mismos. Por otro lado, se trata de un grafo con poca conectividad, lo que indica que representan un amplio abanico de aspectos desde el punto de vista de la seguridad. Finalmente, podemos concluir que, si bien es cierto que representan un amplio abanico, la distancia de los nodos entre sí es pequeña.

3.4 RESULTADOS OBTENIDOS

Para evaluar el modelo, se otorgó más peso, por un lado a la privacidad, y por el otro se les otrogó el mismo peso a todos los valores.

A continuación detallamos los resultados obtenidos aplicando el modelo visto en el apartado 3.2 a la biblioteca de ataques desarrollada en la sección 3.3. Para ello, mostraremos los valores obtenidos al aplicar esta metodología con dos configuraciones disintas. La primera de ellas, prima la privacidad sobre el resto de valores, otorgándole el máximo peso a este valor y el mínimo al resto de valores. Con ello, veremos cuáles son las amenazas más significativas para este valor. Posteriormente, evaluaremos las amenazas más importantes a las que se enfrentan estos dispositivos dándoles a todos los valores el máximo peso posible.

Tabla 3.9: Configuración del modelado de amenazas para el primer experimento.

ID	Descripción	Valor
Va01	Privacidad	10
Va02	Activos financieros	1
Va03	Seguridad física	1
Va04	Disponibilidad	1
Va05	Reputación	1

En el primer experimento, la conectividad WiFi no confiable fue clasificada como la mayor amenaza para la privacidad.

En primer lugar, analizamos el resultado obtenido para la configuración del primer experimento, la cual se puede consultar en la tabla 3.9. En ellos, se ha ponderado la privacidad sobre otros elementos. El resultado obtenido puede verse en la tabla 3.10.

El primer experimento pone el énfasis en la privacidad. En él podemos observar que las primeras posiciones aparecen la conexión con redes WiFi no confiables y la pérdida del dispositivo. Ambas amenazas afectan directamente a la privacidad de los da-

Tabla 3.10: Tabla de resultados del modelado de amenazas ponderando la privacidad.

ID	Descripcion	Valor
Am05	Conexión a WiFi no confiable.	1,527921625440
Am14	Perdida o extravío del dispositivo.	1,525620685020
Am15	Acceso al dispositivo de otras personas.	1,271246376980
Am16	Modificación de los datos del teléfono.	1,108171884360
Am02	Transmisión de los datos sin conocimiento del usuario.	1,099465622500
Am12	Obtención de datos GPS.	1,092327746250
Am08	Fragilidad de la infraestructura de mensajes cortos.	1,032704668240
Am01	Instalación de aplicaciones.	0,826143237394
Am10	Localización física.	0,820519130584
Am11	Coordenadas GPS.	0,820211217027
Am04	Acceso de datos por parte de las aplicacioness.	0,659408608945
Am09	Debilidades en los sistemas de *backend*.	0,626858737758
Am03	Modificación de los datos del teléfono.	0,595454211645
Am13	Extracción de información en las fotografías y vídeos.	0,548779507033
Am06	Amenazas relacionadas con la infraestructura de red telefónica	0,053792218377
Am07	Criptografía de la red GSM	0,030403831338

tos que se envían a través del dispositivo y los que contienen. También podemos clasificar dentro de este grupo a la tercera amenaza, que es la que se refiere al acceso al dispositivo de otras personas. Estas tres amenazas son las que más en peligro ponen la privacidad del usuario, según el modelo desarrollado.

Las siguientes amenazas están relacionadas con las aplicaciones. En concreto, estas amenazas tienen una relación directa con el *malware*. De esta forma, se comprueba que el software con intenciones aviesas tiene una influencia muy directa en la privacidad de los *smartphones* o, para ser más precisos, en su falta.

También cabe destacar que las distintas amenazas relacionadas con la infraestructura de red están situadas en las últimas posiciones de la clasificación, a excepción de aquellas relacionadas con el envío de mensajes cortos. Se puede deducir de ello que, según el modelo generado, la influencia de la infraestructura dentro de la privacidad es muy limitada.

Tabla 3.11: Configuración del modelado de amenazas para el segundo experimento.

ID	Descripción	Valor
Va01	Privacidad	10
Va02	Activos financieros	10
Va03	Seguridad física	10
Va04	Disponibilidad	10
Va05	Reputación	10

El segundo experimento considera todos los valores igual de probables. Estos resultados pueden verse en la tabla 3.12. El objetivo era comprobar cuáles son las amenazas más peligrosas en estos dispositivos. Lo primero que llama la atención es la escasa diferencia entre los distintos valores.

En el ranking, la primera posición es para la modificación de los datos en el teléfono, seguido por la transmisión de los mismos. Este par de amenazas, muy ligadas al *malware*, muestran que la privacidad es uno de los aspectos que más puede sufrir dentro del uso de los terminales móviles.

En el segundo experimento, a igualda de importancia todos los valores es la modificación de los datos del teléfono los que adquieren una mayor peso.

Tras ellas, se sitúan la obtención de datos del GPS, que combina los tres activos analizados, seguidas por amenazas que afectan directamente a la privacidad, como son el acceso al dispositivo por parte de otras personas y la conexión a redes WiFi no confiables.

Tabla 3.12: Tabla de resultados del modelado de amenazas dando la misma importancia a todos los valores.

ID	Descripcion	Valor
Am16	Modificación de los datos del teléfono.	0,208171884
Am02	Transmisión de los datos sin conocimiento del usuario.	0,199465622
Am12	Obtención de datos GPS	0,192327746
Am15	Acceso al dispositivo de otras personas.	0,191246377
Am05	Conexión a WiFi no confiable	0,177921625
Am14	Perdida o extravío del dispositivo	0,175620685
Am01	Instalación de aplicaciones	0,151143237
Am10	Localización física	0,145519131
Am03	Modificación de los datos del teléfono.	0,145454212
Am11	Coordenadas GPS	0,145211217
Am08	Fragilidad de la infraestructura de mensajes cortos.	0,132704668
Am04	Acceso de datos por parte de las aplicacioness.	0,119408609
Am13	Extracción de información en las fotografías y vídeos.	0,098779507
Am09	Debilidades en los sistemas de *backend*.	0,086858738
Am06	Amenazas relacionadas con la infraestructura de red telefónica	0,053792218
Am07	Criptografía de la red GSM	0,030403831

Estos resultados muestran cómo la privacidad es el valor que más importancia le concede el modelo cuando igualamos el peso para todos los elementos, haciendo especial énfasis en aquellas amenazas relacionadas con el software.

3.5 LIMITACIONES DEL MODELO

El modelo es muy dependiente del grafo generado.

En el desarrollo del modelo hemos percibido una serie de limitaciones del mismo. En primer lugar, éste es muy dependiente del banco de ataques que se desarrolle para nutrirlo. Por ello, la creación del banco debe de ser lo más completo posible y abarcar la mayor cantidad de elementos posibles.

En este trabajo doctoral hemos desarrollado el primer banco de ataques sobre *smartphones*, pero dista de ser completo. Por ello, se está trabajando en una herramienta que permita a la comunidad de investigadores alimentar el banco para permitir al modelo ser más preciso.

Un banco de ataques más completo reportaría unos resultados más precisos.

Por otro lado, la métrica utilizada para calcular el peso de las amenazas puede ser mejorada. Si bien la métrica desarrollada en este trabajo tiene en cuenta las relaciones de cada uno de los elementos con los que están relacionadas las amenazas, es posible desarrollar una métrica que explote mejor la información derivada del grafo. En esta línea se trabajará en el futuro para mejorar el modelo.

3.6 CONCLUSIONES

En este capítulo hemos desarrollado un nuevo modelo que permite caracterizar las situaciones de riesgo que se producen sobre los *smartphones*. En concreto hemos desarrollado un método para, basado en los distintos caminos de ataque que se producen, generar un grafo que permita clasificar las amenazas a las que se enfrentan los usuarios de este tipo de terminales.

Este modelo explota dos tipos de información con el fin de realizar la clasificación. En primer lugar, explota el conocimiento experto, a través del cual se les dota de peso a cada uno de los nodos del camino. Posteriormente, se aplica el algoritmo *PageRank* para extraer información del grafo y, combinada con el conocimiento experto, otorgar el peso a cada uno de los nodos del grafo.

Para alimentar este modelo se ha desarrollado una biblioteca de ataques. En ella hemos visto una gran cantidad de ataques, vulnerabilidades y amenazas a las que este tipo de terminales se enfrenta cada día. Por resumir en números, hemos identificado 3 grandes grupos de activos, 32 ataques distintos, 12 vulnerabilidades y 16 amenazas distintas, todos ellos categorizados en función del tipo de amenaza que representa.

A nivel general, podemos indicar que este tipo de dispositivos representan un nuevo escenario, en el que se combinan estos elementos virtuales como reales. También se puede apreciar que comparte muchos elementos con las amenazas que se dan en las redes sociales [LSAGB10].

Los resultados que hemos obtenido al aplicar este modelo al banco de ataques generado se han centrado en evaluar cuáles son las principales amenazas a las que se enfrentan estos dispositivos, teniendo en cuenta aquellos valores o aspectos en los que el usuario quiere hacer énfasis a la hora de evaluar esta amenazas (por ejemplo, darle más importancia a la privacidad frente a la disponibilidad). Para ello, analizamos los resultados obtenidos centrándonos en la privacidad. Los resultados obtenidos reflejan la importancia de amenazas físicas, como puede ser la pérdida o extravío del dispositivo o el acceso al dispositivo por parte de otras personas, como las relacionadas con el canal de transmisión, a través de redes WiFi no seguras. También refleja la importancia de todos aquellos elementos software relacionados con el terminal, haciendo especial énfasis en dos de ellos: modificación de los datos del teléfono y la transmisión de los datos sin conocimiento expreso del usuario.

Por otro lado, hemos analizado el resultado obtenido dándoles a todos los elementos el mismo peso. En este caso, las amenazas relacionadas con el software son las que coparon los primeros puestos del análisis, así como todas aquellas amenazas relacionadas con los datos del terminal. En este caso concreto, se puede deducir que el *malware* es una de las mayores amenazas de seguridad para este tipo del terminales, a tenor de los resultados obtenidos con esta segunda configuración.

Este modelo generado presenta una serie de limitaciones. La primera de ellas es que es muy dependiente de la biblioteca de ataques. En nuestro caso, la biblioteca ha sido generada tras una larga investigación sobre los distintos ataques que este tipo de dispositivos han sufrido en los últimos años. Sin embargo, esta biblioteca es mejorable. Por ello, esta en proceso la liberación de una herramienta que permita a la comunidad alimentar esta

biblioteca de ataques para mejorar los resultados de la herramienta y ayudar a identificar a la sociedad, tanto investigadora como civil, cuáles son las amenazas a las que se enfrentan a la hora de usar estos dispositivos.

Por otro lado, el algoritmo que calcula el peso de los nodos de amenazas puede ser mejorado, explotando más la información proveniente de la red. En esta línea, estamos investigando para mejorar este algoritmo, a fin de que el resultado no sea tan dependiente de la estructura que tiene la red.

Este modelo presenta como punto fuerte que es posible evaluar cada uno de los nodos en función de los valores del resto de los nodos de la red. Si bien es cierto que nos hemos centrado en las amenazas, no es menos cierto que esa misma metodología puede ser utilizada para clasificar los ataques o las vulnerabilidades a las que se enfrentan.

En resumen, estos nuevos dispositivos traen consigo un nuevo escenario que requiere de nuevos métodos para categorizar estas nuevas amenazas, combinando amenazas del mundo digital con otras más tradicionales. Por ello, es necesario plantear nuevos modelos que caractericen este nuevo escenario, como el planteado en este trabajo doctoral. Del resultado obtenido aplicando este modelo podemos deducir que el activo más amenazado son los datos presentes en el terminal, obteniendo la máxima puntuación en el mismo todas aquellas amenazas relacionadas con los software, por lo que es necesario desarrollar nuevos métodos que protejan a los usuarios frente a las aplicaciones desarrolladas con fines maliciosos.

4 EL ECOSISTEMA DE ANDROID

«Todos somos muy ignorantes. Lo que ocurre es que no todos ignoramos las mismas cosas.»

A. Einstein
(1879 – 1955)

ÍNDICE

En este capítulo vamos a ver más a fondo cómo esta desarrollado el sistema operativo Android. Para ello, definiremos la composición del árbol de directorios, cómo están compuestos los binarios y cómo se ejecutan las aplicaciones. Pero también analizaremos aspectos importantes desde el punto de vista de la seguridad: la instalación de las aplicaciones. Analizaremos cómo se gestionan los permisos de las aplicaciones, así

como la especificación de estos permisos. Con todo ello, veremos cómo está conformado el modelo de seguridad de la plataforma.

4.1 MODELO DE SEGURIDAD DE ANDROID

A continuación se detalla el modelo se seguridad que implementa el sistema operativo Android. Primeramente, se mostrará un esquema general de la arquitectura del sistema operativo para, posteriormente, entrar en detalle en el modelo de seguridad. Sin embargo, es importante conocer conceptos particulares de la arquitectura, como son por ejemplo el «tiempo de vida» de las aplicaciones.

4.1.1 Arquitectura de Android

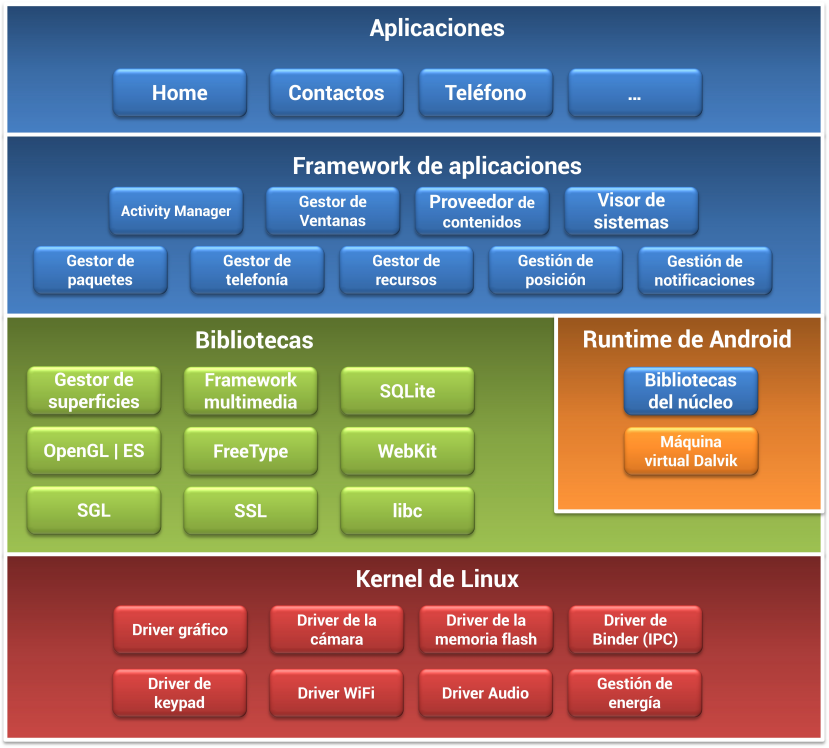


Figura 4.1: Esquema general de la arquitectura del sistema operativo Android.

La Figura 4.1, obtenida de la propia documentación de Android[1], ilustra la arquitectura de este sistema operativo. Empe-

1 http://developer.android.com/guide/basics/what-is-android.html

zando desde abajo, podemos observar el núcleo de Linux que posee. Android utiliza este núcleo para manejar los drivers del dispositivo, y la gestión de la memoria y de los procesos, además de la gestión de la energía y del tráfico de red. Los desarrolladores no pueden programar en esta capa directamente.

Android posee un núcleo basado en Linux.

La siguiente capa contiene las bibliotecas nativas de Android. Éstas, escritas en C/C++, son llamadas por los desarrolladores a través de interfaces de Java. En esta capa se puede encontrar el gestor de superficies encargado de la gestión de las ventanas, la gestión de las 2D/3D a cargo de *OpenGL*, el *Media Framework* que es el componente que se encarga de gestionar los codecs multimedia (como el MPEG-4, el H.264 o el MP3), la base de datos SQL, y el motor del navegador de web, entre otros.

El siguiente es el runtime de Android, el cual incluye la máquina virtual Dalvik. Esta máquina virtual ejecuta los ficheros con extensión *dex*, que son convertidos en tiempo de ejecución en clases estándar de Java y en un fichero con extensión *jar*. La razón por la que se utilizan éstos en lugar de ficheros nativos de Java es que este tipo de ficheros son más compactos y eficientes, punto éste crucial a la hora de hablar de dispositivos con memoria y batería limitadas, como son los *smartphones*. Dentro del runtime también se encuentran las bibliotecas de Java. Android provee un gran conjunto de paquetes de la versión *Java 5 Standard Edition*.

Las aplicaciones se ejecutan dentro de una máquina virtual, denominada Dalvik.

En el siguiente nivel está la capa del framework de aplicación. Esta capa está compuesta por las herramientas provistas por Google, así como extensiones y servicios escritos por los desarrolladores. El componente principal dentro de este framework es el *Gestor de Actividad*, el cual maneja el ciclo de vida de las aplicaciones y el *back-stack* (es decir, volver hacia atrás en las aplicaciones) de la navegación del usuario.

Finalmente, la capa superior es la capa de aplicaciones. La mayoría del código creado para la plataforma estará aquí, tanto las aplicaciones creadas por desarrolladores externos como las que vienen con el sistema operativo. Todas aquellas aplicaciones desarrolladas por Google conviven en esta misma capa, accediendo a través de la misma API pública que usan los externos, lo que permite que se puedan sustituir aplicaciones nativas del teléfono por otras que han sido desarrolladas por terceros.

4.1.2 Modelo de desarrollo de Android

El entorno de ejecución busca aislar los programas del resto del sistema.

Como hemos dicho anteriormente, gran parte de las aplicaciones de la plataforma están escritas en Java. Todo el código compilado, así como todo aquel archivo de recurso que sea necesario, es empaquetado dentro de un archivo con extensión *.apk*.

Cada una de las aplicaciones de Android se ejecuta en el siguiente entorno:

- Por defecto, cada aplicación se ejecuta en su propio proceso de dentro de GNU/Linux.
- Cada proceso tiene su propia instancia de la máquina virtual, con lo que el código de la aplicación está completamente aislado del resto de aplicaciones.
- Por defecto, cada aplicación es asignada a un único `ID` de usuario de GNU/Linux. De esta forma, los ficheros de la aplicación son visibles únicamente a ese usuario, aunque es posible establecer formas de compartirlos con otras aplicaciones.

Las aplicaciones de Android no tienen un único punto de entrada.

Debido a las necesidades de optimización, dadas por las limitaciones de los dispositivos, con este modelo de desarrollo se busca compartir elementos entre aplicaciones, dando, para ello, permiso a las aplicaciones. Para poder llevar a cabo esta tarea, el sistema debe ser capaz de iniciar cualquier parte de la aplicación cuando sea necesario, por lo que las aplicaciones de Android no tienen un único punto de entrada para todo. En vez de ello, hay 4 tipos de componentes esenciales que el sistema puede instanciar y ejecutar en caso necesario. Los componentes son los siguientes:

- `Activity`. En caso de que la aplicación tenga interfaz de usuario, tendrá al menos un elemento `Activity`. Una de las principales tareas de `Activity`, es mostrar los elementos de la Interfaz de Usuario (User Interface, *UI*).
- `Service`. Si el ciclo de vida de la aplicación es prolongado, como por ejemplo en una utilidad de sincronización de datos de fondo que se ejecute de forma continuada, debe incluirse un `Service`.
- `BroadcastReceivers`. Si una aplicación desea recibir y responder a un evento global, como por ejemplo una llamada de teléfono o un mensaje de texto, debe registrarse como

`BroadcastReceivers`. También sirve para eventos generados por una aplicación, denominados *intents*, que no tienen un destinatario concreto.

- `ContentProviders`. Si una aplicación gestiona datos y debe mostrarlos a otras aplicaciones ejecutadas en el entorno de Android, es necesario implementar esta clase. Éste es el contenedor que permite proporcionar datos a una actividad o a un servicio de la misma aplicación o de otra diferente.

Existen cuatro componentes esenciales que el sistema puede instanciar: Activity, Service, BroadcastReceivers y ContentProviders.

Además de estos cuatro componentes básicos, existen dos componentes más que buscan mejorar el flujo de navegación de la aplicación, un aspecto harto importante en este tipo de dispositivos. El primero de ellos es el denominado objetivo o `Intent`. Este objeto es una declaración de necesidades, formado por fragmentos de información que describen la acción o servicio deseados. El otro es el `IntentFilter`, que es una declaración de la capacidad e interés por ofrecer asistencia a los componentes que la necesitan. Este tipo de objetos pueden ser genéricos o específicos con respecto a los elementos `Intent` a los que presta servicio.

A modo de resumen, podemos decir que una aplicación Android contiene una serie de elementos que podríamos considerar como mínimos: `Activity`, `Service`, `BroadcastReceivers` o `ContentProviders`, de los que algunos de ellos muestran los `Intent` que desean procesar a través del `IntentFilter`. Todos estos fragmentos de información deben combinarse para poder ejecutar la aplicación. Esto se hace en el archivo `AndroidManifest.xml`, que se encuentra en el directorio raíz de la aplicación y que contiene todas las relaciones desarrolladas en tiempo de diseño y sus `Intent`. En este archivo también se definen los privilegios de seguridad que requiere la aplicación. Así, si por ejemplo, la aplicación tiene que acceder a los contactos del dispositivo, se definen sus permisos a través de la etiqueta que se muestra en Código 1:

AndroidManifest.xml se encarga de la definición de permisos y privilegios de ejecución de la aplicación.

Código 1: Etiqueta para asignar permisos de lectura de contactos.

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

Todos éstos son los elementos principales que hay que tener en cuenta a la hora de desarrollar una aplicación en Android. De la misma forma, es importante tener en cuenta estos aspectos a la hora entender los aspectos de seguridad de este sistema operativo.

La API de Android posee muchas clases y servicios provenientes de Java.

Además de todos estos elementos, la API de Android provee de una serie de clases y servicios que ayudan a mejorar la seguridad de las aplicaciones. Estas herramientas están heredadas del entorno de Java, y concentrándose en el paquete `java.security`. Dentro de estos recursos, encontramos, entre otros, el `java.security.acl`, que provee de todas las clases e interfaces necesarias para constituir listas de control de acceso; el `java.security.cert`, encargado de verificar y administrar los certificados X.509, o el paquete `java.security.spec`, que facilita el uso de algoritmos de firma y cifrado. Dentro de `java.security`, cabe destacar el paquete `java.security.interfaces`, que se encarga de proveer los interfaces necesarios para la generación de los siguientes elementos:

- Claves para el uso del algoritmo RSA de cifrado asimétrico, a través del estándar PKCS#1.
- Claves para el uso del DSA (*Digital Signature Algorithm*), especificado por el FIPS-186.
- Claves para el algoritmo genérico de cifrado asimétrico de curva elíptica.

4.1.3 El Android Market

El principal punto de entrada de la instalación de una aplicación en este sistema operativo es el *Android Market*. El objetivo de esta plataforma es el de facilitar la comunicación entre los desarrolladores y los usuarios. Así, los desarrolladores tienen un espacio donde poder dejar las aplicaciones y los usuarios pueden acceder a ellas desde el mismo teléfono. Esto evita la dispersión que existía en las aplicaciones para móviles, ya que cada desarrollador tenía su propio espacio para poder mostrarlas.

El Android Market busca priorizar las aplicaciones relevantes frente a las que no aportan novedades.

El objetivo final del Android Market es el de crear un entorno darwiniano en el que las buenas aplicaciones sobresalgan sobre las malas, que van cayendo en el olvido. De esta forma, se premia a los mejores desarrolladores, forzando al resto a intentar sobrevivir.

A la hora de implementar la seguridad en el Android Market, se ha buscado conseguir un equilibrio entre la seguridad y el mantenimiento de un sistema abierto para que cada usuario pueda instalar las aplicaciones que considere oportunas. Así, el Android Market no es la única manera de instalar aplicaciones en los *smartphones* con sistema operativo Android, pero sí que es

la forma más común de hacerlo. Así, no nos encontramos dentro de un entorno cerrado como ocurre en la AppStore, la tienda de aplicaciones de Apple[2].

En cuanto a la firma de aplicaciones, cada uno de los desarrolladores firma sus propias aplicaciones, sin necesidad de una entidad certificadora. De esta forma, el desarrollador se vincula al Android Market a través del certificado que ha creado él mismo. Esto permite a los desarrolladores tener más de un perfil dentro del Market a la hora de desarrollar las aplicaciones. Las razones que esgrime[3] el jefe de seguridad de Android, Rich Cannings, para tomar esta decisión se basan en los problemas de verosimilitud que empiezan a sufrir las entidades de certificación, por lo que el usarlas como intermediarios no garantiza la veracidad de las aplicaciones. Por eso, al no existir un mecanismo que lo garantice, optaron por este método intermedio.

Las aplicaciones poseen la firma digital de los autores.

El objetivo que se logra con este modelo de firmado es doble. Por un lado, en lo que se refiere a la seguridad de la aplicación, y mediante el uso de estos certificados, es posible determinar quién puede acceder a los permisos basados en firma y quién puede compartir el identificativo de usuario de la aplicación. Por el otro, se garantiza que las actualizaciones de las aplicaciones las hacen los propios desarrolladores, dificultando la modificación de éstos por terceros.

4.1.4 Modelo de Seguridad de Android

A la hora de desarrollar el modelo de seguridad, se asumieron las siguientes hipótesis [Cano9]:

- El número de recursos y la cantidad de tiempo disponible para el desarrollo del sistema es finito.
- En general, los seres humanos tienen dificultades para entender los riesgos. Esta apreciación está basada en la experiencia de Google en materia de seguridad.
- A la hora de crear el sistema es mejor asumir que la mayoría de los desarrolladores no entienden de seguridad, de la misma forma que la mayoría de los usuarios no la entienden. Para solventarlo, era necesario que las cuestiones de seguridad fueran transparentes y obligatorias.

2 Tienda de aplicaciones desarrollada por Apple para distribuir las aplicaciones en los sistemas operativos de la compañía.

3 http://www.usenix.org/events/sec09/tech/tech.html#cannings

Figura 4.2: Ciclo continuo del modelo de seguridad implementado en Android.

El modelo de seguridad de Android se basa en cuatro dogmas: prevenir, minimizar, detectar y reaccionar.

Basado en estos conceptos, se desarrollaron los cuatro dogmas de la filosofía de seguridad que sustenta Android[4]:

- Es necesario *prevenir* las brechas de seguridad antes de que ocurran.
- Es necesario *minimizar* el impacto de estas brechas de seguridad.
- En necesario *detectar* las vulnerabilidades y las brechas de seguridad.
- Es necesario *reaccionar* ante las vulnerabilidades y brechas de seguridad con rapidez.

Así, el modelo de seguridad de Android se basa en un ciclo continuo que incluye estas etapas (ver Figura 4.2). Todas las acciones referentes a la seguridad de Android se pueden clasificar en uno de estos cuatro puntos:

Prevención

Google ha puesto gran énfasis en este punto. Así, una de los características principales del sistema operativo, el ser de código

4 http://www.nis-summer-school.eu/presentations/kralevich.pdf

abierto, facilita el trabajo de la comunidad a la hora de localizar brechas de seguridad. El hecho de utilizar un gran número de bibliotecas *open source* (más de 100), hace que la seguridad de todas ellas sea evaluada por el grupo de expertos que la desarrolló. De esta forma, el esfuerzo de los programadores de Android se centra en la seguridad del conjunto, y no en la de cada uno de los elementos, dado que para ello ya está trabajando la comunidad que lleva cada una de las bibliotecas.

Al tratarse de código abierto, existe un gran número de vigilantes potenciales del código.

Por otro lado, el equipo de seguridad de Android está compuesto por expertos de demostrada valía, como son los expertos en seguridad de Google y responsables de seguridad de los partners que colaboraron en el desarrollo. De esta forma se aglutina la experiencia de todos ellos para trabajar en la seguridad del sistema.

En vista de los recursos y tiempo limitado del que disponían, el equipo se centró en la seguridad de las áreas de alto riesgo, como son los drivers y codecs, la posibilidad de recibir ataques remotos y la creación de nuevas y personalizadas características de seguridad. Se centraron en la creación de características que requieran un bajo esfuerzo pero que provean un gran beneficio, como la *ProPolice stack overflow protection* (que previene la manipulación de los datos de la aplicación en la pila) o la *Heap protection in dlmalloc* (funciones provenientes de OpenBSD que proveen una protección adicional y dificultan la realización de ataques por desbordamientos de pila).

Minimizar

Pese al esfuerzo por prevenir, hay que ser conscientes de que (i) las vulnerabilidades existen, (ii) hay código malicioso que el usuario va a instalar en su *smartphone* y (iii) habrá código mal escrito que pueda causar daños a los datos de los usuarios. Por ello, es necesario *minimizar* el impacto que estos problemas de seguridad tienen. Para ello, se ha seguido el mismo modelo que siguen los navegadores Web. En otras palabras, mientras que los sistemas operativos tradicionales están basados en la máquina y hacen separación de los usuarios, esta arquitectura no tiene sentido dentro de los dispositivos móviles, ya que los sistemas operativos móviles son únicamente para un único usuario. Por ello, se crea un único `UID` (traducción del vocablo inglés *Identificador de Usuario*) por cada una de las aplicaciones, de tal forma que los privilegios que tienen cada una de las aplicaciones son

El diseño del sistema busca minimizar el impacto de las vulnerabilidades.

Figura 4.3: Modelo de separación por defecto de privilegios

independientes. Se intentó diseñar esta separación de privilegios lo más transparente posible a fin de facilitar el desarrollo.

Uno de los elementos claves a la hora de minimizar los daños, es la aplicación de una caja de arena o *sandbox*, que es un mecanismo por el cual se aíslan los procesos de ejecución en entornos separados. De esta forma se completa el modelo de separación por defecto de privilegios, puesto que cada aplicación se ejecuta en un entorno distinto con un usuario diferente, tal y como se puede apreciar en la Figura 4.3.

Con el modelo de Android, basado en máquina virtual, es sencillo compartir los recursos del dispositivo.

Con este modelo es fácil compartir de forma segura recursos como la CPU, la memoria, los ficheros, etc., pues todos ellos tienen permisos conforme a su necesidad de compartir datos. Para ello, es conveniente poner todos esos recursos cerca de los controles de acceso, de forma que se cree un perímetro más pequeño y, por consiguiente, más fácil de defender.

A la hora de otorgar permisos a las aplicaciones, se sigue el modelo de lista blanca: en un primer paso, dar el mínimo número de permisos por defecto para que, posteriormente, sea el usuario el que le dé acceso a los recursos. Para que este modelo funcione, es necesario tener claro que hay que hacerle el menor número de preguntas posibles al usuario, siendo cada una de ellas lo más comprensibles posibles. En Android es posible otorgar hasta 130[5] permisos distintos, lo que ofrece una gran

5 http://developer.android.com/reference/android/Manifest.permission.html

granularidad al desarrollador, pero también resta comprensibilidad a los usuarios. Por defecto, las aplicaciones de escritorio de GNU/Linux poseen demasiados privilegios, por lo que se rebajan esos privilegios por defecto. En cualquier caso, confiar en que el usuario tome la decisión de seguridad correcta tiene su riesgo.

Otro de los puntos en los que se puso un gran esfuerzo a la hora de minimizar los daños es en el *OpenCore Media Library*, que es el servidor de codecs para multimedia. Los codecs multimedia tienden a ser trozos de código muy complejos, por lo que suelen ser muy inseguros. Por ello, son relegados a procesos con muy bajo nivel de privilegios con un usuario específico para el servidor multimedia que los ejecuta. De esta forma, si un atacante consigue hacerse con este usuario, únicamente tendrá acceso a los codecs multimedia, y no a otras partes del sistema (p.ej., el correo electrónico o la agenda de contactos).

Detectar

Android busca el apoyo de la comunidad para detectar los problemas de seguridad existentes.

Cualquier problema de seguridad, incluso los de bajo impacto, sigue siendo un problema de seguridad. Por ello, pese a tener procesos internos de detección de errores, como pueden ser auditorias de código, *honeypots*[6], etc., Android busca apoyos para detectar estos errores. Estos apoyos se centran en dos grupos principalmente. El primero es el de los investigadores de seguridad. Así, distintos expertos como Charlie Miller o Collin Mulliner han reportado distintos problemas, como por ejemplo una vulnerabilidad a la hora de procesar los ficheros MP3 o un ataque a través de una biblioteca de WebKit (el motor de renderizado de páginas web que utiliza el navegador instalado por defecto de Android). Por otro lado, el sistema de reportes de usuarios permite que los usuarios identifiquen aplicaciones que pueden dañar el terminal, de forma que son revisadas y retiradas.

Reaccionar

El sistema de actualizaciones pretende dar solución temprana a los problemas detectados.

Por último, la *reacción* es el punto clave tras la detección. En este sentido, Android dispone de una actualización automática que funciona a través de las redes 3G o WiFi, denominada *Android's Over-The-Air* (OTA). Este sistema de actualizaciones permite un rápido despliegue de las actualizaciones de seguridad a todos los terminales. La interacción del usuario es opcional, por lo que

6 Conjunto de ordenadores y software colocados con vulnerabilidades intencionadamente, a fin de atraer a los atacantes y su posterior análisis.

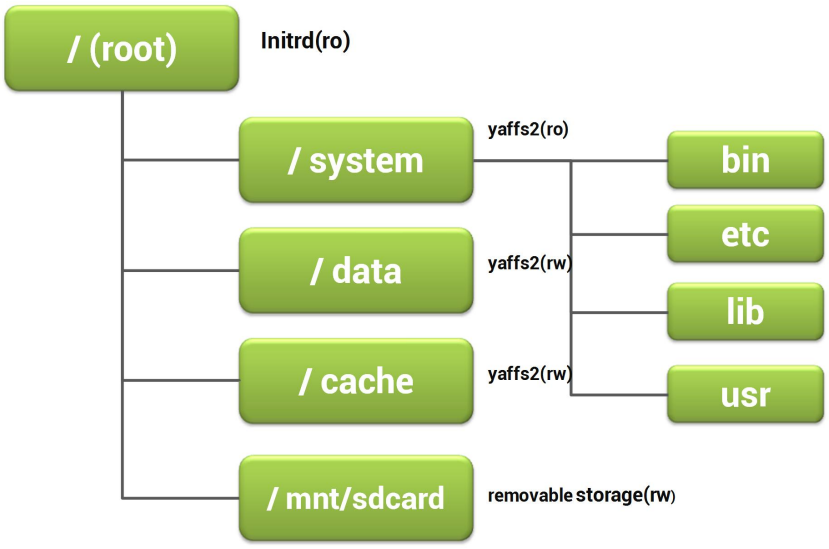


Figura 4.4: Árbol de directorios del sistema operativo Android

se consigue un alto ratio de efectividad en las actualizaciones. Además, gracias a este sistema el *smartphone* es capaz de actualizarse sin necesidad de ningún otro dispositivo, como por ejemplo un cable y un ordenador, facilitando la instalación de las actualizaciones. Sin embargo, en la práctica, en este punto se encuentran con un problema ya que no queda claro quién es el responsable de la actualización. De esta manera, existen versiones del sistema operativo provistas por la propia Google, otras por los fabricantes de los dispositivos y finalmente las distribuidas por las operadoras; cada una de ellas tiene una manera distinta de actualizar el dispositivo.

4.2 ÁRBOL DE DIRECTORIOS DE ANDROID

El árbol de directorios de Android se puede ver completo en la figura 4.4. Está compuesto principalmente por 3 puntos de montaje cada uno de los cuales alberga una parte bien diferenciada del sistema.

El árbol de directorios de Android sigue el mismo esquema que el de otros sistemas UNIX.

El primero de ellos es la carpeta `/system`, la cual almacena todos aquellos programas y recursos necesarios para poder ejecutar el sistema operativo. Esta carpeta es únicamente de lectura, y alberga una estructura de ficheros típica de entornos UNIX, como son `/bin` para los comandos binarios del sistema, `/etc` para albergar los ficheros de configuración, `/lib` para almacenar los

recursos compartidos por los distintos elementos del sistema, y `/usr` como jerarquía secundaria para datos compartidos de solo lectura.

Por otro, lado tenemos la carpeta `/data`, que almacena todos aquellos elementos necesarios para la ejecución de cada una de las aplicaciones. Dentro de este punto de montaje se encuentra una carpeta por cada una de las aplicaciones que tiene instalada el terminal. En su interior, cada carpeta alberga todos aquellos elementos necesarios (p.ej., binarios, recursos o datos) para que la aplicación pueda ejecutarse. La carpeta entera tiene un único identificador de usuario `UID` para todos los elementos que almacena.

Por último, el punto de montaje `/mnt/sdcard` sirve para acceder a la tarjeta de memoria del dispositivo. Aun cuando no exista la tarjeta de memoria en él, este elemento sigue apareciendo, ya que es el encargado de almacenar ficheros de gran tamaño, como pueden ser los ficheros multimedia (p.ej., fotos sacadas de la cámara del dispositivo) o ficheros que requieren ser almacenados por otras aplicaciones, como puede ser un fichero descargado del navegador.

4.3 INVESTIGACIONES PREVIAS SOBRE EL SISTEMA ANDROID

Los sistemas que gobiernan los *smartphones* son considerados como sistemas operativos completos sobre dispositivos limitados y, como tal, los investigadores han aplicado las técnicas tradicionales adaptadas a las limitaciones que otorga este nuevo entorno, como por ejemplo la medición de la integridad y la certificación a distancia [DSW09].

Sin embargo, los *smartphones* no se circunscriben sólo al sistema operativo, sino que involucran a una gran cantidad de partes interesadas. Según la especificación móvil del «Grupo de Computación Confiable» (del inglés *Trusted Computing Group* , TCG) [TGC07], existen cuarto grupos de interés bien diferenciados: el fabricante del terminal, la red del operador, servicios de terceros y el usuario final. A fin de mejorar en la seguridad en este nuevo entrono, se definió el *Mobile Remote-Owner Trusted Module* (MRTM) y el *Mobile Local-Owner Trusted Module* (MLTM) para conformar las tareas del «módulo de plataforma segura» (*Trust Module Plattform*). Éste es el nombre de un pliego de con-

La especificación del Trusted Computing Group describe a cuatro grupos de interes: fabricantes, operadores, servicios de terceros y usuarios.

diciones publicadas por el TCG detallando un criptoprocesador seguro que puede almacenar claves de cifrado que protegen la información.

Una arquitectura compatible con esta especificación debe tener un dominio lógico del tipo MRTM en cada uno de los grupos de interés, a excepción del dominio de usuario final, que utiliza un dominio lógico MLTM.1 Zhang *et ál.* [ZAS09] desarrollaron un sistema que cumple esta especificación utilizando SELinux para aislar estos dominios. En la misma linea, Nauman *et ál.* [NKZS10] implementaron un sistema de certificación remota para Android, incluyendo la medición de las aplicaciones.

Algunos investigadores han desarrollado un sistema de aceptación de permisos parciales en Android en base a circunstancias.

La virtualización se ha aplicado a los teléfonos para proporcionar la separación estricta entre los conjuntos de aplicaciones. La solución de VMware móvil [VMw] instala una máquina virtual alojada en el terminal que contiene las aplicaciones de negocio. Sin embargo, con este modelo, un usuario puede instalar aplicaciones de carácter personal, lo que puede comprometer al terminal y, por extensión, a las aplicaciones de uso profesional. Para contrarrestarlo, los laboratorios de OK [Lab] utilizan L4 como un hipervisor para ejecutar simultáneamente Android, Symbian y Windows. Por otro lado, Lee *et ál.* [LSJM08] fuerza cumplir las políticas de MAC para la comunicación entre las máquinas virtuales que se ejecutan en un teléfono. Este tipo de arquitectura tiene similitudes con una configuración de TrustZone ARM para Linux embebido [Win08].

En lo que se refiere a la plataforma Anrdroid, uno de los puntos más recurrentes en el tema de los permisos es la imposibilidad de aceptarlos de forma individual. Estos deben de ser aceptados en su conjunto o rechazados de la misma manera. En este sentido, Nauman *et ál.* presentaron Apex [NKZ10], una aproximación que cambia este modelo de todo o nada permitiendo una mayor granularidad. Para ello se fijó tanto en la posibilidad de aceptarlos de forma individual como en base a las circunstancias que se dan en un momento concreto, como puede ser la hora del día, por ejemplo. En esta misma línea, Conti *et ál.* presentaron CRePE [CNC11], un sistema de permisos alternativo basado en la definición, por parte del usuario, de políticas contextuales (p. ej., la localización en la que se encuentra el usuario).

Otro enfoque es el utilizado por Ontang *et ál*, que propusieron Saint [OMEM09a], un modelo que busca mejorar la capacidad de los desarrolladores a hora de definir las políticas de seguridad de sus aplicaciones. Así, restringe tanto la instalación de las aplicaciones como las interacciones de estas en tiempo de eje-

cución, basándose en distintos enfoques: permite dar permisos a otras aplicaciones mediante la configuración, mediante firmas (como pueden ser las listas blancas o listas negras) o mediante el contexto del medio ambiente (como la ubicación).

El estudio más profundo sobre la materia ha sido el realizado por Felt *et ál* [FCH+11]. En él, hicieron un análisis exhaustivo sobre el sistema de permisos que se desarrolló para la plataforma. Modificaron el sistema operativo de Android, en concreto la versión 2.2, para que volcara en un log todas aquellas peticiones de permisos del sistema. Identificaron 1.207 llamadas (el 6,26 % de los métodos existentes) a la API que requerían comprobación de permisos. En sus conclusiones reflejaron que, en base a sus experimentos, una de cada tres aplicaciones de Android requiere más privilegios de los que realmente necesita para ejecutar su funcionalidad.

Según Felt et ál, 1 de cada 3 aplicaciones piden más permisos de los que necesitan.

Barrera *et ál* [BKvOS10] desarrollaron una nueva metodología para el análisis empírico de los sistemas de seguridad basado en permisos, con su posterior aplicación al sistema operativo Android. Este modelo, basado en los mapas auto-organizativos de Kohonen [Koh90]. Sus resultados demostraron que existe un pequeño conjunto de permisos que son usados muy frecuentemente, mientras que otro gran conjunto de permisos no son usados casi nunca.

Otras investigaciones han analizado las limitaciones de los frameworks de políticas a nivel de aplicación. Shin *et ál.* [SKFT10] modelaron el lenguaje de políticas de la seguridad de Android entre aplicaciones. Mediante el uso de este modelo, es posible identificar un ataque de escalada de privilegios de una aplicación, bien sea por la explotación de las vulnerabilidades de otras aplicaciones o por colusión [SKK+10]. Para solucionar este problema es necesario el uso de la traza transitiva de los permisos. Algunas investigaciones, como la de Chaudhuri [Cha09], definen una aproximación basada en el lenguaje para evaluar la comunicación entre las aplicaciones. ScanDroid [FCF09] utiliza este modelo para certificar las aplicaciones automáticamente basándose en su código fuente. Sin embargo, la comparación de los permisos de Android no es trivial, dada su granularidad, dificultando enormemente este enfoque, ya que es necesario un orden parcial basado sobre los permisos de Android ya existentes.

Shin et ál. diseñaron un lenguaje de políticas para Android.

Por su parte, Bugiel *et ál* [BDD+11] desarrollaron y desplegaron un sistema de control de acceso obligatorio sobre el núcleo de Linux que extiende el motor de referencia que viene con el sistema operativo. Usando Tomoyo [HHT04], este sistema evita

a nivel de aplicación los ataques de escalada de privilegios en *runtime*. Según su evaluación, su sistema es resistente a distintos ataques [EOM08, DDSW11, LRW10, SZZ+11], con una sobrecarga sobre el sistema de 0,015 ms, lo que es imperceptible para el ojo humano.

4.4 COMPOSICIÓN DE LOS BINARIOS DE ANDROID

Las aplicaciones de Android se instalan mediante un fichero APK. Este fichero es un paquete que contiene todos los recursos necesarios para llevar a cabo la instalación de la aplicación. Estos recursos están compuestos por:

Los binarios están compuestos por el código de la aplicación, los recursos necesarios y el fichero de manifiesto.

- El código de la aplicación compilada: estos ficheros con extensión `.dex` (*Dalvik Executable*) contienen el código ejecutable de la aplicación.
- Recursos: imágenes, sonidos y demás recursos necesarios para poder ejecutar la aplicación.
- Fichero de manifiesto `AndroidManifest.xml`: en este fichero se especifican detalles de la aplicación y de su configuración, como por ejemplo los permisos que la aplicación necesita para poder ser ejecutada

De todos ellos, los elementos más importantes son el fichero de manifiesto y los ficheros compilados. A continuación veremos sus principales características.

4.4.1 El fichero AndroidManifest.xml

El fichero de manifiesto, `AndroidManifest.xml`, contiene toda la información esencial que el sistema Android necesita antes de poder ejecutar la aplicación.

Entre otras funciones, realiza las siguientes tareas:

- Contiene el nombre del paquete de Java de la aplicación.
- Describe e identifica los componentes de la aplicación (actividades, servicios, etc.).
- Declara los permisos que necesita la aplicación, tanto para interactuar con el sistema como para que otras aplicaciones interactúen con élla.

- Declara la versión mínima de la API de Android que requiere para ser ejecutada.

- También incluye la lista de bibliotecas que la aplicación requiere.

El fichero de manifiesto es uno de los elementos vitales a la hora de configurar la seguridad de las aplicaciones, ya que en él se detallan dos de los elementos críticos a la hora de evaluar la seguridad de la misma. El primero de ellos son los permisos, los cuales se verán con más detalle en el punto 4.6.2. Por otro lado, este fichero determina cuáles son los mensajes que la aplicación puede recibir o enviar al sistema.

El fichero de manifiesto configura datos importantes para la aplicación, como por ejemplo los permisos necesarios para ejecutarla o las bibliotecas que necesita.

De momento, no está documentada ninguna forma (bien sea a través de la API o de la explotación de vulnerabilidades) que permita la modificación de los datos contenidos en el fichero `AndroidManifest.xml` en tiempo de ejecución, ya que estos se determinan en tiempo de instalación de la aplicación.

4.4.2 Los binarios de Android

Los archivos binarios dentro de los paquetes de Android se encuentran el los ficheros con extensión `.dex`. A continuación vamos a describir en profundidad el contenido y la estructura de estos ficheros [Pro11].

En la figura 4.5 se puede ver el esquema general de cómo se estructuran estos ficheros. Esta estructura está compuesta de los siguientes elementos:

- *Cabecera (header)*, la cual está compuesta por una serie de comprobaciones (*checksum*), así como desplazamientos a otras partes (*offset*).

- *Lista de identificadores de strings (strings_ids)*. En esta tabla se almacena la longitud y los desplazamientos para cada cadena en el archivo `.dex`, como pueden ser constantes de cadena, los nombres genéricos o nombres de variables, entre otros.

- *Lista de identficadores de tipos (types_ids)*, que incluye todos aquellos tipos (clases, arrays o tipos primitivos) que son referenciados por el fichero `.dex`. Esta lista debe ser ordenada por el `string_id`.

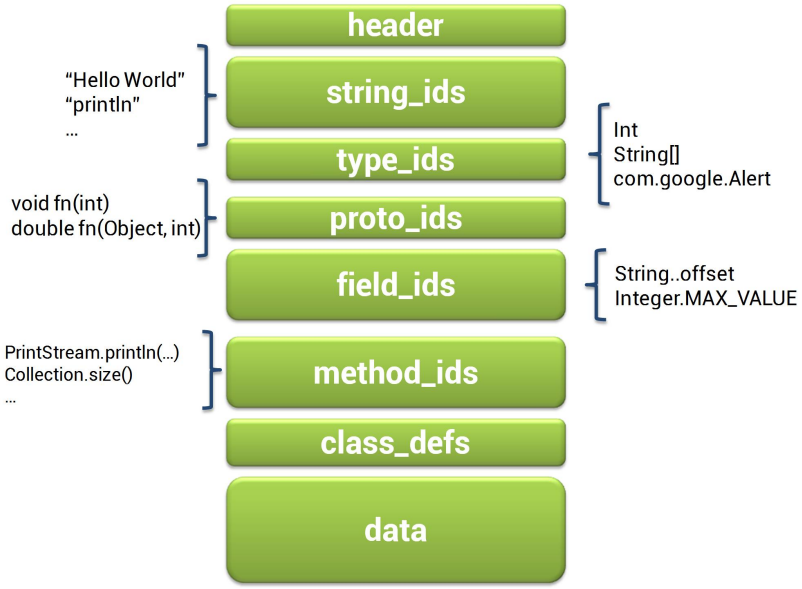


Figura 4.5: Arquitectura de un fichero binario en Android.

- *Lista de identificadores de prototipos (prot_ids)*, que identifica todos los prototipos de funciones a los que se refiere el fichero. Están ordenados por identificador de tipo de retorno y por identificador de argumentos (ambos alojados en la lista *type_id*).

- *Lista de identificadores de campos (field_id)*, los cuales están ordenados por el tipo de definición, posteriormente por el nombré del método (obtenido de la lista de *strings*).

- *Lista de identificadores de métodos (method_id)*, ordenados por el tipo, el nombre del método y el prototipo del método.

El binario está organizado de tal forma que optimice los escasos recursos con los que cuenta el dispositivo.

- *Lista de definiciones de clases (class_defs)*, las cuales deben estar ordenadas de tal manera que, dadas las interfaces que implementan, tanto él como su clase padre, aparezcan en la lista antes que clase referida. Por otra parte, no es válido que una definición de la clase con el mismo nombre aparezca más de una vez en la lista.

- *Área de datos (data)*, la cual contiene todos los datos de soporte para las tablas anteriores.

- *Datos Linkados (link_data)*, que son los datos utilizados en los archivos enlazados estáticamente.

Códigos operacionales

Si nos atenemos a la definición de Wikipedia[7], se define código operacional u «opcode» como una porción de una instrucción de lenguaje máquina que especifica la operación realizada. Estos códigos operacionales también pueden encontrarse en lo que se denominan *bytecode*. Este bytcode se trata de un código intermedio más abstracto que el código máquina, el cual es interpretado por una máquina virtual que convierte este código intermedio en código binario. Esta solución se suele adoptar para poder aislar el código binario de la implementación del *hardware*, dejando este trabajo a la máquina virtual.

El código fuente se convierte en un código intermedio denominado bytecode.

El caso más conocido, y en el que se basa el código intermedio de Dalvik, es Java. De hecho, en el proceso de compilación del código fuente a *bytecode* a Android, se convierte a bytecode de Java para, posteriormente, por medio de la herramienta *dx*, se genera el binario *bytecode* de Dalvik [Ehr10].

Sin embargo, existen varias diferencias entre los *bytecodes* de los dos sistemas. La más importante es que mientras el *bytecode* de Java está basado en pila (*stack-based*), Dalvik está basado en registro (*register-based*). Tradicionalmente, las máquinas virtuales de este tipo de sistemas se han diseñado con una arquitectura basada en pila, fundamentalmente por 3 motivos: sencillez en la implementación, facilidad para escribir el compilador *back-end* (esto es, la parte que genera el código máquina, específico de una plataforma, a partir de los resultados de la fase de análisis[8]) y densidad de código (es decir, los ejecutables de este tipo de arquitecturas suelen ser menores que los basados en registros).

Dalvik está basado en registro, en vez de estar basado en pila.

Sin embargo, la estructura de los binarios de Android, detallada en el punto 4.4.2, permite superar estas limitaciones. Alguna de las características de los códigos operacionales de Android son [Pro07]:

- Los registros son de 32 bits de ancho. Para los valores de 64 bits se usan pares de registros adyacentes.
- La unidad de almacenamiento en el flujo de instrucciones de 16 bits sin signo. Algunos bits de algunas instrucciones se ignoran o pueden ser 0.
- Las instrucciones no están limitadas a un tipo determinado (p.ej., las instrucciones que se mueven de valores de regis-

7 http://en.wikipedia.org/wiki/Opcode. Fecha de último acceso: 23-07-2012

8 http://es.wikipedia.org/wiki/Compilador Fecha de último acceso: 23-07-2012

tro de 32-bit sin interpretación no tiene que especificar si se están moviendo enteros o flotantes).

- Algunos *opcodes* tienen un sufijo de desambiguación, separado de los nombres principales por el carácter «/», para distinguir entre operaciones que son idénticas pero que mantienen distintos diseños de instrucción o distintas opciones. Con ello se busca que exista una correspondencia de uno a uno con las constantes estáticas en el código que genera e interpretan los ejecutables (es decir, para reducir la ambigüedad para los humanos que lean el código).

Para desambiguar los distintos códigos operacionales, se utiliza un sufijo separado por el carácter «/».

Todas esta son sólo algunas de las características que tienen los códigos operacionales de Dalvik. En general, podemos ver que el objetivo al utilizar este tipo de arquitectura es el de optimizar al máximo los escasos recursos de los dispositivos en los que se ejecutan. Por ello, se divide el ejecutable para maximizar las referencias y minimizar la duplicidad dentro de los mismos.

4.5 DESARROLLO DE APLICACIONES

A la hora de plantear la seguridad de la plataforma, uno de los elementos que tuvieron en cuenta es, como se comenta en la sección 4.1, que los desarrolladores no entienden de seguridad. Esta premisa fue clave a la hora de plantear el modelo de desarrollo de la plataforma.

A continuación vamos a ver algunos de los aspectos más relevantes desde el punto de vista de la seguridad a la hora de desarrollar aplicaciones, debido a que las aplicaciones son uno de los puntos en los que los terminales son más vulnerables (sección 3.4).

4.5.1 Control de la seguridad en aplicaciones

Son dos los elementos que permiten este control.

La API de Android consta de 8.648 métodos.

El primero de ellos son los permisos. Éstos son necesarios para interactuar con la interfaz de programación de la aplicación (de la locución inglesa Application Programming Interface, *API*) del sistema, los datos y el sistema de paso de mensajes.

La API del sistema [Tea11],describe 8.648 métodos distintos, de los cuales sólo algunos de ellos necesitan permisos específicos para poder acceder a ellos. De la misma forma, para acceder a los

lugares donde se almacenan los datos del sistema, denominados `Content providers`, también requieren de permisos específicos para poder acceder a ellos (p.ej., para poder acceder a los contactos almacenados es necesario poseer el permiso `READ_CONTACTS`).

Estos vienen definidos por un fichero de configuración, denominado `AndroidManifest.xml`, del cual se puede encontrar más información en el sección 4.4.1.

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

Esta máquina virtual permite la ejecución de las aplicaciones en un entorno controlado, lo que favorece, desde el diseño del sistema, el control sobre las aplicaciones.

Con estas dos medidas, desde el diseño, se controla la ejecución de las aplicaciones desde la fase de instalación para conformar una arquitectura segura de la plataforma.

4.5.2 Datos compartidos entre aplicaciones

Una de las características que posee Android es la capacidad de las aplicaciones de este sistema para compartir datos entre ellas. Esta características se consigue a través de los `Content Providers`.

La compartición de los datos se hace a través de los Content Providers.

En Android no existe un lugar centralizado en donde se almacenan todos los datos. Son los `Content Providers` los que se encargan de la gestión de los mismos en cada aplicación, por lo que es el desarrollador el encargado de la gestión que hacen tanto su aplicación como el resto de las aplicaciones.

4.6 INSTALACIÓN DE APLICACIONES

Dentro de la plataforma Android, a la hora de instalar aplicaciones, existen distintos aspectos a tener en cuenta desde el punto de vista de la seguridad. En esta sección vamos a ver cuáles son. En primer lugar veremos las tiendas de aplicaciones, ya que son la manera más habitual de instalar aplicaciones en la plataforma. Analizaremos los distintos tipos de tiendas de aplicaciones que hay y sus connotaciones respecto a la seguridad.

Posteriormente, diseccionaremos los permisos de las aplicaciones. Los permisos son la herramienta que define, en primera instancia, la seguridad de la aplicación, así como los recursos a los que es necesario que acceda. En este apartado, haremos un exhaustivo estudio sobre los distintos permisos de la plataforma Android.

4.6.1 Tiendas de aplicaciones

Además de la tienda oficial de Android, existen otras tiendas alternativas desde donde se pueden instalar aplicaciones.

La manera habitual de obtener aplicaciones para los usuarios son las tiendas de aplicaciones. Concretamente se tratan de repositorios en las que los desarrolladores depositan sus aplicaciones, normalmente a cambio de una parte de los ingresos obtenidos por la venta de las mismas. Por ello, son uno de los posibles vectores de ataque a los *smartphones*, como hemos identificado en la sección 3.3.2.

Dentro del sistema Android, no existe una única tienda de aplicaciones. Pese a que existe la tienda oficial, es posible instalar aplicaciones directamente descargándose el fichero APK. Gracias a ello se han desarrollado tiendas de aplicaciones alternativas, como la de Amazon[9] o GetJar[10].

Esta diversidad, desde el punto de vista de la seguridad, presenta un problema. Cada una de las tiendas de aplicaciones sigue una política distinta en cuanto a la seguridad. Por ejemplo, la tienda de Amazon, hace una evaluación de las aplicaciones antes de que la aplicación sea publicada. Sin embargo, otras tiendas, como el GetJar no realizan comprobación ninguna.

Esta situación provoca que aplicaciones maliciosas sean subidas a tiendas de aplicaciones, potenciando su difusión. Por otro lado, las distintas políticas hacen que los usuarios no tengan claro el proceso de subida de aplicación, y no prestan la atención necesaria la descarga de las mismas.

Sin embargo, estas tiendas de aplicaciones poseen un repositorio importante de información relacionada con las aplicaciones, tanto objetiva (p.ej., tamaño instalada o permisos que requiere) como subjetiva (p.ej., valoración de los usuarios).

Las distintas maneras de distribuir aplicaciones son claves a la hora de evaluar la seguridad de un sistema. Las tiendas de aplicaciones son el mejor medio de distribución de aplicaciones que tiene la plataforma y poseen una gran cantidad de información.

9 http://www.amazon.com/mobile-apps/b?ie=UTF8&node=2350149011
10 http://m.getjar.com/

4.6.2 Permisos de las aplicaciones

Como hemos comentado con anterioridad, el sistema operativo Android otorga un usuario del sistema (denominado `UID`) a cada una de las aplicaciones que se instala. Estos permisos son otorgados tanto a los ejecutables como a los recursos de los que hace uso la aplicación. Desde este punto de partida, se otorgan los permisos necesarios a las aplicaciones para su ejecución.

Los permisos de las aplicaciones pueden determinarse en tiempo de ejecución o en tiempo de instalación.

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

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

El sistema operativo Android se caracteriza por un sistema de permisos en tiempo de instalación. Este tipo otorga una serie de ventajas [FGW11]:

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

Android posee 4 niveles de permisos: normal, peligroso, firmado y firma-o-sistema.

Los permisos de Android poseen además dos características adicionales importantes. La primera de ellas es que los permisos de instalación están clasificados en cuatro niveles: *normal*, *peligroso*, *firmado*, *firma-o-sistema*. Los *permisos normales* son aceptados por defecto. Estos permisos son otorgados para proveer al sistema de defensa en profundidad y revisión de triaje. Los *permisos peligrosos* son aquellos que involucran algún aspecto relacionado con la seguridad y la privacidad del terminal. Los *permisos*

de firma permiten a los desarrolladores controlar los permisos que se proveen a otras aplicaciones a través de las interfaces exportadas. Para ello, deben estar firmados con la misma firma del desarrollador. Por último, los permisos que se encuentran en la categoría *firma-o-sistema* son aquellos que están firmadas con la clave de firmware. Este grupo pretende evitar que aplicaciones de terceros haga uso de las funciones del sistema. De todos ellos, únicamente los permisos que se solicitan dentro del nivel peligroso se presentan al usuario.

La otra característica importante de los permisos en Android es que ofrece la posibilidad de delegar permisos. Gracias a ello, una aplicación de correo electrónico que, por ejemplo, puede otorgar permisos a un reproductor multimedia para reproducir un vídeo adjuntado en un correo. Sin embargo, el reproductor no tiene acceso a todos los adjuntos del cliente de correo.

4.7 EJECUCIÓN DE APLICACIONES EN ANDROID

Las aplicaciones de Android se ejecutan en una instancia de la máquina virtual Dalvik.

En esta sección vamos a ver distintos elementos que entran en juego a la hora de ejecutar una aplicación en el sistema Android. Analizaremos tanto aspectos técnicos del sistema, como por ejemplo la máquina virtual *Dalvik* y la gestión de memoria del sistema. Y el modelo de ciclo de vida que tienen las aplicaciones dentro del sistema. Esto nos permitirá tener una visión más clara de cómo el diseño del sistema puede condicionar distintos aspectos de la seguridad de la plataforma.

4.7.1 La máquina virtual Dalvik

La máquina virtual *Dalvik* es el entorno en el que se ejecutan las aplicaciones. Las limitaciones técnicas que tiene son principalmente tres. La primera es que debe correr en una CPU lenta. Si bien es cierto que los procesadores móviles hoy en día han avanzado, la capacidad del procesador de un terminal móvil dista mucho de la del procesador de un PC convencional.

A partir de la versión 2.2 de Android se añadió un compilador JIT.

Por otro lado, posee relativamente poca memoria RAM, lo que obliga a la optimización del uso de la misma. Esta limitación se solventa con tres preceptos: repetición mínima, tipado explícito de las variables y etiquetado implícito. Mas información sobre la gestión de la memoria se puede ver en el punto 4.7.3.

Por último, se ejecuta con una limitación más importante que todas las anteriores: se alimenta de una batería. Esto hace que las optimizaciones en cuanto a consumo y eficiencia sean críticas para garantizar la mayor autonomía posible en el dispositivo.

A partir de la versión 2.2 del sistema se añadió un compilador JIT (*Just In Time*). Este tipo de compiladores buscan generar el código binario en tiempo de ejecución, buscando, de esta forma, mejorar el rendimiento. El modelo para la implementación por el que se optó es el de *Trace-granularity JIT*, principalmente porque minimiza el uso de memoria, aspecto crítico en los dispositivos móviles.

Google publica periódicamente información sobre *Dalvik* en su conferencia anual, siendo los más interesantes en esta materia los de 2008[11] y 2011[12]

4.7.2 El ciclo de vida de las aplicaciones

La `Activities` es el elemento base de una aplicación en Android. El ciclo de vida de éstas se representa con la máquina de estados finitos que está representada en la figura 4.6. En el podemos ver las distintas transiciones de estados que realiza el sistema operativo y, más concretamente, el `Activity Manager`. Estas transiciones se realizan en función del código de la aplicación, de los recursos disponibles y de las órdenes dadas por el usuario. De esta manera, es el sistema el que se encarga de priorizar las distintas actividades que se producen en el sistema (p. ej., mostrar una llamada entrante en vez de consultar el correo).

Los distintos estados o *callbacks* en los que puede encontrase una aplicación son los siguientes:

Una aplicación puede encontrarse en 4 estados distintos: activo, pausado, detenido o terminado.

- Activo: la actividad se encuentra en primer plano y ejecutándose.
- Pausado:el usuario puede ver la actividad y se está ejecutando, pero no puede interactuar con ella. Esto normalmente se produce porque hay otro elemento encima (por ejemplo una notificación).
- Detenido: la actividad se está ejecutando pero hay otras actividades que se están mostrando. En este estado, la acti-

11 https://sites.google.com/site/io/dalvik-vm-internals
12 http://www.google.com/events/io/2010/sessions/jit-compiler-androids-dalvik-vm.html

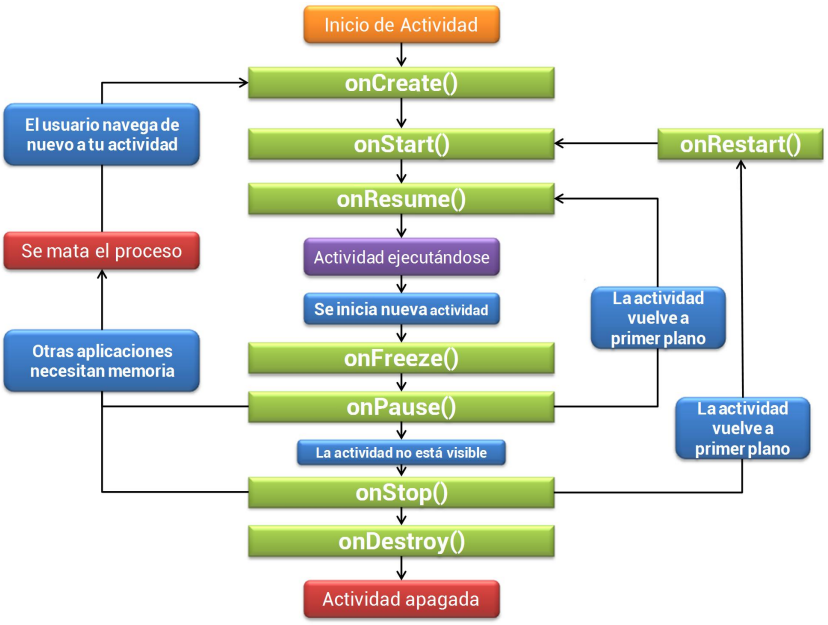


Figura 4.6: Ciclo de vida de una aplicación dentro del sistema operativo Android.

vidad sólo puede mostrar información al usuario a través de los sistemas de notificaciones.

- Terminada: la actividad está en este estado bien porque nunca se inició (p.ej., al reiniciar el teléfono), o porque el sistema la finalizó por falta de recursos.

Para la gestión del ciclo de vida de las aplicaciones, los desarrolladores disponen de varios métodos.

Para gestionar este ciclo de vida, los desarrolladores deben controlarlo a través de varios métodos:

- `onCreate()`: se lanza al ser llamada la `activity` por primera vez. Se realiza la inicialización de los datos, creación de las vistas, etc.
- `onRestart()`: se llama cuando la actividad ha sido parada y vuelve a utilizarse. Se ejecuta justo antes de lanzar `onStart()`.
- `onStart()`: se lanza nada más mostrar la pantalla al usuario.
- `onResume()`: se ejecuta cuando la aplicación ha terminado de cargarse en el dispositivo y el usuario comienza a interactuar con ella.

- `onPause()`: se llama cuando el sistema necesita los recursos porque arranca una nueva `activity`.
- `onStop()`: se ejecuta cuando la `activity` ya no es visible para el usuario.
- `onDestroy()`: Llamada final de la `activity`. Tras esto, se vuelve a la llamada inicial.

El desarrollador puede controlar la ejecución de su aplicación a través de funciones cuya firma está definida por las interfaces.

Todas estas funciones conforman el núcleo de funciones que el desarrollador debe conocer para adecuar el comportamiento de la aplicación al sistema operativo mejorando, de esta forma, tanto la experiencia del usuario como el rendimiento que se puede obtener del terminal.

Para decidir que aplicaciones deben terminarse, Android determina las prioridades de los procesos en función de su estado. Así, la clasificación se hace en torno a tres categorías distintas: prioridad crítica, alta y baja.

Dentro de la categoría de aplicaciones con prioridad crítica se encuentran todos aquellos procesos activos. Éstos se refieren a aquellos componentes con los cuales el usuario está interactuando. Acaparan una mayor cantidad de recursos, aunque tienden a ser pocos procesos. Normalmente, los procesos activos incluyen:

- `Activities`, `Services`, o `Broadcast Receivers` que están ejecutando `onRecieve`, que es el método maneja la gestión que se produce en el sistema de los eventos.
- `Activities` en estado «activo», es decir, que están en primer plano interactuando con el usuario.
- Servicios manejadores de eventos que están ejecutando métodos críticos, como son `onStart`, `onCreate` o `onDestroy`.

La gestión de los procesos se hace de forma dinámica, categorizando las aplicaciones entre otras cosas, en función de su estado y su última utilización.

Por otro lado, los procesos con alta prioridad se pueden clasificar en dos grupos: procesos visibles y procesos de servicio iniciados. Los procesos visibles son aquellos que están visibles, pero no se encuentran en primer plano de proceso. Son muy pocos procesos y únicamente son eliminados en circunstancias extremas. Los procesos que tienen un servicio iniciado son aquellos que no tienen ningún elemento visible, ejecutando procesos en marcha que deben lanzarse sin interfaz visible. Estos procesos no serán eliminados a menos que los anteriores necesiten recursos para ser ejecutados.

Por último, los procesos con prioridad baja se dividen a su vez en dos tipos de procesos distintos: los procesos en segundo

plano o *background* y procesos vacíos. Los primeros son todos aquellos procesos que albergan `Activities` que se han iniciado pero no se engloban dentro de la categoría de servicios de *backgroud*. Se trata de una larga lista de procesos que han sido ejecutados y que son eliminados en base a una pila que se ordena en función del tiempo que hace que no es visible, es decir, el que hace más tiempo que se ha mostrado al usuario es el primero en eliminarse. Finalmente, los procesos vacíos son aquellos procesos que se mantienen en memoria a pesar de haber finalizado su tiempo de vida. Esto se hace para mejorar el rendimiento general del sistema. Android lo mantiene en memoria para mejorar el tiempo de arranque de la aplicación en caso de que vuelva a ser lanzada. Obviamente, son las primeras aplicaciones que se eliminan en caso de necesidad de recursos.

4.7.3 Gestión de memoria

Android utliza un sistema de memoria virtual con paginación para la gestión de la memoria.

El modelo de una máquina virtual bajo un único proceso para cada una de las aplicaciones aporta robustez desde el punto de vista de la seguridad. Es la máquina virtual *Dalvik* la que se encarga de gestionar la memoria interactuando con el núcleo, que es el que se encarga de la coordinación de las mismas. Android utiliza un sistema de memoria virtual con paginación para la gestión de la memoria del dispositivo. La conversión de las direcciones lógicas a las direcciones físicas se hace con el módulo *Memory Manager Unit* (MMU)

Sin embargo, este modelo de un proceso y una única máquina virtual por aplicación genera, entre otros, un problema de ineficiencia importante: cada uno de estos procesos debe cargar en memoria todas aquellas clases estándar de Java que hacen falta para la ejecución de la aplicación. Esto provoca que el tiempo de arranque de cada una de las aplicaciones sea muy grande. Para solucionarlo utilizaron un recurso denominado *zyogte*.

Linux cuenta con un mecanismo de copia denominado *copy-on-write* (COW). Este mecanismo permite que los procesos hijos (creados a través de la instrucción `fork()`), compartan parte de la memoria con el proceso padre. Cuando se crea el proceso hijo, este utiliza la memoria y las tablas de páginas (es decir, tablas que son usadas por el sistema operativo para realizar las traducciones de direcciones de memoria virtual (o lógica) a memoria real (o física) del proceso padre. Pero estas páginas son marcadas como de sólo lectura, de tal manera que si algún proceso hijo intenta escribir en las tablas del proceso padre, se produce

una interrupción por error (denominada *trap*). En este caso, el sistema operativo se encarga de crear una copia de la página y redirige la tabla de páginas del proceso a la nueva copia creada y, de esta manera se continúa la ejecución de ambos procesos.

Aprovechándose de este mecanismo, Android genera un proceso inicial, denominado *zygote*, el cuál carga la máquina virtual *Dalvik*, las bibliotecas más utilizadas de Java y la API de Android, a partir del cual todos los procesos del sistema son generados. Por ello, todas las aplicaciones de Android comparten en memoria el código que no se necesita modificar la máquina virtual, así como las bibliotecas del sistema. De esta forma, se consiguen dos cosas: la primera, la creación de procesos hijo en Android es muy rápida, ya que gran parte de las bibliotecas necesarias ya están cargadas en memoria. La segunda, con este modelo de creación de procesos se produce un ahorro considerable de memoria.

El proceso zygote permite que todas las aplicaciones compartan en memoria las bibliotecas más utilizadas.

Sin embargo, desde el punto de vista de la seguridad, el proceso *zygote* se convierte en objetivo de ataques, ya que modificando las o que carga se tiene un gran control sobre el sistema.

4.8 COMUNICACIÓN DENTRO DE LA PLATAFORMA

Android utiliza intensivamente un sistema de comunicación denominado *Intents*. Este sistema permite la comunicación tanto entre las aplicaciones como dentro de la propia aplicación. Las aplicaciones controlan los intents a través de los permisos, los cuales son puestos por las aplicaciones dentro de los propios *Intents* [EOM09]. Gracias a estos permisos, las aplicaciones pueden controlar quién recibe los *Intents*. Este mismo mecanismo es el utilizado por el sistema operativo para gestionar estos mensajes.

4.9 SUMARIO

En este capítulo hemos visto las particularidades del sistema operativo Android, desde su árbol de directorios hasta la instalación de las aplicaciones, todo ello desde el punto de vista de la seguridad. También hemos hecho especial hincapié en uno de los elementos más importantes desde el punto de vista de la seguridad: los permisos de las aplicaciones.

En general, podemos decir que Android está diseñado teniendo en cuenta la seguridad desde el principio del proceso. Sin embargo, su carácter abierto dificulta la seguridad del mismo, ya que es más fácil encontrar vectores de ataque.

Otra parte importante del diseño del sistema son las optimizaciones llevadas a cabo. Dada la naturaleza de los dispositivos en los que se ejecuta, la utilización de técnicas para optimizar el uso de los recursos limitados de los que dispone supone una parte importante del diseño del mismo. Sin embargo, estas optimizaciones pueden exponer nuevos vectores de ataque al sistema.

5 ANÁLISIS ESTÁTICO DE APLICACIONES EN ANDROID

«Lo que un hombre puede inventar, otro lo puede descubrir»

Sherlock Holmes, creado por
Sir Arthur Conan Doyle
(1859 – 1930)

ÍNDICE

Este capítulo se centrará en el desarrollo de distintas técnicas para la detección de *malware* sobre la plataforma Android. En primer lugar, analizaremos distintos aspectos relacionados con la detección estática de *malware* para, seguidamente, hacer un breve repaso a las técnicas de aprendizaje automático que se utilizarán más adelante para la clasificación de las muestras.

Posteriormente, definiremos el proceso de creación del conjunto de datos que permitirá validar el modelo presentado. Tras ello, iniciaremos el estudio del análisis estático de las aplicaciones. Utilizaremos distintas características extraídas de las mismas para que, en combinación con los distintos modelos de aprendizaje automático, permitan la detección de comportamientos maliciosos. Finalmente, estudiaremos los resultados obtenidos y realizaremos la valoración correspondiente de los mismos.

5.1 DETECCIÓN DE SOFTWARE MALICIOSO

En este capítulo veremos distintas técnicas que tienen como objetivo desarrollar un modelo que permita, ante una nueva muestra, determinar si una aplicación es maliciosa o no.

La detección de *malware* presenta un problema análogo al principio de indecibilidad de Turing [Lin11]. Según este principio, no es posible escribir un programa que decida si otro programa cualquiera está correctamente escrito. En la misma línea, no es posible desarrollar un programa que determine si un programa es *malware* o no, como demostraron Chew y White [CW00].

Sin embargo, trabajos anteriores han demostrado que es posible ofrecer soluciones que, si bien no serán jamás perfectas, ofrecen un resultado adecuado para el día a día.

Primero veremos el modelo de detección de anomalías desarrollado, y luego aplicaremos algoritmos de aprendizaje automático sobre los parámetros de predicción elegidos.

En este capítulo desarrollaremos dos técnicas distintas para la detección de *malware* de manera estática. En primer lugar, veremos algunos aspectos relacionados con los algoritmos de aprendizaje automático que se utilizarán más adelante. A continuación, desarrollaremos un sistema para la detección de software malicioso basado en detección de anomalías. Finalmente, analizaremos nuestra aportación a la detección de aplicaciones maliciosas a través de la aplicación de modelos de aprendizaje automático.

5.2 CONJUNTO DE DATOS

En este apartado se detallará la forma en la que se ha seleccionado el conjunto de datos que se utilizarán durante la experimentación de este capítulo. Las características que se consideraba inicialmente que debía cumplir este conjunto de datos son las siguientes:

- *Debe ser heterogéneo*. Debe reflejar la diversidad de tipos de aplicaciones que se encuentran disponibles.
- *Debe ser proporcional* al número de muestras que existen de cada tipo.

Para la creación del conjunto de datos se ha buscado la heterogenidad y la proporcionalidad.

Para ello, se han creado dos conjuntos de datos. El primero engloba todas las muestras que han sido clasificadas como software benigno o *goodware*. El segundo recoge aquellas muestras catalogadas como software malicioso o *malware*.

La creación de estos conjuntos de datos se detalla a continuación.

5.2.1 Conjunto de datos de software benigno

Para conformar este conjunto de datos se obtuvo una colección de 1811 muestras de aplicaciones de Android de diversos tipos. A fin de clasificarlas de forma adecuada, se optó por seguir la misma clasificación que se da en la tienda oficial de Android: el *Android Market*. Para ello, gracias a una librería no oficial desarrollada en Java, denominada android-market-api[1], obtuvimos la clasificación.

Una vez clasificadas todas las muestras, se procedió a la selección del subconjunto de muestras que conforma el conjunto de datos de *goodware* final. La metodología utilizada para la obtención de estas muestras fue la siguiente:

El dataset de goodware está compuesto por 333 muestras.

1. *Determinar el número de muestras totales*. Para facilitar la creación de modelos, es preferible que el número de muestras de las dos clasificaciones posibles este balanceada. Por ello, y dado que el número de muestras de *malware* disponibles es inferior a las de *goodware*, se optó por limitar el número de muestras de *goodware* a 333.
2. *Determinar el número de muestras por categoría*. Para poder reflejar la distribución estadística existente dentro del Android Market, se seleccionó el número correspondiente de muestras por categoría, de tal forma que se mantenga la proporcionalidad en cada una de las categorías.
3. *Tipos de aplicación*. Dentro del ecosistema Android existen distintos tipos de aplicaciones: nativas (desarrolladas con el SDK), web (desarrolladas mayoritariamente con HTML,

1 http://code.google.com/p/android-market-api/

JavaScript y CSS) y widgets (aplicaciones sencillas para poner en el escritorio que se caracterizan por ser rápidas y con una funcionalidad concreta, desarrolladas mayoritariamente como las aplicaciones web). Todas estos tipos de aplicaciones poseen características distintas. A la hora de realizar este conjunto de datos, no se han hecho distinciones en el tipo de aplicaciones, por lo que todos ellos están representados en el conjunto de datos final.

4. *Selección de muestras por categoría*. Una vez determinado el número de muestras de cada categoría, se procedió a la selección de las mismas. Ésta se realizó seleccionando de forma aleatoria, a través de un muestreo de Montecarlo [Sha03], los ejemplares de la categoría, evitando distintas versiones de la misma aplicación.

Se seleccionaron muestras de goodware de distintas categorías para hacer representativo el conjunto de datos.

Siguiendo esta metodología, se obtuvo el conjunto de datos de muestras de *goodware*. El número de muestras obtenidas por cada una de las categorías queda reflejado en la Tabla 5.1.

Arcade y acción	32	Librerías y demos	2
Carreras	2	Libros y referencia	10
Casuales	10	Medicina	1
Comics	1	Multimedia y vídeo	23
Compras	3	Música y audio	12
Comunicación	20	Negocios	1
Deportes	5	Noticias y revistas	7
Educación	0	Personalización	6
Empresa	4	Productividad	27
Entretenimiento	16	Puzzles	16
Estilo de vida	4	Salud y bienestar	3
Finanzas	3	Sociedad	25
Fotografía	6	Tiempo	2
Herramientas	80	Transporte	2
Juegos de cartas y casino	2	Viajes y guías	8

Total: 333

Tabla 5.1: Número de muestras de *goodware* por categoría.

5.2.2 Conjunto de datos de software malicioso

Las muestras de malware se obtivieron de VirusTotal.

Para la elaboración del conjunto de datos de *malware*, las muestras se obtuvieron de la empresa VirusTotal.[2]. En concreto, se trata de un servicio de análisis de ficheros sospechosos y de URLs, desarrollada por la empresa Hispasec Sistemas[3], laboratorio especializado en el ámbito de la seguridad y tecnologías de la información.

La herramienta VirusTotal permite el análisis de los ficheros enviados a través de distintos motores de casa de antivirus. En concreto, realiza el análisis con más de 40 motores diferentes.

Dentro de la herramienta, poseen una serie de servicios denominados «VirusTotal Malware Intelligence Services», los cuales permiten a los investigadores realizar búsquedas sobre su base de datos de análisis realizados.

Metodología para la selección de malware

En un primer paso se seleccionaron todas aquellas muestras de la base de datos que sean del sistema operativo Android. Para ello, generamos una alerta en el sistema que identificaba todas aquellas muestras que pertenecían al sistema operativo Android y que al menos uno de los antivirus clasificaban como *malware*.

El siguiente paso fue la selección de las muestras. Para ello se ha seguido el siguiente proceso. Inicialmente, el sistema detectó 2.808 muestras que cumplían la regla generada para elegir las muestras.

Se llevo a cabo un proceso de selección de muestras del malware que tenía en cuenta la fiabilidad de los antivirus comerciales a la hora de clasificar las muestras.

Seguidamente se llevó a cabo la ponderación de los antivirus. El objetivo de este paso es el de evaluar los antivirus para determinar el grado de fiabilidad que tienen a la hora de clasificar muestras de *malware* sobre la plataforma Android. Para ello, asumimos que todas aquellas muestras que han sido detectadas por al menos un antivirus es considerada como *malware*. Posteriormente, se pasó a evaluar los antivirus. Evaluamos el ratio de detección de cada uno de los antivirus respecto al total de muestras. De manera formal,

$$a_w = \frac{n}{n_t} \tag{12}$$

donde n es el número de muestras identificadas por el motor y n_t el número total de muestras. Este peso de cada uno de los antivirus sirve para medir la importancia de la muestra. Para

2 http://www.virustotal.com
3 http://www.hispasec.com/

ello, evaluamos cada muestra en función del peso de los antivirus que la detectan. Así, no es lo mismo una muestra detectada por un antivirus con una gran capacidad de detección que la detectada por una antivirus con menores capacidades.

Para esta evaluación, se aplicó la siguiente métrica descrita en la ecuación 13. Siendo $a_i = (a_1, a_2, \ldots, a_n)$ el conjunto los pesos de antivirus que detectan la muestra,

$$m_w = \sum a_w(a) \quad \forall a \in A \tag{13}$$

donde $A = a_1, a_2, \ldots, a_n$. Por lo tanto, m_w evalúa la detección de la muestra en función de los antivirus que la detectan.

Se determinó un umbral o *threshold*, por debajo del cual una muestra no puede entrar a formar parte del conjunto de datos, para garantizar que las muestras que entran a formar parte del conjunto de datos sean lo suficientemente significativas. Este umbral se determinó en un valor de 0,1, lo que dejó el conjunto de muestras seleccionables en 1.202 muestras.

Por otro lado, determinamos que el resultado arrojado por los distintos antivirus sea expresamente de muestras de Android. Esto se hace en base al nombre que le otorga el motor antivirus a cada una de las muestras. Finalmente, eliminamos todas aquellas muestras cuya resultado era duplicado, es decir, que el mismo antivirus lo había clasificado como el mismo comportamiento malicioso.

El conjunto de muestras de malware está compuesto de 333 elementos.

Mediante esta metodología se han obtenido 333 muestras únicas de aplicaciones clasificadas como maliciosas por las casas de antivirus que han superado las limitaciones impuestas. Si nos atenemos al último informe de la empresa Lookout[4], especializada en seguridad en dispositivos móviles, éste representa el 75 % de las muestras de *malware* existentes a julio de 2011. Por ello, consideramos que el conjunto de datos es suficientemente representativo del estado actual del *malware* sobre Android, y válido para llevar a cabo los experimentos de la tesis.

5.3 PREDICTORES UTILIZADOS EN EL ANÁLISIS ESTÁTICO

En esta sección analizaremos cuáles son los predictores que se utilizarán a lo largo del capítulo para determinar si una mues-

4 https://www.mylookout.com/_downloads/lookout-mobile-threat-report-2011.pdf

tra es maliciosa o no. En concreto, se utilizarán tres atributos distintos. El primero de ellos es el conjunto de permisos que la aplicación requiere para ser instalada en el terminal. El segundo de ellos son las características. Esta información está dentro del fichero `AndroidManifest.xml` y ofrece información adicional sobre las características y capacidades de la aplicación. Finalmente, utilizaremos el conjunto de cadenas de caracteres que se encuentran dentro de la aplicación.

Utilizaremos los permisos de las aplicaciones, las características y las cadenas de caracteres.

5.3.1 Los permisos de las aplicaciones

En esta sección se muestra el estudio realizado a los permisos de las aplicaciones que componen el conjunto de datos, a fin de determinar su utilidad a la hora de clasificar una aplicación como maliciosa.

Para ello, inicialmente extrajimos los permisos que poseen cada una de las aplicaciones que componen este conjunto de datos. Esta labor se hizo mediante la herramienta «aapt», disponible dentro del conjunto de herramientas que provee el SDK de Android.

En la Figura 5.1 podemos observar los permisos más frecuentes en cada una de las dos categorías que conforman el conjunto de datos. Vemos que sí que existe una diferenciación entre los permisos que solicitan cada una de las aplicaciones. Lo primero que llama la atención es que las aplicaciones maliciosas tienden a pedir más permisos que las benignas (del orden de un 43 % más).

En ambos casos, podemos apreciar que `INTERNET` es uno de los permisos más requeridos por las aplicaciones de ambas categorías. Sin embargo, en las aplicaciones con fines maliciosos se hace especial hincapié en todos aquellos permisos relacionados con el envío y la recepción de mensajes de texto. Por otro lado, también destaca la gran diferencia entre el número de aplicaciones malignas que solicitan el permiso `INSTALL_PACKAGES` y el número de aplicaciones benignas que lo hace. Este permiso, combinado con la conexión a Internet del terminal, permite al atacante instalar cualquier tipo de aplicación. Otro tipo de permisos, destinados a recabar información del usuario, también tienen especial relevancia dentro del ámbito del software malicioso.

Los permisos determinan los recursos del terminal a los que la aplicación puede acceder.

Por otro lado, hemos realizado un análisis del número de permisos de cada aplicación, el cual se puede ver en la figura 5.2. En ella se puede apreciar que el número de permisos en cada

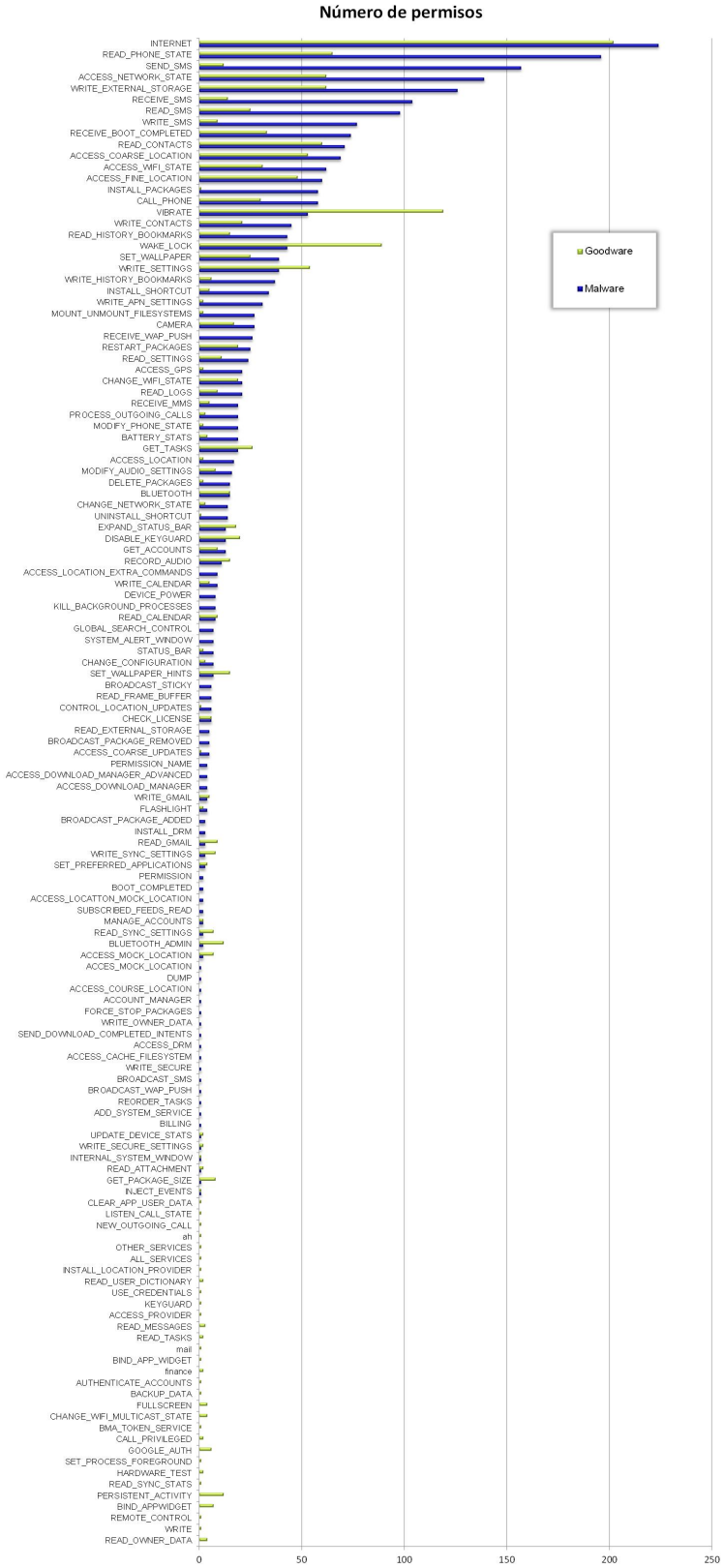


Figura 5.1: Permisos extraídos de las aplicaciones que conforman el conjunto de datos.

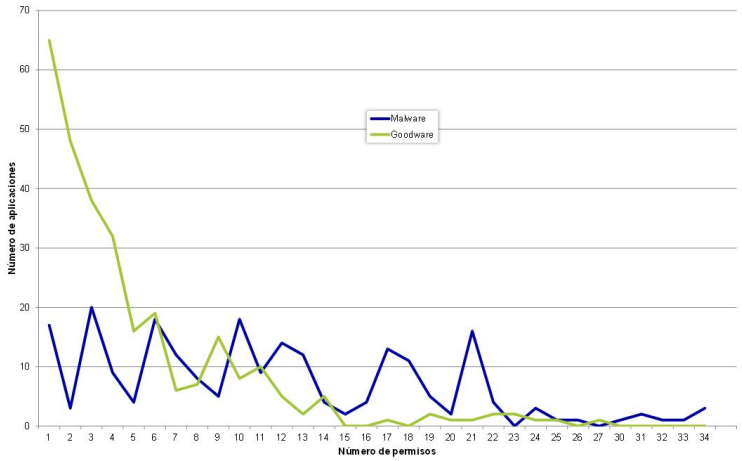


Figura 5.2: Número de permisos por aplicación.

una de las categorías del conjunto de datos es similar. Destaca el número de aplicaciones de *goodware* con 1 único permiso. Sin embargo, el número de aplicaciones benignas con 2 ó 3 permisos son superiores a las muestras de *malware*. Con estos datos podemos afirmar que el número de permisos que requiere una aplicación para tener un comportamiento malicioso no se diferencia del de una aplicación normal.

5.3.2 Características de la aplicación

Dentro del fichero `AndroidManifest.xml` además de los permisos, también se incluye otro tipo de información que puede ser relevante a la hora de determinar si una aplicación tiene intenciones maliciosas o no. Entre esta información, una de las más relevantes son las características.

Las características vienen determinadas dentro del manifiesto de la aplicación con el tag `<uses-feature />`, y determinan algunas características, tanto software como hardware, que son necesarias para la ejecución de la aplicación. Así, por ejemplo, el uso del bluethoot o de la cámara vienen determinados por las etiquetas `android.hardware.bluethoot` o `android.hardware.camera`.

Las características determinan algún elemento, tanto software como hardware, que son necesarias para la ejecución de la aplicación.

Este tipo de elementos dentro del manifiesto tienen un carácter meramente informativo. Si bien no afecta a la hora de determinar si puede ejecutarse o no dentro de un terminal, este tipo de información es usada por otro tipo de servicios o aplicaciones para mejorar la interacción entre las aplicaciones. Sin embargo,

esta falta de obligatoriedad hace que muchas aplicaciones no dispongan de esta información en su fichero de manifiesto.

En el conjunto de datos seleccionado, las características extraídas hacen referencia en su totalidad a características hardware, como pueden ser el uso de sensores de localización, bien sea mediante GPS o por las antenas de telefonía, o a los sensores de proximidad o luz.

A la luz de su naturaleza, hemos considerado esta información puede ser relevante a la hora de determinar si una aplicación tiene fines maliciosos o no, ya que ofrece un conocimiento extra sobre el comportamiento que puede tener a la hora de ejecutarse.

5.3.3 Las cadenas de caracteres

Una de las técnicas que más ampliamente se ha utilizado en la detección de *malware* ha sido la extracción de las cadenas de caracteres que contiene el ejecutable [SPDB09]. Esta técnica extrae todas las cadenas de caracteres que se encuentran en el ejecutable. Estas cadenas pueden contener, por ejemplo, opciones de los menús de la aplicación, o URLs a las direcciones a las que debe conectarse. De esta manera, mediante el análisis de estas cadenas de caracteres, es posible extraer información de las mismas, permitiendo determinar si una aplicación tiene intenciones maliciosas o no.

Las cadenas de caracteres se obtienen del opcode `const-string`.

El proceso que se ha seguido es el siguiente. La obtención de las cadenas de caracteres se hace procesando la aplicación decompliada. Para ello, se buscan los argumentos del código operacional `const-string`, que es aquel que define las cadenas de caracteres estáticas dentro de la aplicación. Posteriormente, eliminamos las palabras comunes dentro del leguaje o «stopwords»[WS96]. Para ello, obtenemos estas palabras de una fuente externa[5] tanto para el castellano como para el inglés. Convertimos la cadena de caracteres en un conjunto de atributos que representan las ocurrencias de las distintas palabras del texto que figura en las cadenas de caracteres. La separación de las palabras se realiza mediante el «tokenizer» es decir, los *tokens* que separan las palabras. Entre los tokenizadores más comunes se encuentran, entre otros, el espacio en blanco, el punto y coma o la coma. Esto permite crear un vector con estos atributos, a los cuales se les puede aplicar los modelos de aprendizaje automático que se han detallado en el punto 5.5.

5 http://paginaspersonales.deusto.es/claorden/es/resources.shtml

5.4 ANÁLISIS ESTÁTICO BASADO EN ANOMALÍAS

Como primera aproximación a la detección de *malware*, proponemos un sistema de detección de anomalías para detectar aquellas aplicaciones cuyos atributos se desvían de lo que podría considerarse como normales.

Se busca detectar aquellas aplicaciones cuyos atributos se desvían de lo que podrían considerarse como normales

Para ello, utilizaremos un método que hemos aplicado con éxito a otras áreas [LUPS+12]. Esta aproximación representa, en base a un conjunto de datos de entrenamiento, lo que se considera un comportamiento normal de una aplicación. Una vez definida este comportamiento normal, el método determina que, todo aquel que se desvía de esa normalidad es considerado anómalo y, por tanto, *malware*.

Para ello, se generan los vectores con cada uno de los predictores descritos en el apartado 5.3. De esta manera, se ha aplicado el método a los permisos, a las características, a las cadenas de caracteres, así como combinando los permisos con las características extraídas del `AndroidManifest.xml` y a las tres características combinadas.

Para medir la desviación entre los distintos vectores, hemos utilizado tres métricas distintas.

- *Distancia Manhattan*. La distancia Manhattan toma su nombre de las calles de la isla del mismo nombre, que dada su forma reticular, hace que esta métrica defina la distancia que hay que recorrer en el camino más corto para llegar entre dos puntos de la isla. Esta distancia se obtiene de la suma de las proyecciones del segmento de línea entre los puntos del sistema.

  Formalmente, dados dos puntos $P = (p_1, p_2, \ldots, p_n)$ y $Q = (q_1, q_2, \ldots, q_n)$ en un sistema de coordenadas cartesianas, como el espacio euclideano, la distancia Manhattan entre ambos puntos se define como:

$$d_M = \|P - Q\|_1 = \sum_{i=1}^{n} |p_i - q_i|^2 \tag{14}$$

- *Distancia euclidiana*. La distancia euclidiana, también conocida como distancia ordinaria, entre dos puntos del espacio euclídeano se deduce a partir del teorema de Pitagoras. Formalmente, dados dos puntos $P = (p_1, p_2, \ldots, p_n)$

y $Q = (q_1, q_2, \ldots, q_n)$, dentro de un espacio euclídiano n-dimensional, se define la distancia euclidiana entre ambos puntos como:

$$d_E = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2} \tag{15}$$

Se utilizan tres métricas para medir: la distancia, euclidiana, Manhattan y distancia del coseno.

- *Distancia del coseno*. Se basa en las propiedades que se dan dentro del espacio euclidiano, y se calcula a partir del coseno del ángulo que forman dos vectores en dicho espacio. Formalmente, dados dos puntos $P = (p_1, p_2, \ldots, p_n)$ y $Q = (q_1, q_2, \ldots, q_n)$, pertenecientes al espacio euclidiano n-dimensional, esta distancia se calcula de la siguiente forma:

$$d_C = \frac{\sum_{i=1}^{n} (p_i \cdot q_i)^2}{\sqrt{\sum_{i=1}^{n} p_i^2 \cdot \sum_{i=1}^{n} q_i^2}} \tag{16}$$

Todas estas medidas nos permiten calcular las desviaciones que se producen entre dos distintas representaciones de aplicaciones. Sin embargo, es necesario combinar esta representación con las aplicaciones que se encuentran en el conjunto de datos etiquetado como goodware. Es por ello que se hace imprescindible una métrica de combinación para obtener un valor de la distancia final que considere todas las medidas realizadas. A tal efecto, nuestro método contempla 3 métricas distintas:

- *Valor medio*: se calcula a partir del valor medio de cada distancia en el conjunto de datos de entrenamiento.
- *Valor mínimo*: se calcula el menor valor de distancia de cada distancia al conjunto de datos de entrenamiento.
- *Valor máximo*: se calcula el mayor valor de distancia de cada distancia al conjunto de datos de entrenamiento.

Por ello, nuestro método está compuesto por la combinación de la métrica de distancia que se utilice y métrica de combinación de la distancia que se elija respecto al conjunto de datos de entrenamiento. A continuación, detallaremos el método propuesto para la validación empírica del mismo.

Validación del método propuesto

Para la validación del método, hemos utilizado el conjunto de datos descrito en el apartado 5.2. Con los permisos, las características y las cadenas de caracteres, hemos generado un vector que permita su representación dentro de un espacio vectorial.

Para todos ellos, hemos calculado el peso que tienen cada uno de los atributos dentro de la clasificación. Esto lo hemos hecho a través del método de «Information Gain» [Ken83].

Para el experimento, hemos llevado a cabo los siguientes pasos. Primero, hemos creado las carpetas necesarias para realizar la validación cruzada [Koh95].

Para ello, hemos seleccionado el conjunto de datos de *goodware*, y lo hemos separado en 5 carpetas distintas. Para el conjunto de datos, de 333 elementos, los hemos separado en 2 carpetas de 66 elementos y otras 3 de 67 elementos.

Posteriormente, hemos extraído los predictores y les hemos aplicado las 3 medidas combinadas con los 3 métodos de combinación explicados previamente. Para cada una de las combinaciones posibles, hemos calculado 10 umbrales distintos para determinar si una aplicación es maliciosa. El umbral más bajo de los 10 es el mayor valor posible que hace que una aplicación maliciosa del conjunto de datos no sea clasificada correctamente. Por contra, el más alto de los umbrales elegidos para cada combinación es el menor valor que clasifica correctamente todas las aplicaciones maliciosas. El resto de umbrales se reparte equitativamente entre estos dos. Además, estos umbrales permiten configurar el sistema para reducir tanto el número de falsos positivos como el de falsos negativos.

La validación se realiza a través del método de la validación cruzada.

Los resultados obtenidos por este método se pueden ver en el apartado 5.6.3.

5.5 ANÁLISIS ESTÁTICO BASADO EN APRENDIZAJE AUTOMÁTICO SUPERVISADO

El aprendizaje automático [Bel06] es un área de investigación dentro de la inteligencia artificial. Esta rama busca el desarrollo y el diseño de algoritmos que permitan a los ordenadores generalizar comportamientos, utilizando como punto de partida información de entrada en forma de ejemplos.

El aprendizaje automático permite generalizar comportamientos, usando como punto de partida los ejemplos de entrada.

Tradicionalmente, estos algoritmos se agrupan en una taxonomía, en función de la salida de los mismos. Los algoritmos que

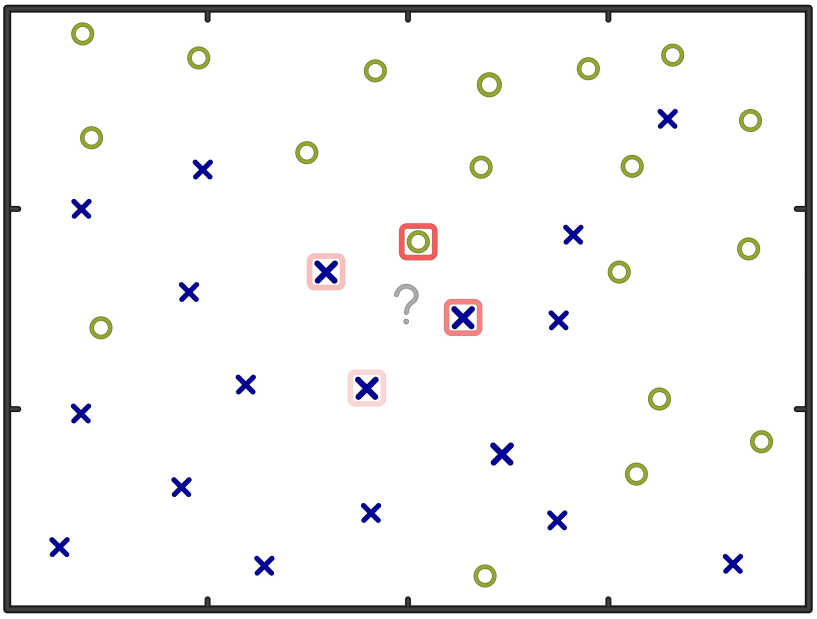


Figura 5.3: Ejemplo de clasificación mediante el algoritmo kNN.

se emplean en esta investigación pertenecen a la rama de aprendizaje supervisado [KZP07]. Este tipo de aprendizaje establece, en base a los ejemplos de entrada con los que se entrena, las correspondencias con la salida deseada, a fin de aplicarlas cuando se encuentre ante nuevos casos. La base de conocimiento está formada por los elementos de entrada convenientemente etiquetados para poder establecer las correspondencias. Esta fase es conocida como fase de entrenamiento.

A continuación detallaremos los algoritmos de aprendizaje automático que hemos aplicado posteriormente.

5.5.1 K-vecinos más cercanos

El algoritmo k-NN busca las muestras más cercanas dentro de su espacio de referencia.

El algoritmo de los k-vecinos más cercanos (k-*nearest neighbor algorithm* (k-NN)) [FHJ52] es un método de clasificación basado en la cercanía de las muestras de entrenamiento dentro del espacio muestral, como se puede ver en la figura 5.3.

Formalmente, en la fase de entrenamiento, sea un conjunto de datos $S = \{s_1, s_2, \ldots, s_m\}$, representado en un espacio n-dimensional donde n es el número de características de cada instancia.

En la fase de clasificación, al llegar una nueva instancia, para decidir la clase a la que pertenece, se mide la distancia entre

las instancias de entrenamiento y la nueva instancia. Para medir esta distancia, se puede utilizar métricas comunes, como por ejemplo la distancia euclidiana. Formalmente, dado un conjunto de atributos a, la distancia euclidiana entre dos puntos x e y se calcula de la siguiente manera (ver ecuación 15)

Finalmente, el método de clasificación que se suele aplicar es el de clasificar la instancia como la clase más común dentro de las k instancias más cercanas a la que se pretende clasificar.

La implementación utilizada para los cálculos es la que está presente en WEKA [HFH+09], denominada IBK. Esta implementación utiliza las distancias normalizadas para todos los atributos de manera que los atributos en diferentes escalas tienen el mismo impacto en la función de la distancia. La clasificación final se hace mediante un sistema de votos, de tal forma que se le atribuye la clase a la que pertenece la mayoría de las instancias de las k más cercanas.

La métrica de distancia que se utiliza es la euclidiana.

5.5.2 Árboles de decisión

Este tipo de clasificadores se puede representar en forma de árbol, de tal manera que los nodos interiores representan los estados posibles de las condiciones del problema y los nodos finales constituyen el resultado devuelto por el clasificador. De forma gráfica se puede representar como se ve en la figura 5.4.

De manera formal, podemos definir un árbol de decisión como un grafo $G = (V, E)$, siendo V un conjunto no vacío de nodos y E un conjunto de aristas. Definimos un «camino del árbol» como la secuencia de aristas que recorre, representado de la forma $(v_1, v_2), (v_2, v_3), \ldots (v_{n-1}, v_n)$.

Si tenemos en cuenta que (v, w) es una arista del árbol, se define a v como «nodo padre» de w, al cual se le se considera «nodo hijo». Dentro del árbol, el único nodo que no tiene nodo padre es el «nodo raíz», siendo «nodo interno» cualquier otro nodo interior del árbol y «hojas» los nodos finales del árbol.

Los nodos interiores representan los estados posibles de las condiciones del problema y los nodos finales constituyen el resultado devuelto por el clasificador.

En este trabajo hemos utilizado el algoritmo J48, que es la implementación presente en WEKA [HFH+09] del algoritmo C4.5 [Qui93], el cual es una extensión del algoritmo ID3 [Qui86]. Se basan en el concepto de *entropía de la información*, que se suele referir también como la entropía de Shannon [Sha51]. Específicamente, calcula la incertidumbre de una fuente de información, a través de la medición del valor esperado de la información dentro del mensaje.

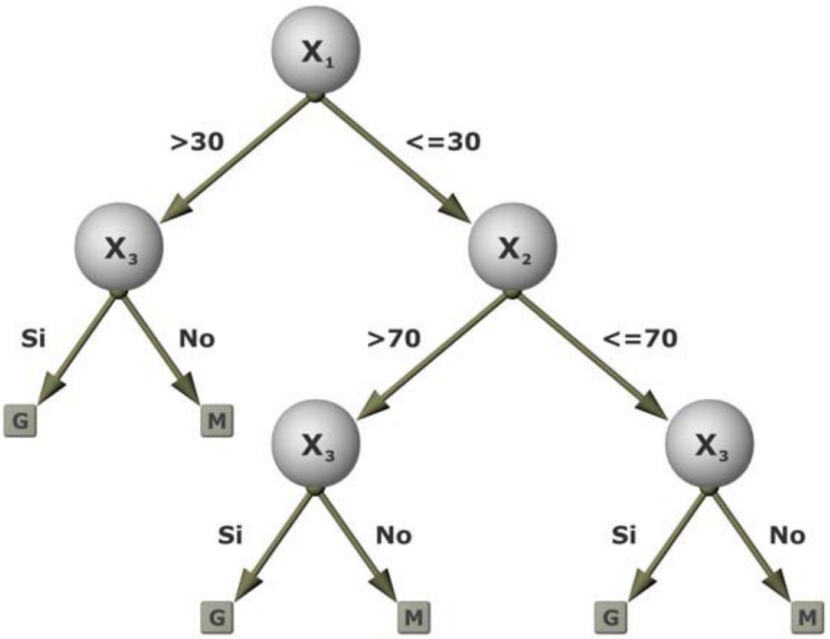


Figura 5.4: Ejemplo de clasificación mediante el algoritmo de árboles de decisión.

En este trabajo también se ha utilizado el método bosques aleatorios (en inglés, *random forests*) [Bre01] . Este método combina varios árboles de decisión. Con cada uno de estos árboles, se genera un vector de entrada. Este vector lo conforma un subconjunto del total de datos elegidos, utilizando una discriminación estocástica [Kle96]

Posteriormente, cada árbol genera una clasificación, eligiendo la clasificación definitiva mediante un sistema de votos.

La clasificación definitva se obtiene mediante un sistema de votos.

Este método produce clasificadores precisos, aun cuando tiene un gran número de variables de entrada. Sin embargo, tiende a sobreajustarse (es decir, se adapta muy bien al conjunto de entrenamiento pero pierde precisión con datos nuevos), especialmente con datos ruidosos (esto es, aquellos datos que son susceptibles de contener errores) [SLTW04].

5.5.3 Redes bayesaianas

En 1763, el reverendo Thomas Bayes desarrolló el teorema que lleva su nombre [BP63], y que se ha convertido en la base para un método estadístico que determina la probabilidad de que una hipótesis sea cierta en base a un número de observaciones. Este método, denominado inferencia bayesiana, ajusta las proba-

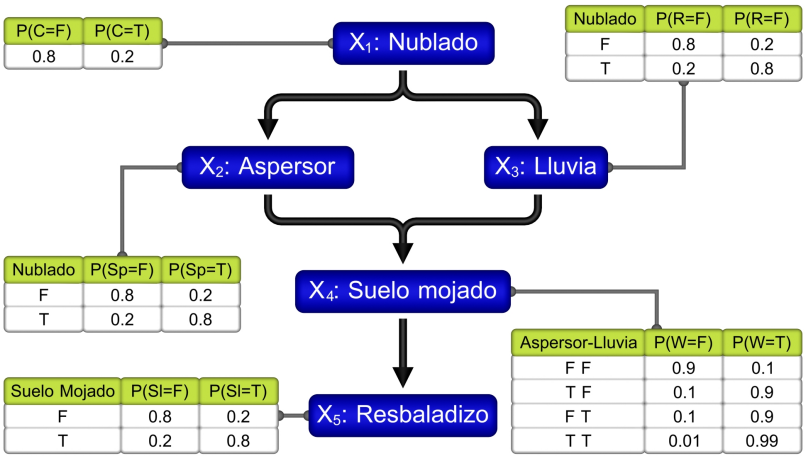


Figura 5.5: Ejemplo de clasificación mediante el algoritmo de redes bayesianas.

bilidades una vez existen nuevas observaciones. Según su formulación clásica, dados dos eventos A y B, la probabilidad condicionada $P(A|B)$ de que ocurra A habiéndose dado B, se puede obtener según se muestra en la ecuación 17.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \tag{17}$$

Siguiendo este enfoque, se desarrollaron las redes bayesianas, de las que se puede ver un ejemplo en la figura 5.5. Estas redes son modelos probabilísticos para análisis multivariable. Se definen como un grafo dirigido con una función de distribución probabilística asociada, en el que los nodos del grafo representan las variables y las aristas indican las dependencias entre las distintas variables. Los nodos pueden actuar tanto una premisa como una conclusión [CGH97].

Las redes bayesianas se definen como un grafo dirigido con una función de distribución probabilística asociada

En esta tesis hemos utilizado varios algoritmos para el aprendizaje de la estructura de la red. En concreto, hemos utilizado K2 [CH91] , *Tree Augmented Naïve* (TAN) [FGG97], y Naïve Bayes (clasificador ingenuo) [Lew98].

5.5.4 Máquinas de soporte vectorial

Este tipo de algoritmos están principalmente relacionados con problemas de clasificación y regresión. El espacio sobre el que clasifican este tipo de algoritmos es un hiperplano que divide la representación n-dimensional de los datos, en varias regiones,

buscando la separación óptima entre los puntos de cada clase. Se define el hiperplano como aquel que maximiza el margen (m) entre las dos clases del espacio.

Formalmente, dado un conjunto de datos de entrenamiento D tal que,

$$D = \{(x_i, c_i) x_i \in \mathbb{R}^p, c_i \in \{-1, 1\}\}_{i=1}^{n} \tag{18}$$

donde c_i indica la clase del punto x_i, cuyo valor oscila entre 1 ó -1, el cual es un vector de números reales en un espacio p-dimensional.

El objetivo del algoritmo es encontrar el hiperplano con un mayor margen m que divide los puntos c_i cuyo valor es 1 de los que su valor es -1. La representación del plano se hace mediante los puntos x_i tales que, realizado el producto escalar entre x y w y restado el valor de m, su valor sea cero, donde w es un vector normal y ortogonal respecto al hiperplano.

En las máquinas de soporte vectorial, el espacio es un hiperplano que divide la representación n-dimensional de los datos, en varias regiones, buscando la separación óptima entre los puntos de cada clase.

De esta manera tenemos dos hiperplanos, uno para cada clase de c_i que se pueden describir según la ecuación

$$c_i \, (w \cdot x_i - m) \geqslant 1, \quad \forall x_i | \, 1 \leqslant i \leqslant n \tag{19}$$

El problema de optimización para encontrar w y m lleva a una clasificación lineal, ya que se puede formular en términos de productos escalares. Sin embargo, en muchas ocasiones no se puede dividir el espacio por hiperplanos lineales, por lo que es necesario utilizar las denominadas «funciones de kernel» [AW99]. Estas funciones llevan clasificaciones polinomiales, radiales o sigmoidales.

5.6 RESULTADOS OBTENIDOS

En este apartado vamos a analizar cada uno de los resultados obtenidos para cada uno de los métodos anteriormente expuestos. Primeramente veremos el método de evaluación que se ha realizado para, posteriormente, ver los resultados obtenidos.

5.6.1 Método de evaluación

Para la evaluación de las dos aproximaciones se ha utilizado la misma métrica. Esta se detalla en el punto 5.6.2. En el caso

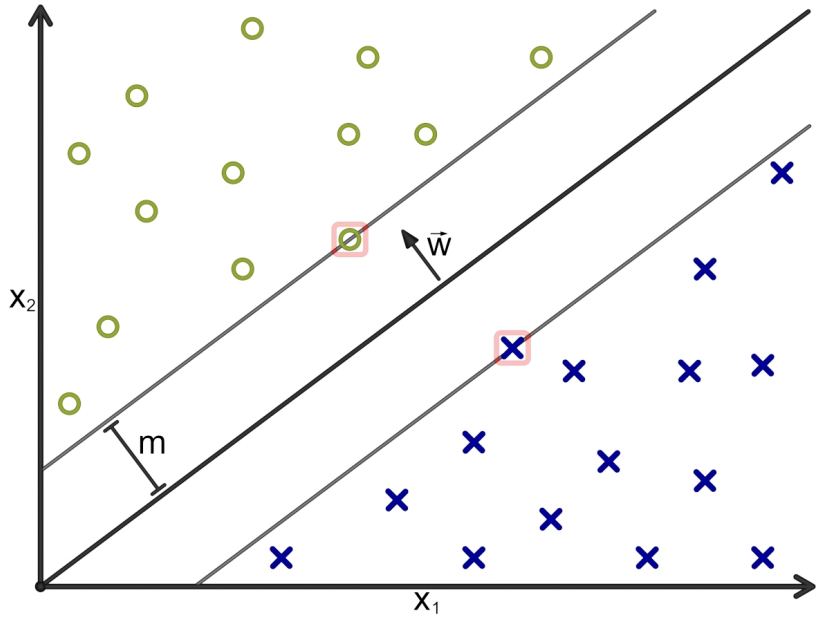


Figura 5.6: Ejemplo de clasificación mediante el algoritmo de máquinas de soporte vectorial en un espacio bi-dimensional.

Para la evaluación de los resultados del primer método se ha desarrollado una herramienta ad-hoc, mientras que para los modelos de aprendizaje automático se ha usado WEKA.

de la aproximación por detección de anomalías, la herramienta utilizada ha sido desarrollada ad-hoc a fin de evaluar esta aproximación. En cambio, para la evaluación de la aplicación de los algoritmos de aprendizaje automático hemos empleado la herramienta WEKA (*Waikato Environment for Knowledge Analysis*)[6]. En concreto, los algoritmos utilizados de esta herramienta se pueden ver en la tabla 5.2. En aquellos que no se indica nada, la configuración utilizada ha sido la que provee el programa por defecto. Para una referencia más exhaustiva de los modelos de aprendizaje consúltese la sección 5.5.

Los resultados obtenidos se ha medido mediante la técnica de validación cruzada [Koh95, DK82]. Está práctica separa el conjunto de datos de entrada en k subconjuntos complementarios, utilizando uno de ellos para la conformación de conjunto de datos de muestra, denominado «conjunto de prueba», mientras que el resto de subconjuntos conforma el «conjunto de entrenamiento». Para obtener el ratio de error para la muestra final, se realiza la media aritmética de los ratios de error obtenidos en cada una de las k iteraciones.

6 http://www.cs.waikato.ac.nz/ml/weka/

Algoritmos utilizados	Variantes usadas
SimpleLogistic	
NaïveBayes	
BayesNet	K2 y TAN
SMO	PolyKernel y NormalizedPolyKernel
IBK	Valores de K: 1, 3 y 5
J48	
RandomTree	
RandomForest	Valor de I: 10, 50 y 100

Tabla 5.2: Algoritmos utilizados en la evaluación de los predictores de *malware*.

5.6.2 Medición de los resultados

Los resultados se miden en base a 4 parámetros: TPR, FPR, AUC y Precisión.

En esta sección detallaremos los resultados obtenidos a la hora de evaluar los distintos elementos analizados como predictores de *malware*. Para cada uno de ellos, se realizará un experimento para demostrar su validez como predictor de *malware*, usando distintos algoritmos.

La evaluación se realizará en base a 4 parámetros:

- *True Positive Ratio* (TPR), también denominada sensitividad, el cual se calcula dividiendo el número de muestras de *goodware* correctamente clasificadas (TP) entre el total de muestras ($TP + FN$).

$$TPR = \frac{TP}{(TP + FN)} \tag{20}$$

- *False Positive Ratio* (FPR), el cual se calcula dividiendo el número de muestras de *malware* cuya clasificación se erró (FP) entre el número total de muestras ($FP + TN$).

$$FPR = \frac{FP}{(FP + TN)} \tag{21}$$

- *Precisión*, que se calcula dividiendo el total de aciertos en entre el número total de instancias del conjunto de datos.

$$\text{Precisión} = \frac{TP + TN}{(P + N)} \tag{22}$$

- *Area Under ROC Curve* [SKM09], la cual establece la relación entre los falsos negativos y los falsos positivos.

5.6.3 Resultados del método de detección de anomalías

A continuación se muestran los resultados obtenidos tras la aplicación del método de detección de anomalías, detallando dentro del apartado 5.4 a cada conjunto de datos utilizado, descrito en el apartado 5.2.

El conjunto de tablas que engloban estos resultados van desde la 5.3 hasta la 5.14. De la misma forma, se han representado las curvas ROC de cada uno de los modelos, que se muestran en el grupo de figuras que va desde la 5.7 hasta la 5.18.

El mejor resultado se obtuvo con los aplicando los permisos, la distancia del coseno combinando con la media, con un resultado de AUC = 0,91 *y* Precisión = 89,04 %

En cuanto a los resultados obtenidos, cabe destacar la gran cantidad de modelos generados cuya predicción no es mejor que la que se obtiene tirando una moneda al aire. Sin embargo, en cada uno de los grupos es cierto que alguna combinación da un área bajo la curva ROC cercana al 90 %, con unas tasas de precisión buenas, cercanas al 100 %.

En concreto, las combinaciones que mejores resultados ofrecen son las que se obtienen a través de los permisos, utilizando como métrica la distancia del coseno y como métrica de combinación la media, con 0,91 puntos de área bajo la curva ROC y un 89,04 % de precisión.

5.6.4 Resultados del método de aprendizaje automático supervisado

Cadenas de caracteres

A continuación se muestran los resultados obtenidos al generar los modelos con las cadenas de caracteres Estos resultados se pueden ver reflejados en la tabla 5.18.

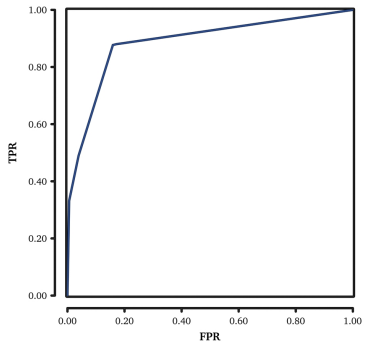
En estos resultados podemos ver como se consigue el mejor resultado con la utilización de Simple Logistic. Bajo esta configuración, obtenemos una curva ROC con un valor de 0,97 y una precisión superior al 93 %. El resto de valores son bastante dispares, aunque destaca la baja precisión que se obtiene con los algoritmos IBK.

Permisos

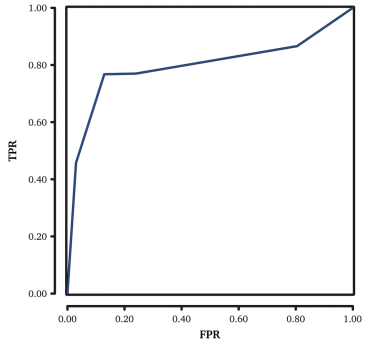
Los resultados obtenidos en el análisis de los permisos están reflejados en la tabla 5.19. En ella podemos apreciar unos resultados similares en las distintas familias de los algoritmos, si bien

Tabla 5.3: Resultados obtenidos de las características utilizando la distancia Manhattan como combinador.

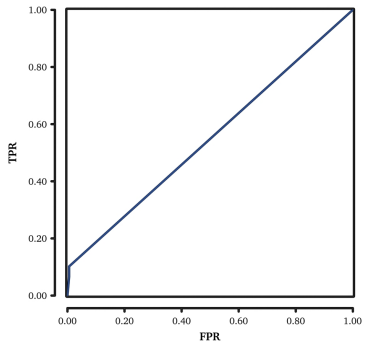
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	2199,44361	1,00000	1,00000	**0,88622**	50,00 %
	4099,28321	0,87988	0,17117		85,44 %
	5999,12281	**0,87688**	**0,15916**		**85,89 %**
	7898,96241	**0,87688**	**0,15916**		**85,89 %**
	9798,80201	**0,87688**	**0,15916**		**85,89 %**
	11698,64161	0,48949	0,03904		72,52 %
	13598,48121	0,33153	0,00601		66,28 %
	15498,32081	0,33033	0,00601		66,22 %
	17398,16041	0,33033	0,00601		66,22 %
	19298,00001	0,00000	0,00000		50,00 %
Máx.	13757,00000	1,00000	1,00000	0,79695	50,00 %
	15849,33333	0,86607	0,80480		53,06 %
	17941,66667	0,76997	0,24024		76,49 %
	20034,00000	**0,76757**	**0,12913**		**81,92 %**
	22126,33334	**0,76757**	**0,12913**		**81,92 %**
	24218,66667	**0,76757**	**0,12913**		**81,92 %**
	26311,00001	**0,76757**	**0,12913**		**81,92 %**
	28403,33334	**0,76757**	**0,12913**		**81,92 %**
	30495,66668	0,45766	0,03003		71,38 %
	32588,00001	0,00000	0,00000		50,00 %
Mín.	0,00000	1,00000	1,00000	0,54794	50,00 %
	1882,11111	**0,10210**	**0,00601**		**54,80 %**
	3764,22222	0,06607	0,00601		53,00 %
	5646,33334	0,06607	0,00601		53,00 %
	7528,44445	0,06607	0,00601		53,00 %
	9410,55556	0,06607	0,00601		53,00 %
	11292,66667	0,06607	0,00601		53,00 %
	13174,77779	0,06607	0,00601		53,00 %
	15056,88890	0,06607	0,00601		53,00 %
	16939,00001	0,00000	0,00000		50,00 %



(a) Distancia Manhattan usando la media como métrica de combinación.



(b) Distancia Manhattan usando la distancia máxima como métrica de combinación.



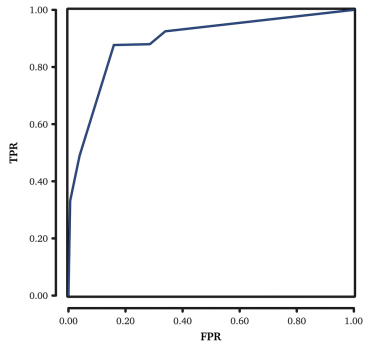
(c) Distancia Manhattan usando la distancia mínima como métrica de combinación.

Figura 5.7: Curvas ROC que representan el uso de las características utilizando como medida la distancia Manhattan. Como se puede apreciar, el mejor resultado se obtiene utilizando la media. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.3.

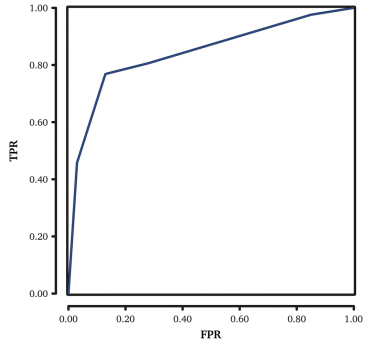
.

Tabla 5.4: Resultados obtenidos de las características utilizando la distancia euclidiana como combinador.

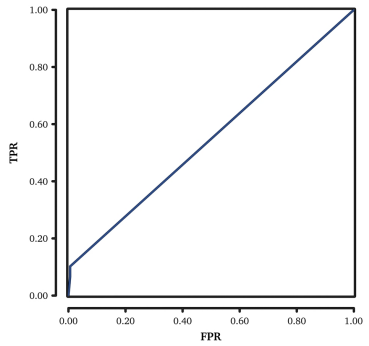
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	24,03266	1,00000	1,00000	**0,892042493**	50,00 %
	36,80053	0,92492	0,33934		85,44 %
	49,56840	0,87988	0,28529		85,89 %
	62,33627	0,87688	0,15916		85,89 %
	75,10414	**0,87688**	**0,15916**		**85,89 %**
	87,87201	0,87688	0,15916		72,52 %
	100,63987	0,48949	0,03904		66,28 %
	113,40774	0,33153	0,00601		66,22 %
	126,17561	0,33033	0,00601		66,22 %
	138,94348	0,00000	0,00000		50,00 %
Máx.	117,29024	1,00000	1,00000	0,842501961	50,00 %
	124,31593	0,97598	0,84985		53,06 %
	131,34162	0,80601	0,27928		76,49 %
	138,36731	**0,76997**	**0,13514**		**81,92 %**
	145,39301	**0,76757**	**0,12913**		**81,92 %**
	152,41870	**0,76757**	**0,12913**		**81,92 %**
	159,44439	**0,76757**	**0,12913**		**81,92 %**
	166,47008	**0,76757**	**0,12913**		**81,92 %**
	173,49577	0,45766	0,03003		71,38 %
	180,52147	0,00000	0,00000		50,00 %
Mín.	0,00000	1,00000	1,00000	0,547939832	50,00 %
	14,46111	**0,10210**	**0,00601**		**54,80 %**
	28,92222	0,10210	0,00601		53,00 %
	43,38333	0,10210	0,00601		53,00 %
	57,84444	0,06607	0,00601		53,00 %
	72,30555	0,06607	0,00601		53,00 %
	86,76666	0,06607	0,00601		53,00 %
	101,22777	0,06607	0,00601		53,00 %
	115,68888	0,06607	0,00601		53,00 %
	130,14999	0,00000	0,00000		50,00 %



(a) Distancia euclidiana usando la media como métrica de combinación.



(b) Distancia euclidiana usando la distancia máxima como métrica de combinación.



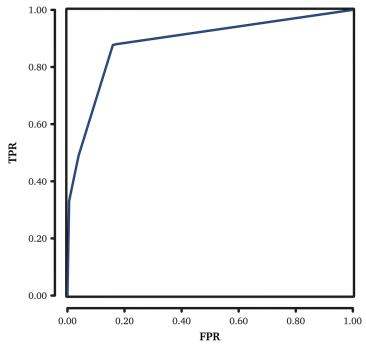
(c) Distancia euclidiana usando la distancia mínima como métrica de combinación.

Figura 5.8: Curvas ROC que representan el uso de las características utilizando como medida la distancia euclidiana. En este caso, tanto la media como la combinación con distancia máxima ofrecen buenos resultados. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.4.

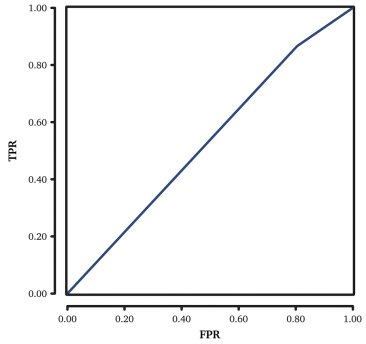
.

Tabla 5.5: Resultados obtenidos de las características utilizando la distancia del coseno como combinador.

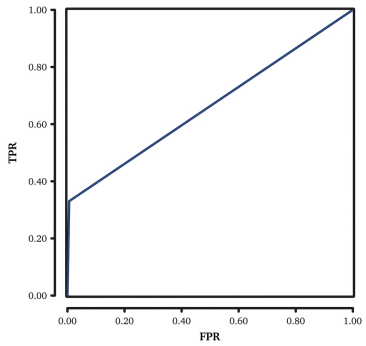
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	0,03220314	1,00	1,00	**0,882513144**	50,00 %
	0,13974723	**0,88**	**0,16**		**85,89 %**
	0,24729133	0,33	0,01		66,22 %
	0,35483542	0,33	0,01		66,22 %
	0,46237952	0,33	0,01		66,22 %
	0,56992362	0,33	0,01		66,22 %
	0,67746771	0,33	0,01		66,22 %
	0,78501181	0,33	0,01		66,22 %
	0,89255590	0,33	0,01		66,22 %
	1,00010000	0,00	0,00		50,00 %
Máx.	0,20343300	1,00	1,00	0,530630631	50,00 %
	0,29194156	0,87	0,80		53,06 %
	0,38045011	**0,87**	**0,80**		**53,06 %**
	0,46895867	**0,87**	**0,80**		**53,06 %**
	0,55746722	**0,87**	**0,80**		**53,06 %**
	0,64597578	**0,87**	**0,80**		**53,06 %**
	0,73448433	**0,87**	**0,80**		**53,06 %**
	0,82299289	**0,87**	**0,80**		**53,06 %**
	0,91150144	**0,87**	**0,80**		**53,06 %**
	1,00001000	0,00	0,00		50,00 %
Mín.	-0,00010600	1,00	1,00	0,662162162	50,00 %
	0,11101800	**0,33**	**0,01**		**66,22 %**
	0,22214200	**0,33**	**0,01**		**66,22 %**
	0,33326600	**0,33**	**0,01**		**66,22 %**
	0,44439000	**0,33**	**0,01**		**66,22 %**
	0,55551400	**0,33**	**0,01**		**66,22 %**
	0,66663800	**0,33**	**0,01**		**66,22 %**
	0,77776200	**0,33**	**0,01**		**66,22 %**
	0,88888600	**0,33**	**0,01**		**66,22 %**
	1,00001000	0,00	0,00		50,00 %



(a) Distancia del coseno usando la media como métrica de combinación.



(b) Distancia del coseno usando la distancia máxima como métrica de combinación.



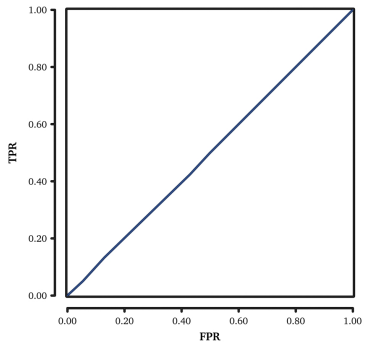
(c) Distancia del coseno usando la distancia mínima como métrica de combinación.

Figura 5.9: Curvas ROC que representan el uso de las características utilizando como medida la distancia del coseno. En este caso, únicamente la media consigue un valor de área bajo la curva superior al 80 %. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.5.

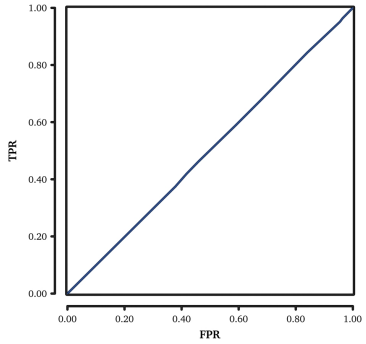
.

Tabla 5.6: Resultados obtenidos de los permisos utilizando la distancia Manhattan como combinador.

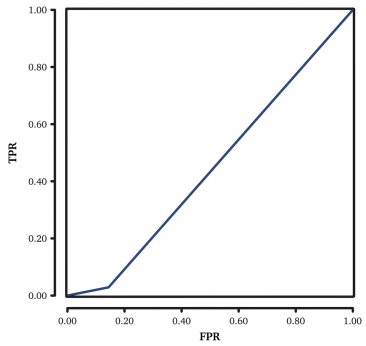
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	24701,35955	1,00000	1,00000	0,49945	50,00 %
	28549,25163	0,49850	0,49850		50,00 %
	32397,14371	0,42402	0,42943		49,73 %
	36245,03579	0,34535	0,34835		49,85 %
	40092,92787	0,30751	0,30931		49,91 %
	43940,81995	**0,13273**	**0,12913**		**50,18 %**
	47788,71203	0,05045	0,05405		49,82 %
	51636,60412	0,01982	0,02102		49,94 %
	55484,49620	0,00601	0,00601		50,00 %
	59332,38828	0,00000	0,00000		50,00 %
Máx.	54175,00000	1,00000	1,00000	**0,49961**	50,00 %
	57340,55557	0,96336	0,96396		49,97 %
	60506,11113	0,95135	0,95495		49,82 %
	63671,66670	0,84384	0,84084		50,15 %
	66837,22227	0,68108	0,68168		49,97 %
	70002,77783	0,58799	0,58859		49,97 %
	73168,33340	**0,46246**	**0,45946**		**50,15 %**
	76333,88897	0,41922	0,41742		50,09 %
	79499,44453	0,37477	0,37838		49,82 %
	82665,00010	0,00000	0,00000		50,00 %
Mín.	**0,00000**	**1,00000**	**1,00000**	0,44234	**50,00 %**
	2212,55557	0,02883	0,14414		44,23 %
	4425,11113	0,01381	0,06907		47,24 %
	6637,66670	0,00841	0,04204		48,32 %
	8850,22227	0,00541	0,02703		48,92 %
	11062,77783	0,00300	0,01502		49,40 %
	13275,33340	0,00240	0,01201		49,52 %
	15487,88897	0,00180	0,00901		49,64 %
	17700,44453	0,00180	0,00901		49,64 %
	19913,00010	**0,00000**	**0,00000**		**50,00 %**



(a) Distancia Manhattan usando la media como métrica de combinación.



(b) Distancia Manhattan usando la distancia máxima como métrica de combinación.



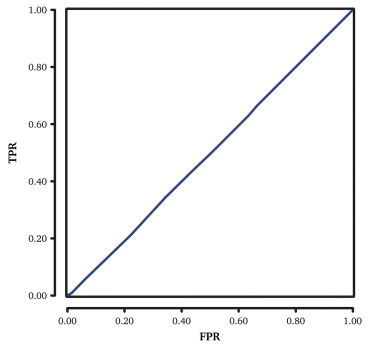
(c) Distancia Manhattan usando la distancia mínima como métrica de combinación.

Figura 5.10: Curvas ROC que representan el uso de las características utilizando como medida la distancia Manhattan. Se puede apreciar que ninguna combinación ofrece buenos resultados. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.6.

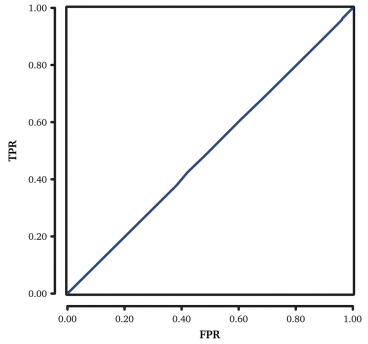
.

Tabla 5.7: Resultados obtenidos de los permisos utilizando la distancia euclidiana como combinador.

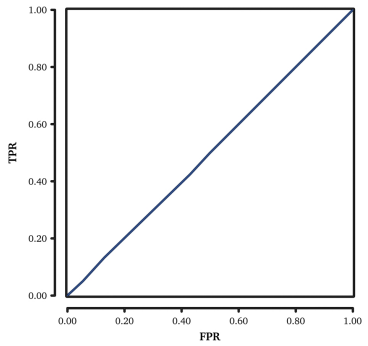
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	121,59535	1,00000	1,00000	**0,496929362**	50,00 %
	134,55437	0,66366	0,66366		50,00 %
	147,51340	0,62883	0,63363		49,76 %
	160,47242	0,50270	0,50751		49,76 %
	173,43144	0,43123	0,43243		49,94 %
	186,39046	**0,34234**	**0,34234**		**50,00 %**
	199,34949	0,21201	0,22222		49,49 %
	212,30851	0,05826	0,06306		49,76 %
	225,26753	0,00901	0,01502		49,70 %
	238,22655	**0,00000**	**0,00000**		**50,00 %**
Máx.	232,75524	1,00000	1,00000	0,498845693	50,00 %
	238,83968	0,96336	0,96396		49,97 %
	244,92412	0,95736	0,96096		49,82 %
	251,00856	0,87928	0,88288		49,82 %
	257,09301	0,69489	0,69670		49,91 %
	263,17745	0,60541	0,60360		50,09 %
	269,26189	0,48769	0,48649		50,06 %
	275,34633	**0,42402**	**0,42042**		**50,18 %**
	281,43078	0,37778	0,38138		49,82 %
	287,51522	0,00000	0,00000		50,00 %
Mín.	**0,00000**	**1,00000**	**1,00000**	0,413513514	**50,00 %**
	15,67927	0,04324	0,21622		41,35 %
	31,35854	0,03664	0,18318		42,67 %
	47,03781	0,02883	0,14414		44,23 %
	62,71708	0,01922	0,09610		46,16 %
	78,39635	0,00961	0,04805		48,08 %
	94,07562	0,00541	0,02703		48,92 %
	109,75489	0,00240	0,01201		49,52 %
	125,43416	0,00180	0,00901		49,64 %
	141,11343	**0,00000**	**0,00000**		**50,00 %**



(a) Distancia euclidiana usando la media como métrica de combinación.



(b) Distancia euclidiana usando la distancia máxima como métrica de combinación.



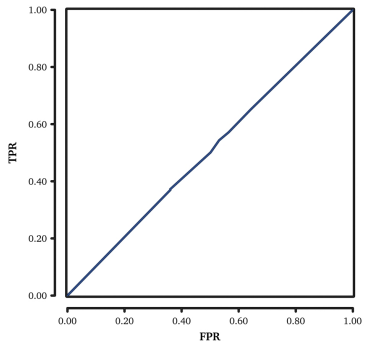
(c) Distancia euclidiana usando la distancia mínima como métrica de combinación.

Figura 5.11: Curvas ROC que representan el uso de las características utilizando como medida la distancia euclidiana. Como se puede apreciar, los resultados no son significativamente distintos a los que se obtiene tirando una moneda al aire. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.7.

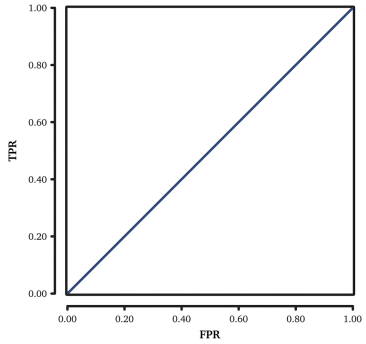
.

Tabla 5.8: Resultados obtenidos de los permisos utilizando la distancia del coseno como combinador.

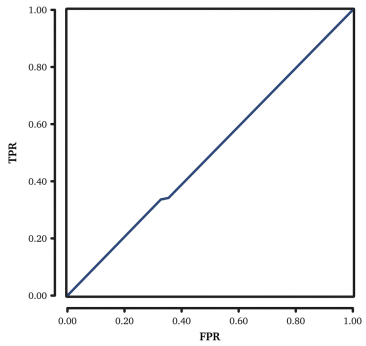
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	0,52044118	1,00	1,00	**0,505140275**	50,00 %
	0,57372660	0,65	0,64		50,45 %
	0,62701203	0,57	0,56		50,33 %
	0,68029745	0,54	0,53		50,60 %
	0,73358288	0,50	0,50		49,97 %
	0,78686830	**0,37**	**0,36**		**50,60 %**
	0,84015373	**0,37**	**0,36**		**50,60 %**
	0,89343915	0,37	0,36		50,45 %
	0,94672458	0,37	0,36		50,45 %
	1,00001000	0,00	0,00		50,00 %
Máx.	**1,00000000**	**1,00**	**1,00**	**0,5**	**50,00 %**
	1,00000111	0,00	0,00		50,00 %
	1,00000222	0,00	0,00		50,00 %
	1,00000333	0,00	0,00		50,00 %
	1,00000444	0,00	0,00		50,00 %
	1,00000556	0,00	0,00		50,00 %
	1,00000667	0,00	0,00		50,00 %
	1,00000778	0,00	0,00		50,00 %
	1,00000889	0,00	0,00		50,00 %
	1,00001000	0,00	0,00		50,00 %
Mín.	-0,00100000	1,00	1,00	0,497354111	50,00 %
	0,11022333	0,34	0,35		49,37 %
	0,22144667	0,34	0,34		49,97 %
	0,33267000	0,34	0,34		49,97 %
	0,44389333	0,34	0,34		50,09 %
	0,55511667	0,34	0,33		50,21 %
	0,66634000	0,34	0,33		50,33 %
	0,77756333	0,34	0,33		50,33 %
	0,88878667	**0,34**	**0,33**		**50,45 %**
	1,00001000	0,00	0,00		50,00 %



(a) Distancia del coseno usando la media como métrica de combinación.



(b) Distancia del coseno usando la distancia máxima como métrica de combinación.



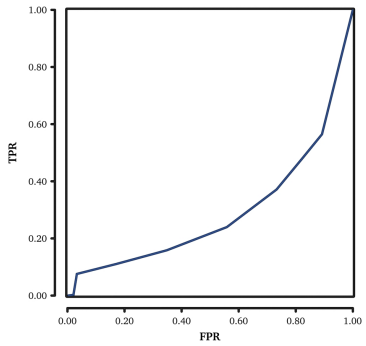
(c) Distancia del coseno usando la distancia mínima como métrica de combinación.

Figura 5.12: Curvas ROC que representan el uso de las características utilizando como medida la distancia del coseno. Como se puede apreciar, los resultados obtenidos no distan mucho de lanzar una moneda al aire. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.8.

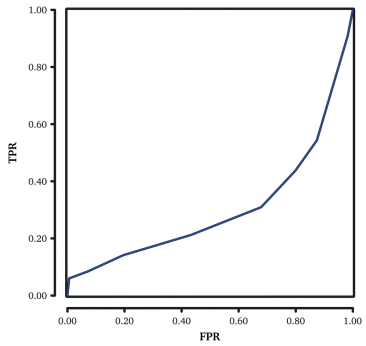
.

Tabla 5.9: Resultados obtenidos de las cadenas utilizando la distancia Manhattan como combinador.

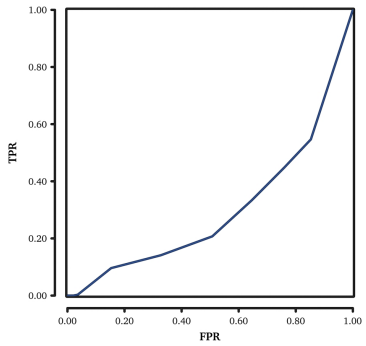
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	1042516,95554	1,00000	1,00000	0,29121	50,00 %
	1297301,02116	0,56456	0,89189		33,63 %
	1552085,08679	0,47988	0,82282		32,85 %
	1806869,15241	0,37117	0,73273		31,92 %
	2061653,21803	0,23964	0,55856		34,05 %
	2316437,28365	0,15856	0,34835		40,51 %
	2571221,34927	0,11051	0,17117		46,97 %
	2826005,41490	**0,07568**	**0,03303**		**52,13 %**
	3080789,48052	0,00240	0,02102		49,07 %
	3335573,54614	0,00000	0,00300		49,85 %
Máx.	2115684,24601	1,00000	1,00000	**0,30224**	49,85 %
	2386946,41796	0,90871	0,98198		50,00 %
	2658208,58992	0,54294	0,87387		46,34 %
	2929470,76187	0,43724	0,79880		33,45 %
	3200732,93383	0,30931	0,67868		31,92 %
	3471995,10578	0,21201	0,43243		31,53 %
	3743257,27774	0,14114	0,19520		38,98 %
	4014519,44969	0,08468	0,07207		47,30 %
	4285781,62165	0,06006	0,00601		50,63 %
	4557043,79360	**0,00000**	**0,00000**		**52,70 %**
Mín.	**0,00000**	**1,00000**	**1,00000**	0,29987	**50,00 %**
	274472,12573	**0,54655**	**0,85285**		**50,00 %**
	548944,25146	0,44805	0,75976		34,68 %
	823416,37718	0,33333	0,64565		34,41 %
	1097888,50291	0,20721	0,50751		34,38 %
	1372360,62864	0,14114	0,32733		34,98 %
	1646832,75437	0,09610	0,15315		40,69 %
	1921304,88009	0,00300	0,03604		47,15 %
	2195777,00582	0,00000	0,02102		48,35 %
	2470249,13155	0,00000	0,00000		48,95 %



(a) Distancia Manhattan usando la media como métrica de combinación.



(b) Distancia Manhattan usando la distancia máxima como métrica de combinación.



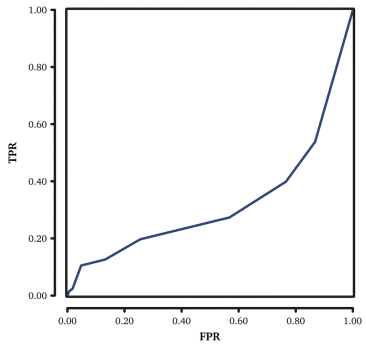
(c) Distancia Manhattan usando la distancia mínima como métrica de combinación.

Figura 5.13: Curvas ROC que representan el uso de las cadenas utilizando como medida la distancia Manhattan. Los resultados son paupérrimos, con un área bajo la curva ROC cercana al 0,3 en todos los casos. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.9.

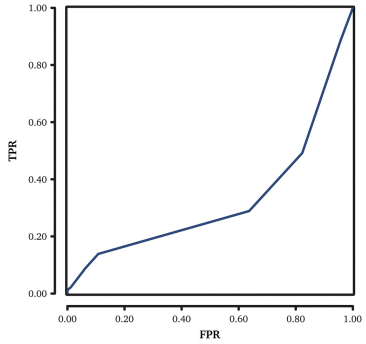
.

Tabla 5.10: Resultados obtenidos de las cadenas utilizando la distancia euclidiana como combinador.

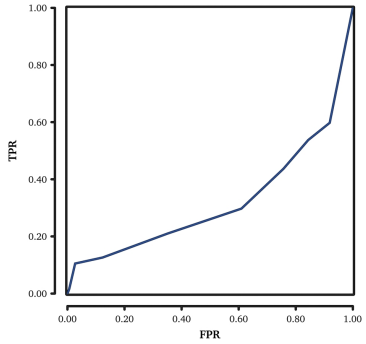
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	1292555,44	0,99880	1,00000	0,321240159	49,94 %
	1494979,97	0,53814	0,86787		33,51 %
	1697404,49	0,39880	0,76577		31,65 %
	1899829,02	0,27327	0,56757		35,29 %
	2102253,55	0,19700	0,25526		47,09 %
	2304678,08	0,12613	0,13213		49,70 %
	2507102,61	**0,10511**	**0,04805**		**52,85 %**
	2709527,14	0,02402	0,01802		50,30 %
	2911951,67	0,01502	0,00601		50,45 %
	3114376,19	0,00000	0,00000		50,00 %
Máx.	2486071,46	1,00000	1,00000	**0,326824121**	50,00 %
	2643030,09	0,88829	0,95796		46,52 %
	2799988,72	0,49189	0,82282		33,45 %
	2956947,35	0,28889	0,63664		32,61 %
	3113905,98	0,19339	0,30030		44,65 %
	3270864,60	**0,13874**	**0,10811**		**51,53 %**
	3427823,23	0,08889	0,06306		51,29 %
	3584781,86	0,02222	0,01201		50,51 %
	3741740,49	0,01201	0,00000		50,60 %
	3898699,12	0,00000	0,00000		50,00 %
Mín.	0,00	1,00000	1,00000	0,320127335	50,00 %
	308138,33	0,59760	0,91892		33,93 %
	616276,67	0,53754	0,84384		34,68 %
	924415,01	0,43664	0,75676		33,99 %
	1232553,35	0,29730	0,60961		34,38 %
	1540691,69	0,21021	0,35135		42,94 %
	1848830,03	0,12613	0,12312		50,15 %
	2156968,37	**0,10511**	**0,02703**		**53,90 %**
	2465106,71	0,01502	0,00601		50,45 %
	2773245,05	0,00000	0,00000		50,00 %



(a) Distancia euclidiana usando la media como métrica de combinación.



(b) Distancia euclidiana usando la distancia máxima como métrica de combinación.



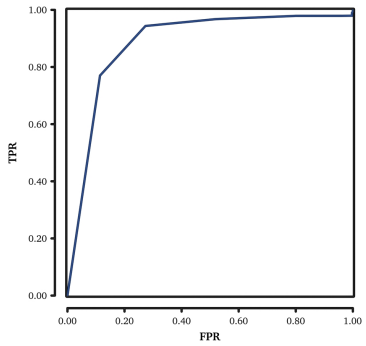
(c) Distancia euclidiana usando la distancia mínima como métrica de combinación.

Figura 5.14: Curvas ROC que representan el uso de las cadenas utilizando como medida la distancia euclidiana. El área bajo la curva ROC tiene resultados muy pobres, cercanos al 0,32. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.10.

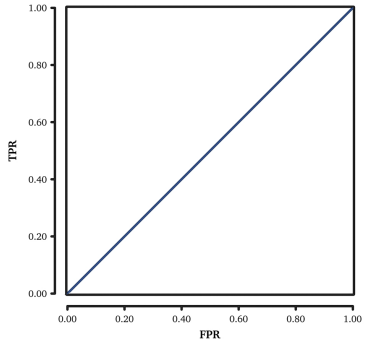
.

Tabla 5.11: Resultados obtenidos de las cadenas utilizando la distancia del coseno como combinador.

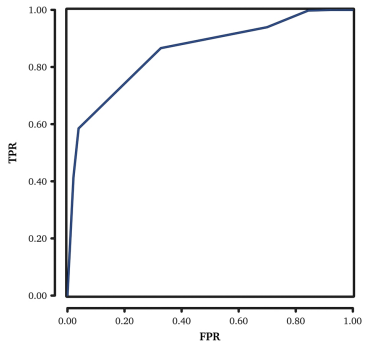
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	0,83693246	1,00	1,00	**0,884413242**	50,00 %
	0,85505219	0,99	1,00		49,70 %
	0,87317191	0,99	1,00		49,61 %
	0,89129164	0,98	1,00		49,13 %
	0,90941137	0,98	0,96		50,90 %
	0,92753109	0,98	0,80		58,86 %
	0,94565082	0,97	0,52		72,40 %
	0,96377055	**0,94**	**0,27**		**83,51 %**
	0,98189027	0,77	0,11		82,79 %
	1,00001000	0,00	0,00		50,00 %
Máx.	**1,00000000**	**1,00**	**1,00**	0,5	**50,00 %**
	1,00000111	0,00	0,00		50,00 %
	1,00000222	0,00	0,00		50,00 %
	1,00000333	0,00	0,00		50,00 %
	1,00000444	0,00	0,00		50,00 %
	1,00000556	0,00	0,00		50,00 %
	1,00000667	0,00	0,00		50,00 %
	1,00000778	0,00	0,00		50,00 %
	1,00000889	0,00	0,00		50,00 %
	1,00001000	0,00	0,00		50,00 %
Mín.	0,00000000	1,00	1,00	0,854307461	50,00 %
	0,11111222	1,00	0,98		51,20 %
	0,22222444	1,00	0,95		52,70 %
	0,33333667	1,00	0,92		53,75 %
	0,44444889	1,00	0,84		57,69 %
	0,55556111	0,94	0,70		61,98 %
	0,66667333	0,87	0,33		76,94 %
	0,77778556	**0,58**	**0,04**		**77,30 %**
	0,88889778	0,41	0,02		69,64 %
	1,00001000	0,00	0,00		50,00 %



(a) Distancia del coseno usando la media como métrica de combinación.



(b) Distancia del coseno usando la distancia máxima como métrica de combinación.



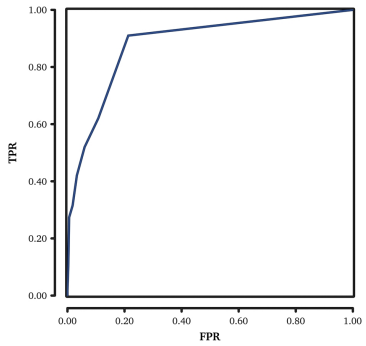
(c) Distancia del coseno usando la distancia mínima como métrica de combinación.

Figura 5.15: Curvas ROC que representan el uso de las características utilizando como medida la distancia del coseno. En este caso, tanto la media como la distancia mínima ofrecen resultados superiores al 0,80 como valor del área bajo la curva ROC. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.11.

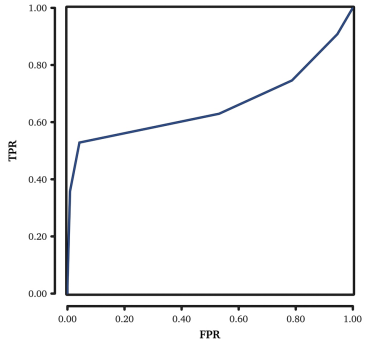
.

Tabla 5.12: Resultados obtenidos de las características y los permisos utilizando la distancia Manhattan como combinador.

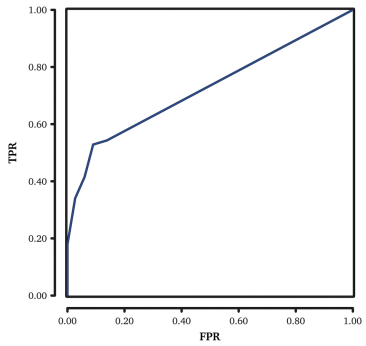
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	8755,51128	1,00000	1,00000	**0,88167**	50,00 %
	17468,67613	**0,90991**	**0,21321**		**84,83 %**
	26181,84099	0,62042	0,10811		75,62 %
	34895,00584	0,51952	0,06006		72,97 %
	43608,17069	0,42042	0,03303		69,37 %
	52321,33555	0,31471	0,01802		64,83 %
	61034,50040	0,27387	0,00601		63,39 %
	69747,66525	0,10751	0,00300		55,23 %
	78460,83011	**0,01862**	**0,00000**		**50,93 %**
	87173,99496	0,00000	0,00000		50,00 %
Máx.	47404,00000	1,00000	1,00000	0,65852	50,00 %
	54503,66667	0,98498	0,99099		49,70 %
	61603,33334	0,90811	0,94595		48,11 %
	68703,00000	0,74595	0,78679		47,96 %
	75802,66667	0,62943	0,53153		54,89 %
	82902,33334	**0,52853**	**0,04204**		**74,32 %**
	90002,00001	0,35796	0,00901		67,45 %
	97101,66667	0,02643	0,00000		51,32 %
	104201,33334	0,00480	0,00000		50,24 %
	111301,00001	0,00000	0,00000		50,00 %
Mín.	0,00000	1,00000	1,00000	0,72431	50,00 %
	3470,00000	0,54294	0,13814		70,24 %
	6940,00000	**0,52853**	**0,09009**		**71,92 %**
	10410,00000	0,41502	0,06006		67,75 %
	13880,00000	0,34054	0,02703		65,68 %
	17350,00001	0,17898	0,00000		58,95 %
	20820,00001	0,10450	0,00000		55,23 %
	24290,00001	0,02583	0,00000		51,29 %
	27760,00001	0,00661	0,00000		50,33 %
	31230,00001	0,00000	0,00000		50,00 %



(a) Distancia Manhattan usando la media como métrica de combinación.



(b) Distancia Manhattan usando la distancia máxima como métrica de combinación.



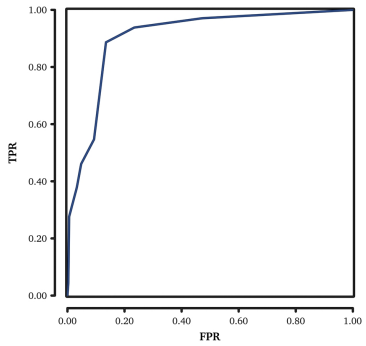
(c) Distancia Manhattan usando la distancia mínima como métrica de combinación.

Figura 5.16: Curvas ROC que representan el uso de las características y los permisos utilizando como medida la distancia Manhattan. Se puede apreciar que la utilización de la media como método de combinación ofrece unos resultados cercanos al 90 %. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.12.

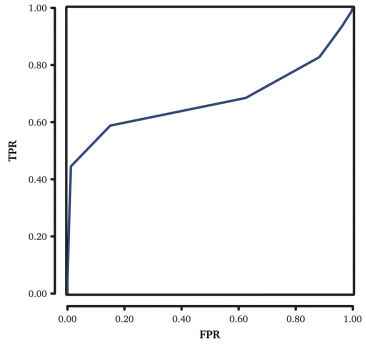
.

Tabla 5.13: Resultados obtenidos de las características y permisos utilizando la distancia euclidiana como combinador.

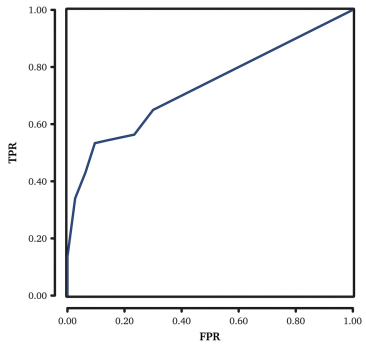
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	70,35688	1,00000	1,00000	**0,906084463**	50,00 %
	95,24643	0,97057	0,47147		74,95 %
	120,13597	0,93814	0,23423		85,20 %
	145,02552	**0,88649**	**0,13514**		**87,57 %**
	169,91506	0,54655	0,09309		72,67 %
	194,80461	0,46066	0,04805		70,63 %
	219,69415	0,37898	0,03303		67,30 %
	244,58370	0,27628	0,00601		63,51 %
	269,47324	0,04444	0,00300		52,07 %
	294,36279	0,00000	0,00000		50,00 %
Máx.	217,72460	1,00000	1,00000	0,67797888	50,00 %
	230,60165	0,98559	0,99399		49,58 %
	243,47870	0,93393	0,96096		48,65 %
	256,35575	0,82763	0,88288		47,24 %
	269,23281	0,68468	0,62462		53,00 %
	282,10986	**0,58799**	**0,15015**		**71,89 %**
	294,98691	0,44505	0,01201		71,65 %
	307,86396	0,05165	0,00000		52,58 %
	320,74102	0,00721	0,00000		50,36 %
	333,61807	0,00000	0,00000		50,00 %
Mín.	0,00000	1,00000	1,00000	0,729316704	50,00 %
	19,63557	0,64985	0,30030		67,48 %
	39,27114	0,56336	0,23423		66,46 %
	58,90671	0,54294	0,13814		70,24 %
	78,54228	**0,53333**	**0,09610**		**71,86 %**
	98,17784	0,43063	0,06306		68,38 %
	117,81341	0,34054	0,02703		65,68 %
	137,44898	0,13634	0,00000		56,82 %
	157,08455	0,01562	0,00000		50,78 %
	176,72012	0,00000	0,00000		50,00 %



(a) Distancia euclidiana usando la media como métrica de combinación.



(b) Distancia euclidiana usando la distancia máxima como métrica de combinación.



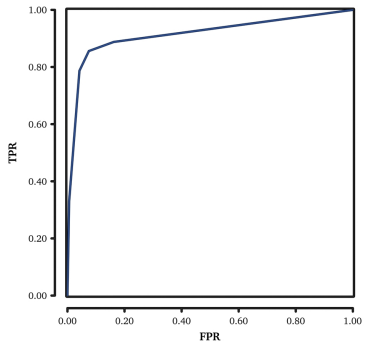
(c) Distancia euclidiana usando la distancia mínima como métrica de combinación.

Figura 5.17: Curvas ROC que representan el uso de las características y los permisos utilizando como medida la distancia euclidiana. Como se puede apreciar, la combinación utilizando la media como método obtiene un 0,9 de área bajo la curva ROC. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.13.

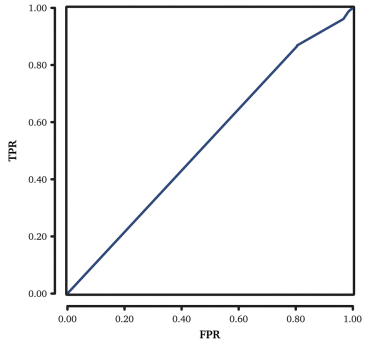
.

Tabla 5.14: Resultados obtenidos de las características y permisos utilizando la distancia del coseno como combinador.

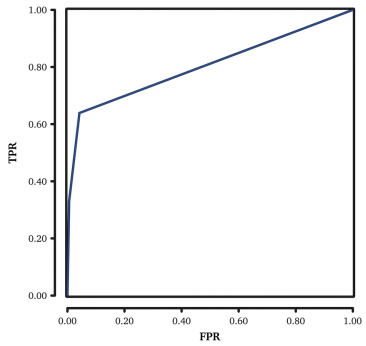
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	0,07126978	1,00	1,00	**0,914959103**	50,00 %
	0,17446314	0,89	0,16		86,28 %
	0,27765650	**0,86**	**0,08**		**89,04 %**
	0,38084985	0,79	0,04		87,24 %
	0,48404321	0,33	0,01		66,22 %
	0,58723657	0,33	0,01		66,22 %
	0,69042993	0,33	0,01		66,22 %
	0,79362328	0,33	0,01		66,22 %
	0,89681664	0,33	0,01		66,22 %
	1,00001000	0,00	0,00		50,00 %
Máx.	0,34282500	1,00	1,00	0,529312195	50,00 %
	0,41584556	0,99	0,98		50,12 %
	0,48886611	0,96	0,97		49,70 %
	0,56188667	0,87	0,80		53,18 %
	0,63490722	**0,87**	**0,80**		**53,06 %**
	0,70792778	**0,87**	**0,80**		**53,06 %**
	0,78094833	**0,87**	**0,80**		**53,06 %**
	0,85396889	**0,87**	**0,80**		**53,06 %**
	0,92698944	**0,87**	**0,80**		**53,06 %**
	1,00001000	0,00	0,00		50,00 %
Mín.	-0,10000000	1,00	1,00	0,803523343	50,00 %
	0,02222333	**0,64**	**0,04**		**79,85 %**
	0,14444667	0,33	0,01		66,22 %
	0,26667000	0,33	0,01		66,22 %
	0,38889333	0,33	0,01		66,22 %
	0,51111667	0,33	0,01		66,22 %
	0,63334000	0,33	0,01		66,22 %
	0,75556333	0,33	0,01		66,22 %
	0,87778667	0,33	0,01		66,22 %
	1,00001000	0,00	0,00		50,00 %



(a) Distancia del coseno usando la media como métrica de combinación.



(b) Distancia del coseno usando la distancia máxima como métrica de combinación.



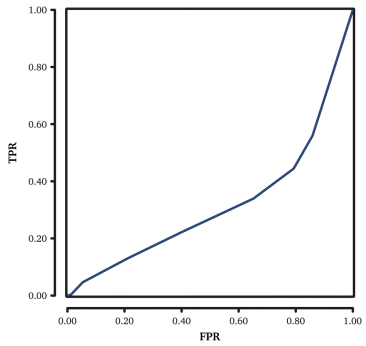
(c) Distancia del coseno usando la distancia mínima como métrica de combinación.

Figura 5.18: Curvas ROC que representan el uso de las características y los permisos utilizando como medida la distancia del coseno. En este caso, se obtiene un área bajo la curva ROC de 0,91 utilizando la media como combinador. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.14.

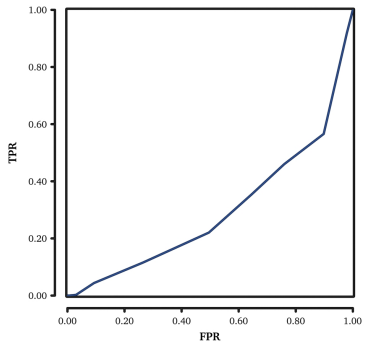
.

Tabla 5.15: Resultados obtenidos de las cadenas, las características y permisos utilizando la distancia Manhattan como combinador.

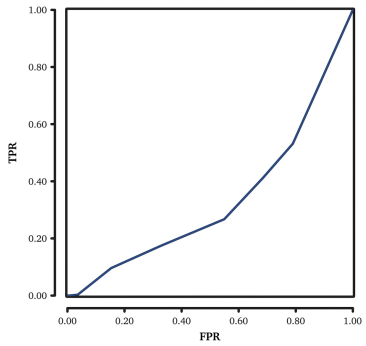
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	1042516,95554	0,55976	0,85886	**0,34846**	35,05 %
	1297301,02116	0,44444	0,79279		32,58 %
	1552085,08679	0,33934	0,65165		34,38 %
	1806869,15241	0,22162	0,39940		41,11 %
	2061653,21803	0,12973	0,21021		45,98 %
	2316437,28365	0,04685	0,05405		49,64 %
	2571221,34927	0,00300	0,01201		49,55 %
	2826005,41490	0,00000	0,00300		49,85 %
	3080789,48052	0,00000	0,00300		49,85 %
	3335573,54614	**0,00000**	**0,00000**		**50,00 %**
Máx.	2115684,24601	0,91832	0,97898	0,29558	46,97 %
	2386946,41796	0,56577	0,89790		33,39 %
	2658208,58992	0,46006	0,75976		35,02 %
	2929470,76187	0,35435	0,64565		35,44 %
	3200732,93383	0,22042	0,49550		36,25 %
	3471995,10578	0,11532	0,26426		42,55 %
	3743257,27774	0,04384	0,09309		47,54 %
	4014519,44969	0,00240	0,03003		48,62 %
	4285781,62165	0,00000	0,00601		49,70 %
	4557043,79360	**0,00000**	**0,00000**		**50,00 %**
Mín.	**0,00000**	**1,00000**	**1,00000**	0,33482	**50,00 %**
	274472,12573	0,53153	0,78979		37,09 %
	548944,25146	0,41141	0,68468		36,34 %
	823416,37718	0,26727	0,54955		35,89 %
	1097888,50291	0,17417	0,32733		42,34 %
	1372360,62864	0,09610	0,15315		47,15 %
	1646832,75437	0,00300	0,03604		48,35 %
	1921304,88009	0,00000	0,00601		49,70 %
	2195777,00582	0,00000	0,00300		49,85 %
	2470249,13155	**0,00000**	**0,00000**		**50,00 %**



(a) Distancia Manhattan usando la media como métrica de combinación.



(b) Distancia Manhattan usando la distancia máxima como métrica de combinación.



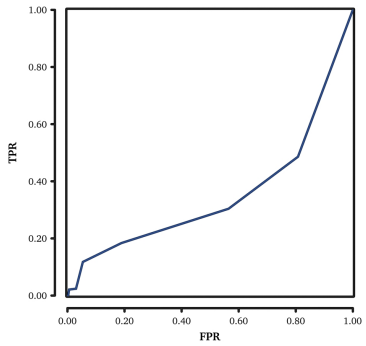
(c) Distancia Manhattan usando la distancia mínima como métrica de combinación.

Figura 5.19: Curvas ROC que representan el uso de las características y los permisos utilizando como medida la distancia del coseno. Lo valores obtenidos son bajos. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.15.

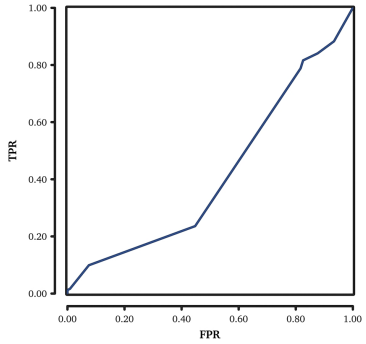
.

Tabla 5.16: Resultados obtenidos de las cadenas, las características y permisos utilizando la distancia euclidiana como combinador.

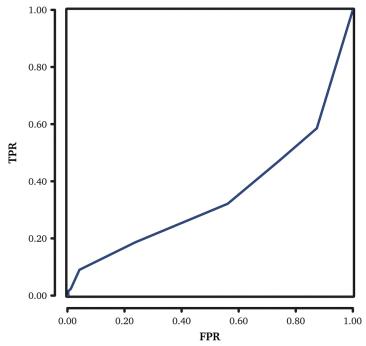
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	1135444,23	1,00000	1,00000	0,353027803	83,33 %
	1404377,88	0,48589	0,80781		75,05 %
	1673311,53	0,30390	0,56456		72,91 %
	1942245,18	0,18378	0,18919		82,93 %
	2211178,84	0,11772	0,05405		91,59 %
	2480112,49	0,02402	0,03003		80,00 %
	2749046,14	**0,02102**	**0,00601**		94,59 %
	3017979,79	0,00300	0,00300		83,33 %
	3286913,44	0,00000	0,00300		0,00 %
	3555847,09	0,00000	0,00000		0,00 %
Máx.	2376763,18	1,00000	1,00000	**0,416374032**	83,33 %
	2607893,81	0,88288	0,93393		82,54 %
	2839024,43	0,84084	0,87688		82,74 %
	3070155,06	0,81622	0,82583		83,17 %
	3301285,68	0,78799	0,81682		82,83 %
	3532416,31	0,23604	0,44745		72,51 %
	3763546,93	0,09910	0,07508		86,84 %
	3994677,55	0,01682	0,00901		90,32 %
	4225808,18	**0,01201**	**0,00000**		**100,00 %**
	4456938,80	0,00000	0,00000		0,00 %
Mín.	0,00	1,00000	1,00000	0,352423595	83,33 %
	370519,43	0,58559	0,87387		77,01 %
	741038,87	0,46907	0,73874		76,05 %
	1111558,31	0,32132	0,56156		74,10 %
	1482077,75	0,18619	0,23724		79,69 %
	1852597,19	0,09009	0,04204		91,46 %
	2223116,63	0,02402	0,01201		90,91 %
	2593636,07	**0,01502**	**0,00300**		**96,15 %**
	2964155,51	0,00000	0,00300		0,00 %
	3334674,95	0,00000	0,00000		0,00 %



(a) Distancia euclidiana usando la media como métrica de combinación.



(b) Distancia euclidiana usando la distancia máxima como métrica de combinación.



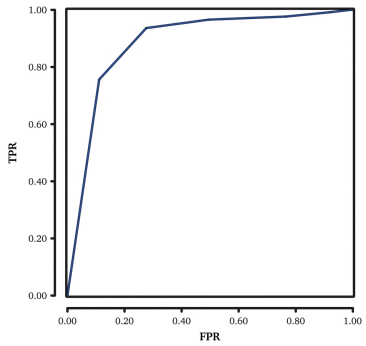
(c) Distancia euclidiana usando la distancia mínima como métrica de combinación.

Figura 5.20: Curvas ROC que representan el uso de las características y los permisos utilizando como medida la distancia del coseno. En este caso, los resultados obtenidos son realmente bajos. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.16.

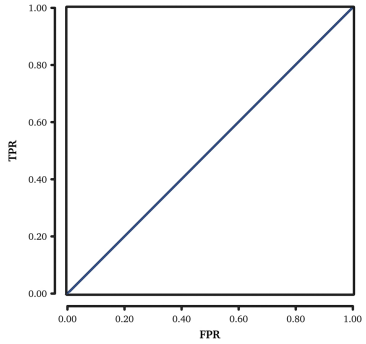
.

Tabla 5.17: Resultados obtenidos de las cadenas, las características y permisos utilizando la distancia del coseno como combinador.

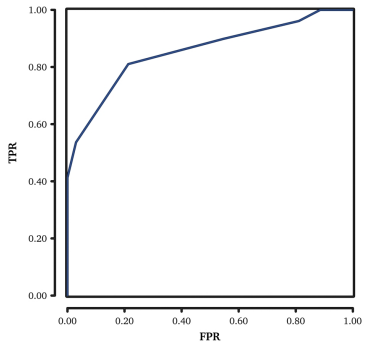
Comb.	Umbral	TPR	FPR	AUC	PRECISIÓN
Media	0,83693246	1,00	1,00	**0,884240096**	50,00 %
	0,85505219	1,00	1,00		50,00 %
	0,87317191	1,00	0,99		50,30 %
	0,89129164	1,00	0,99		50,45 %
	0,90941137	0,99	0,92		53,63 %
	0,92753109	0,98	0,77		60,54 %
	0,94565082	0,97	0,50		73,51 %
	0,96377055	**0,94**	**0,28**		**83,00 %**
	0,98189027	0,76	0,11		82,25 %
	1,00001000	0,00	0,00		50,00 %
Máx.	**1,00000000**	**1,00**	**1,00**	0,500902705	**50,09 %**
	1,00000111	0,00	0,00		50,00 %
	1,00000222	0,00	0,00		50,00 %
	1,00000333	0,00	0,00		50,00 %
	1,00000444	0,00	0,00		50,00 %
	1,00000556	0,00	0,00		50,00 %
	1,00000667	0,00	0,00		50,00 %
	1,00000778	0,00	0,00		50,00 %
	1,00000889	0,00	0,00		50,00 %
	1,00001000	0,00	0,00		50,00 %
Mín.	0,00000000	1,00	1,00	0,855783712	50,00 %
	0,11111222	1,00	0,95		52,40 %
	0,22222444	1,00	0,92		54,20 %
	0,33333667	1,00	0,89		55,71 %
	0,44444889	0,96	0,81		57,51 %
	0,55556111	0,90	0,55		67,60 %
	0,66667333	**0,81**	**0,21**		**79,85 %**
	0,77778556	0,54	0,03		75,32 %
	0,88889778	0,41	0,00		70,63 %
	1,00001000	0,00	0,00		50,00 %



(a) Distancia del coseno usando la media como métrica de combinación.



(b) Distancia del coseno usando la distancia máxima como métrica de combinación.



(c) Distancia del coseno usando la distancia mínima como métrica de combinación.

Figura 5.21: Curvas ROC que representan el uso de las características y los permisos utilizando como medida la distancia del coseno. En este caso, la combinación a través de la media y la mínima, obtiene un área bajo la curva cercana al 0,9 y una precisión cercana al 98 %. Los datos correspondientes a este gráfico se pueden consultar en la Tabla 5.17.

.

Algoritmos utilizados	*TPR*	*FPR*	*AUC*	*Precisión*
SimpleLogistic	0,91	0,05	0,97	93,26 %
NaiveBayes	0,93	0,17	0,90	88,07 %
BayesNet K2	0,71	0,13	0,89	78,68 %
BayesNet TAN	0,83	0,11	0,94	86,09 %
SMO Poly	0,93	0,03	0,95	94,70 %
SMO NPoly	0,77	0,04	0,86	86,45 %
IBK 1	0,35	0,02	0,84	66,24 %
IBK 3	0,22	0,03	0,82	59,85 %
IBK 5	0,17	0,03	0,79	56,75 %
IBK 10	0,08	0,04	0,77	51,77 %
J48	0,83	0,12	0,86	85,54 %
RandomTree	0,79	0,14	0,81	82,45 %
RandomForest 10	0,92	0,13	0,96	89,74 %
RandomForest 50	0,93	0,09	0,97	91,81 %
RandomForest 100	0,94	0,09	0,97	92,04 %

Tabla 5.18: Resultados obtenidos al utilizar las cadenas de caracteres como predictor de *malware*.

es cierto que los pertenecientes a la familia de algoritmos bayesianos obtienen un resultado sensiblemente peor.

En general, obtenemos un ratio de detección en torno al 85 %, aunque es cierto que los valores de AUC son más variados. Podemos concluir que el mejor resultado se obtiene con el algoritmo de *Random Forests*, con una población de 100 árboles. Con ello, se consigue una detección del 87,41 % y un AUC de un 0,95.

Características

Los resultados obtenidos en el análisis de las características están reflejados en la Tabla 5.20. En ellos podemos apreciar que la precisión de los modelos generados sobre las características es sensiblemente mejor que el de los permisos como predictor de *malware*, aunque es cierto que los valores del área bajo la curva ROC son sensiblemente inferiores.

Algoritmos utilizados	*TPR*	*FPR*	*AUC*	*Precisión*
SimpleLogistic	0,77	0,07	0,92	84,88 %
NaiveBayes	0,87	0,42	0,77	72,78 %
BayesNet K2	0,90	0,41	0,79	74,49 %
BayesNet TAN	0,91	0,41	0,85	74,99 %
SMO Poly	0,74	0,07	0,83	83,34 %
SMO NPoly	0,79	0,05	0,87	87,14 %
IBK 1	0,80	0,06	0,93	87,28 %
IBK 3	0,77	0,10	0,93	83,77 %
IBK 5	0,76	0,10	0,92	83,14 %
IBK 10	0,73	0,20	0,89	76,65 %
J48	0,72	0,10	0,87	80,90 %
RandomTree	0,77	0,07	0,89	85,13 %
RandomForest 10	0,80	0,06	0,94	87,07 %
RandomForest 50	0,79	0,05	0,95	87,28 %
RandomForest 100	0,80	0,05	0,95	87,41 %

Tabla 5.19: Resultados obtenidos al utilizar los permisos como predictor de *malware*.

A la vista de estos resultados podemos concluir que las características que se declaran en la aplicación son al menos, tan significativas como los permisos a la hora de determinar si una muestra es maliciosa o no. Sin embargo, esta característica por si sola no es suficiente, ya que al no ser un atributo obligatorio puede darse el caso de que la aplicación no contenga ningún atributo de este tipo. Por ello, el siguiente paso es la combinación de ambos atributos.

Permisos y características combinados

Una vez obtenidos los resultados tanto de los permisos como de las características, vamos a combinar la información otorgada por ambos. De esta manera, seremos capaces de obtener información que, de forma separada, no sería posible inferir. Los resultados se pueden ver en la Tabla 5.21

La combinación de ambos ofrece unos resultados de un 0,98 de área bajo la curva ROC y de un 94,83 % de precisión, como

Algoritmos utilizados	*TPR*	*FPR*	*AUC*	*Precisión*
SimpleLogistic	0,84	0,13	0,88	85,75 %
NaiveBayes	0,81	0,18	0,87	81,38 %
BayesNet K2	0,84	0,13	0,88	85,30 %
BayesNet TAN	0,84	0,12	0,88	85,89 %
SMO Poly	0,84	0,12	0,86	86,10 %
SMO NPoly	0,83	0,11	0,86	85,66 %
IBK 1	0,82	0,11	0,90	85,34 %
IBK 3	0,81	0,10	0,91	85,73 %
IBK 5	0,82	0,11	0,91	85,45 %
IBK 10	0,82	0,12	0,91	85,38 %
J48	0,82	0,08	0,90	87,09 %
RandomTree	0,82	0,11	0,89	85,22 %
RandomForest 10	0,82	0,10	0,91	85,99 %
RandomForest 50	0,82	0,10	0,91	86,06 %
RandomForest 100	0,82	0,10	0,91	86,09 %

Tabla 5.20: Resultados utilizando las características como predictores de *malware*.

resultado más destacado. Esta da una idea de la capacidad de complementarse que tienen ambos atributos a la hora de predecir si una aplicación es maliciosa o no lo es.

Permisos, características y cadenas combinados

Una vez obtenidos los resultados parciales, vamos a comprobar cuál es la mejor predicción que se puede obtener combinando los 3 predictores. Los resultados se pueden ver en la Tabla **??**.

Los mejores resultados se obtuvieron combinando las 3 con RF 100: $\texttt{AUC} = 0,98$ *y* $\texttt{Precisión} = 94,83\,\%$

En concreto, los mejores resultados son los que se obtienen con el algoritmo de *Simple Logistic*, con un área por debajo de la curva ROC de 0,99 y con una precisión de 95,58 %. Del resto de los valores, destaca el resultado obtenido por el conjunto de configuraciones de *Random Forest*, con valores de área bajo la curva de 0,99 pero con una precisión algo menor. En el otro extremo, el algoritmo IBK apenas obtiene una valor de área bajo la curva de 0,77.

Algoritmos utilizados	*TPR*	*FPR*	*AUC*	*Precisión*
SimpleLogistic	0,95	0,07	0,97	93,96 %
NaiveBayes	0,86	0,10	0,91	88,05 %
BayesNet K2	0,89	0,10	0,93	89,28 %
BayesNet TAN	0,92	0,10	0,96	91,11 %
SMO Poly	0,95	0,07	0,94	94,07 %
SMO NPoly	0,95	0,06	0,94	94,50 %
IBK 1	0,96	0,06	0,97	95,22 %
IBK 3	0,97	0,11	0,98	92,85 %
IBK 5	0,96	0,12	0,97	91,91 %
IBK 10	0,96	0,14	0,97	90,82 %
J48	0,91	0,10	0,93	90,60 %
RandomTree	0,92	0,08	0,90	91,62 %
RandomForest 10	0,94	0,06	0,98	93,96 %
RandomForest 50	0,95	0,05	0,98	94,79 %
RandomForest 100	0,94	0,05	0,98	94,83 %

Tabla 5.21: Resultados obtenidos de las utilización de las características y los permisos combinados como predictores de *malware*.

5.7 CONCLUSIONES DEL ANÁLISIS ESTÁTICO

A la vista de estos resultados, podemos concluir que el uso de algoritmos de aprendizaje automático es una herramienta muy poderosa para el análisis estático de las aplicaciones de la plataforma Android, en concreto, para la detección de aplicaciones maliciosas.

En este apartado hemos visto la selección de distintos atributos para poder generar un modelo lo más eficiente posible en la detección de aplicaciones maliciosas en la plataforma. Para ello hemos aplicado dos técnicas distintas. La primera es una detección de anomalías. Concretamente, determinando cuáles son los atributos que tienen las aplicaciones cuyos fines no son maliciosos, hemos trazado umbrales por encima de los cuales, se puede considerar que una aplicación tiene un comportamiento malicioso. Para ello, en primer lugar hemos convertido los atributos en vectores y hemos medido la distancia entre lo que se considera un comportamiento normal y la muestra analizada. Para medir esta distancia hemos utilizado tres métricas distintas: (i)

Algoritmos utilizados	*TPR*	*FPR*	*AUC*	*Precisión*
SimpleLogistic	0,95	0,04	0,99	95,58 %
NaiveBayes	0,93	0,17	0,91	88,22 %
BayesNet K2	0,72	0,12	0,91	79,98 %
BayesNet TAN	0,86	0,10	0,96	87,65 %
SMO Poly	0,98	0,03	0,98	97,51 %
SMO NPoly	0,84	0,02	0,91	91,02 %
IBK 1	0,98	0,07	0,96	95,72 %
IBK 3	0,79	0,14	0,93	82,59 %
IBK 5	0,45	0,16	0,81	64,24 %
IBK 10	0,08	0,04	0.77	51.77 %
J48	0,91	0,07	0,91	92,15 %
RandomTree	0,86	0,13	0,86	86,44 %
RandomForest 10	0,99	0,11	0,98	93,60 %
RandomForest 50	0,99	0,09	0,99	95,03 %
RandomForest 100	0,99	0,09	0,99	95,17 %

Tabla 5.22: Resultados obtenidos de las utilización de las características, los permisos y las cadenas de caracteres combinados como predictores de *malware*.

distancia euclidiana, (ii) distancia Manhattan y (iii) distancia del coseno. Para combinar la distancia con el conjunto de entrenamiento de *goodware* hemos utilizado tres métricas distintas, la distancia media, la máxima y la mínima.

Por otro lado, hemos utilizado algoritmos de aprendizaje automático para que, empleando estos atributos, se puedan construir modelos que clasifiquen las aplicaciones como maliciosas o no. En concreto, hemos utilizado regresión logística, redes bayesianas, máquinas de soporte vectorial, KNN, árboles C.45 y *Random Forests*, todos ellos con distintas configuraciones, para determinar cuál es el que ofrece un mejor resultado.

En cuanto a la selección de atributos, nos hemos centrado en tres de ellos principalmente. Los permisos de la aplicación son aquellos que definen a qué partes sensibles del *smartphone* intenta acceder la aplicación. Por otro lado, están las características. Éstas indican distintos atributos del terminal a las que la aplicación hace referencia. Esta información se encuentra principalmente orientada a dotar de contenido semántico a las tiendas de aplicaciones y a los terminales sobre las capacidades y necesida-

des de la aplicación. Nosotros hemos utilizado estas características, tanto solas como en combinación con los permisos, para complementar la información que estos últimos nos ofrecen. La ventaja de estos dos atributos es que se encuentran dentro del fichero `AndroidManifest.xml`, por lo que el proceso de extracción es muy sencillo. Finalmente, hemos utilizado las cadenas de caracteres presentes dentro de la aplicación, como atributo para la detección del *malware*.

Aplicando nuestra metodología de detección de comportamientos anómalos, hemos obtenido que la mejor combinación se consiga mediante la utilización de las características y los permisos como atributo, la distancia del coseno como combinador y la media. Con esta configuración hemos obtenido un área bajo la curva ROC de 0,91 con una precisión del 89,04 % con 5 umbrales distintos.

En cuanto a la utilización de algoritmos de aprendizaje automático, a fin de determinar cuál es el modelo más efectivo, se han generado distintos modelos aplicando distintas técnicas con diferentes configuraciones. Para la selección de estas características, se han seleccionado algoritmos que ya han probado su efectividad en la detección de *malware* en otras plataformas [SBN+10].

Primeramente, hemos analizado la utilización de los permisos como predictores de *malware*. Con ello, hemos conseguido el mejor resultado con el algoritmo kNN con $k = 1$, en el que hemos alcanzado una precisión del 87,28 % y un valor Área bajo la curva ROC de 0,93. Posteriormente hemos utilizado los mismos modelos aplicando las características de las aplicaciones. En este caso, hemos conseguido el mejor resultado para la precisión con el algoritmo de árboles J48 con una precisión del 86,09 %, mientras que el mayor valor para el área de la curva ROC lo hemos obtenido con el algoritmo Random Forest en todas sus configuraciones e kNN con valores de $k = 3$, $k = 5$, $k = 10$, con un valor de 0,91.

Finalmente, hemos combinado ambos atributos consiguiendo el mejor resultado de precisión para IBK con valor de $K = 1$, con un valor de 95,22 %. El mayor valor de área bajo la curva ROC para los algoritmos de IBK con el valor $k = 3$ y para Random Forest para todas sus configuraciones, con un valor de 0,98. En concreto, con la configuración de 100 árboles, conseguimos una precisión del 94,83 %.

A la luz de estos resultados, podemos determinar que el algoritmo kNN obtiene un muy buen resultado en este ámbito.

Además, este algoritmo, dada su baja complejidad computacional, se adecúa perfectamente a las capacidades de este tipo de terminales, por lo que es la opción más adecuada para obtener un sistema de detección de *malware* de forma comercial en estos dispositivos. Por otro lado, el método desarrollado para la detección de anomalías arroja unos resultados sensiblemente peores que los de los algoritmos de aprendizaje automático.

Sin embargo, pese a lo bueno de los resultados obtenidos, no podemos detenernos aquí. A la luz de las investigaciones realizadas por Zhau y Jiang [ZJ12a], el estado del *malware* sobre la plataforma a día de hoy es todavía muy primitivo. Según sus investigaciones, el 86 % de las muestras analizadas se trataba de software benigno al que se le añadió comportamiento malicioso, siendo el propio usuario el que lo instalaba en su sistema. Esto hace pensar que permisos como el de enviar SMS sean muy demandados por las aplicaciones infectadas, con el fin de mandar mensajes de coste superior y obtener dinero con ello. Este tipo de características hace que los sistemas de aprendizaje automático tengan un buen rendimiento en la detección de aplicaciones maliciosas.

Si atendemos a la evolución que este tipo de aplicaciones han tenido en otras plataformas [WN07], el comportamiento malicioso evolucionará hacia sistemas más complejos, por lo que es necesario mejorar el estado de la técnica actual en el análisis dinámico de este tipo de aplicaciones para anticiparnos a esta evolución.

6 CONCLUSIONES

«No importa lo bonita que sea tu suposición, no importa lo listo que fuera el que realizó la suposición o cuál fuera su nombre. Si no concuerda con los experimentos . . . es errónea.»

R. Feynman
(1918 – 1988)

En los tiempos en los que vivimos, los teléfonos inteligentes o *smartphones* se han convertido en una herramienta indispensable. Podemos definir estos terminales como «un teléfono móvil que, gracias a una alta capacidad de cómputo y al hardware especializado de que dispone, es capaz de realizar funciones avanzadas en movilidad». El crecimiento experimentado por este tipo de terminales en los últimos años ha sido exponencial, siendo ésta una tendencia que no tiene visos de decrecer.

Esta explosión ha coincidido con el crecimiento de las redes sociales que, en combinación con este tipo de terminales, ha permitido a los usuarios compartir una gran cantidad de información en tiempo real. La mayor conectividad (por ejemplo, *bluetooth* o WiFi) es otra de las características que más se han destacado en estos dispositivos. Es decir, cada vez aparecen nuevos terminales con características más avanzadas, gobernados por sistemas operativos cada vez más complejos.

Estos avances no están exentos de peligros. La complejidad añadida a los sistemas y a su alta conectividad pueden provocar situaciones en las que la seguridad del terminal y la privacidad del usuario se encuentren comprometidas. Esta investigación doctoral se centra en abordar la problemática presentada en este nuevo escenario en el que los usuarios poseen un sistema con capacidades similares a los de ordenadores de sobremesa, pero sin ser conscientes de los peligros que conllevan.

HIPÓTESIS Y OBJETIVOS

A la luz de este contexto, la hipótesis de partida es la siguiente:

> *«Es posible, mediante el uso de algoritmos supervisados de inteligencia artificial y minería de datos, hacer una gestión inteligente, automática, y efectiva de la seguridad en las aplicaciones de los teléfonos smartphones tal que libere al usuario de parte de la responsabilidad de la gestión de la seguridad del mismo.»*

Con esta hipótesis iniciamos el proceso investigador, centrado en conseguir los siguientes objetivos:

1. Conocer y categorizar la información que se almacena en los dispositivos móviles inteligentes o *smartphones*.

2. Identificar las amenazas de seguridad que se ciernen sobre estos dispositivos.

3. Desarrollar metodologías y procedimientos que sirvan para proteger estos dispositivos sin necesidad de la intervención del usuario.

Con este punto de partida hemos iniciado la investigación separada en dos etapas. En la primera de ellas, el objetivo era conocer al detalle las amenazas y ataques a los que se enfrentan este tipo de dispositivos. Posteriormente, una vez identificadas las mayores amenazas, planteamos metodologías y técnicas que permiten mitigar estas problemáticas seleccionadas, sin que por ello sea necesaria la intervención del usuario o su conocimiento tácito en materia de seguridad.

CONTRIBUCIONES PRINCIPALES

Este trabajo doctoral ha dado como resultado cuatro contribuciones principales, las cuales avanzan en el estado de la técnica existente en el ámbito de la seguridad en los *smartphones*.

En la primera de ellas, se ha desarrollado un novedoso enfoque para el modelado de amenazas. Posteriormente, a fin de validar este enfoque, lo hemos aplicado al ámbito de los teléfonos inteligentes, dando lugar a un nuevo modelado de amenazas aplicado a este nuevo paradigma surgido en los últimos años.

En segundo lugar, para alimentar este modelado, hemos desarrollado un banco de ataques que identifica las mayores amenazas, ataques y vulnerabilidades a las que se enfrentan este tipo de dispositivos. Hasta donde llega nuestro conocimiento, es la primera colección de estas características que se realiza, dotando de un importante valor añadido al modelado desarrollando. De esta forma, dotamos a la comunidad científica de una importante herramienta para valorar las amenazas que sufren este tipo de terminales.

Tras analizar los resultados arrojados por la aplicación de este nuevo banco de ataques al modelo desarrollado, decidimos ir un paso más allá y mitigar a alguna de las amenazas más importantes a las que se enfrentan los terminales, y así contribuir a la seguridad de este tipo de dispositivos. Para ello, nos centramos en todos aquellos aspectos relacionados con el software del terminal y, más concretamente, con las aplicaciones con fines maliciosos.

En tercer lugar, para llevar a cabo la validación de la siguiente contribución, nos centramos en la plataforma Android. Por ello, en un primer paso, extraer las características que nos permitan representar de forma fidedigna este tipo de aplicaciones. De esta forma, otra aportación de este trabajo doctoral es la identificación de los elementos que permitan esta caracterización de aplicaciones dentro de la plataforma Android.

Finalmente, con las características más relevantes extraídas de las aplicaciones, diseñamos dos aproximaciones para detectar aplicaciones maliciosas en esta plataforma, lo cuál se demostró, según el modelado creado, como una de las mayores amenazas a las que se enfrentan estos dispositivos.

La primera de ellas se basa en la detección de las anomalías. Este enfoque determina, en base a las características extraídas, el comportamiento normal de una aplicación, clasificando como malicioso todo aquello que se desvía de lo que hemos clasificado como normal, con cierto umbral de certidumbre.

El segundo enfoque se basa en modelos de aprendizaje automático clásicos. Para alimentarlos se han utilizado las atributos definidos anteriormente para caracterizar las aplicaciones. En este trabajo doctoral se han utilizado las características identificadas anteriormente.

A continuación profundizaremos en cada una de las aportaciones realizadas.

Nuevo enfoque de modelado de amenazas en smartphones

La primera contribución que se realiza en la investigación doctoral es el nuevo modelado de amenazas, el cual está centrado en este tipo de dispositivos. Para el desarrollo de este modelo nos hemos basado en el concepto de caminos de ataque, que define las distintas rutas que se producen desde el atacante a la víctima, generando un grafo que permite, entre otras cosas, clasificar las amenazas a las que se enfrentan los usuarios de este tipo de terminales.

El grafo está compuesto por distintos tipos de nodos. En concreto, los nodos pueden representar activos, vulnerabilidades, amenazas, ataques o valores. A fin de ponderar cada uno de los nodos, este modelo explota dos tipos de información para llevar a cabo esta clasificación. En primer lugar, explota el conocimiento experto, a través del cual se les dota de peso a cada uno de los nodos del camino. Para extraer este conocimiento se ha llevado a cabo la creación de una guía que permita a los expertos volcar su conocimiento de una forma ordenada. Esta se creó basándose en CVSS, que es una guía utilizada en el ámbito comercial para clasificar las vulnerabilidades de las aplicaciones.

Posteriormente, se aplica el algoritmo *PageRank* para extraer información del grafo. Mediante el mismo, se reparte el peso de cada uno de los nodos en función de las relaciones con cada uno de los nodos a los que está conectado. De esta manera, aquel nodo que está relacionado con más nodos tiene un peso mayor dentro del grafo.

Como resultado de esto, tenemos un grafo dirigido cuyos nodos están ponderados en función del conocimiento experto y de su peso dentro del grafo. Con ello, podemos establecer una clasificación de cada uno de los nodos, función de los valores de los pesos de estos nodos.

A fin de ajustar el modelo a las necesidades del usuario, existe un tipo de nodos, denominados valores, cuyo peso es ponderado de forma manual. Estos nodos «valores» están directamente relacionados con los activos a proteger, de tal forma que la información facilitada por el usuario se propaga por toda la red.

Banco de ataques en smartphones

Para alimentar este modelo se ha desarrollado una biblioteca de ataques. En ella hemos detectado una gran cantidad de ataques, vulnerabilidades y amenazas a las que este tipo de terminales se

enfrenta cada día. Por resumir en números, hemos identificado 3 grandes grupos de activos, 32 ataques distintos, 12 vulnerabilidades y 16 amenazas distintas. No ha sido posible desarrollar una taxonomía, ya que nos encontramos con elementos que aparecen en más de una categoría, por lo que se trata de una categorización.

Se han identificado tres tipos de activos. En primer lugar, los datos que almacena el dispositivo, de distinto carácter y naturaleza. En segundo lugar, el *software* que gobierna el dispositivo. Finalmente, el *hardware*, incluyendo todos aquellos sensores y dispositivos que amplían la funcionalidad del terminal.

Las amenazas han sido clasificados en tres categorías distintas: (i) las amenazas relacionadas con las aplicaciones, (ii) las relacionadas con la conectividad del dispositivo y, (iii) las amenazas relacionadas con los aspectos físicos.

Selección de características representativas de las aplicaciones de Android

La siguiente etapa dentro del trabajo doctoral es el desarrollo de nuevos métodos que permitan la detección de aplicaciones con intenciones maliciosas. Como plataforma para la validación elegimos Android por varios motivos. El primero de ellos es que se trata de una plataforma abierta y con un gran crecimiento en el mercado. Por otra parte, el *malware* se ha presentado como uno de los mayores problemas de la plataforma, desde el punto de la seguridad.

Una vez seleccionada la plataforma, pasamos a seleccionar los atributos que permitan la caracterización de una aplicación. En concreto, nos hemos centrado en tres de ellos principalmente: (i) los permisos, (ii) las características y (iii) las cadenas de caracteres. Los permisos de la aplicación son aquellos que definen los elementos a los que debe acceder la aplicación para un correcto funcionamiento. Por otro lado, están las características, que indican distintas particularidades del terminal a las que la aplicación hace referencia. Esta información se encuentra principalmente orientada a dotar de información semántica a las tiendas de aplicaciones y a los terminales sobre las capacidades y necesidades de la aplicación. Finalmente, las cadenas de caracteres presentes dentro de la aplicación indican información relevante, por ejemplo las URLs a las que se conectan.

Una vez seleccionadas las características, iniciamos el desarrollo de los nuevos métodos para el análisis estático de aplicaciones en Android.

Nuevos métodos de detección de malware en smartphones

Para la detección de *malware* en la plataforma, nos hemos centrado en el análisis estático de las aplicaciones. Para ello, hemos desarrollado dos aproximaciones distintas.

La primera es la detección de anomalías. Este método consiste en caracterizar las aplicaciones cuyos fines no son maliciosos, y clasificar todas aquellas aplicaciones que se desvían de esa caracterización como anómalas.

Hemos determinado los valores de las características elegidas en las aplicaciones categorizadas como benignas. Posteriormente, hemos trazado umbrales por encima de los cuales se puede considerar que una aplicación es maliciosa. Para ello, en primer lugar hemos convertido los atributos en vectores dentro del modelo de espacio vectorial y hemos medido la distancia entre lo que se considera un comportamiento normal y la muestra analizada. Para medir esta distancia hemos utilizado tres métricas distintas: (i) distancia euclidiana, (ii) distancia Manhattan y (iii) distancia del coseno. Para combinar la distancia con el conjunto de entrenamiento de *goodware* hemos utilizando tres métricas distintas, la distancia media, la máxima y la mínima.

Por otro lado, hemos utilizado algoritmos de aprendizaje automático para que, utilizando estos atributos, se puedan construir modelos que clasifiquen las aplicaciones como maliciosas o no. En concreto, hemos utilizado las implementaciones presentes en la herramienta WEKA de los siguientes algoritmos: regresión logística, redes bayesianas (*Bayesian Networks*), máquinas de soporte vectorial (denominado *SMV*), K vecinos más cercanos (denominado *IBK*), árboles C.45 (denominado *J.48*) y bosques aleatorios (denominado *Random Forrest*), todos ellos con distintas configuraciones, y con el objetivo de determinar cuál es el que ofrece un mejor resultado.

RESULTADOS OBTENIDOS

Una vez expuestos los métodos desarrollados en el transcurso de la investigación doctoral, analicemos los resultados obtenidos.

Resultados obtenidos en la categorización de amenazas en teléfonos inteligentes

Los resultados que hemos obtenido al aplicar el modelo desarrollado al banco de ataques se han centrado en evaluar cuáles son las principales amenazas a las que se enfrentan estos dispositivos, teniendo en cuenta aquellos valores o aspectos en los que el usuario quiere hacer énfasis a la hora de evaluar estas amenazas (por ejemplo, darle más importancia a la privacidad frente a la disponibilidad). Para ello, analizamos los resultados obtenidos centrándonos en la privacidad. Estos reflejan la importancia de amenazas físicas, como puede ser la pérdida o extravío del dispositivo o el acceso por parte de otras personas. Destacan a su vez aquellas amenazas relacionadas con el canal de transmisión, en concreto a través de redes WiFi no seguras. También se observa la importancia de todos aquellos elementos software relacionados con el terminal, haciendo especial énfasis en dos de ellos: modificación de los datos del teléfono y transmisión de los datos sin conocimiento expreso del usuario.

Por otro lado, se ha analizado el resultado obtenido dándoles a todos los elementos el mismo peso. En este caso, las amenazas relacionadas con el software son las que coparon los primeros puestos del análisis, así como todas aquellas amenazas relacionadas con los datos del terminal. En este caso concreto, se puede deducir que el *malware* es una de las mayores amenazas de seguridad para este tipo del terminales, ya que engloba las principales amenazas con esta segunda configuración.

Este modelo presenta como punto fuerte la posibilidad de evaluar cada uno de los nodos en función de los valores del resto de los nodos de la red. Si bien es cierto que nos hemos centrado en las amenazas, no es menos cierto que esta misma metodología puede ser utilizada para clasificar los ataques o las vulnerabilidades a las que se enfrentan.

Aplicando este modelo al banco de ataques generado, podemos deducir que el activo más amenazado son los datos presentes en el terminal. Dando la misma importancia a todos los valores de entrada, todas aquellas amenazas relacionadas con el software son las que obtienen una posición más alta en la clasificación, lo que indica que son las amenazas más importantes.

Resultados obtenidos en la detección de malware en smartphones

A continuación valoramos los resultados obtenidos en la creación de nuevos modelos para la detección de software malicioso en este tipo de dispositivos. En primer lugar analizamos la generación del conjunto de datos. Posteriormente, hemos visto los resultados obtenidos a través del detector de anomalías desarrollado. Por último, analizamos los resultados obtenidos a través del enfoque de los algoritmos de aprendizaje automático.

Generación del conjunto de datos

En un primer paso, hemos generado dos conjuntos de datos con aplicaciones benignas (*goodware*) y maliciosas (*malware*). La obtención de éstas últimas se hizo a través del servicio de VirusTotal, herramienta que comprueba las muestras con más de 40 motores de detección de distintas casas de antivirus. A fin de obtener un conjunto de datos de software malicioso lo más fidedigno posible, se desarrolló una metodología para la selección de muestras. Ésta se basa en la capacidad de detección de los distintos antivirus. De esta manera se obtuvieron 333 muestras distintas. Para tener un número balanceado de las mismas, el conjunto de datos de software benigno tenía el mismo número de elementos.

Para que el conjunto de datos sea lo más representativo posible, se seleccionó el número de muestras de forma proporcional a las muestras que había de cada categoría. La categorización de los elementos benignos se llevó a cabo consultando a la tienda oficial de Android.

Resultados obtenidos aplicando la detección de anomalías

Dada la gran cantidad de configuraciones posibles, los resultados obtenidos son de lo más variado. En general, existen varias configuraciones cuyos resultados no pueden ser considerados como buenos, mientras que otras se mueven en un rango más cercano al 0,8 de valor del área bajo la curva ROC.

La mejor combinación se consigue mediante la utilización de las características y los permisos como atributo, la distancia del coseno como métrica y como combinador la media. Con esta configuración hemos obtenido un área bajo la curva ROC de un 0,91 con una precisión del 99,64 % con 5 umbrales distintos.

Resultados obtenidos aplicando algoritmos de aprendizaje automático

En cuanto a la utilización de algoritmos de aprendizaje automático, a fin de determinar cuál es el más efectivo, se han generado distintos modelos aplicando distintas técnicas con distintas configuraciones.

Primeramente, hemos analizado la utilización de los permisos como predictores de *malware*. Con ello hemos conseguido el mejor resultado con el algoritmo IBK con $k = 1$, en el que hemos conseguido una precisión del 87,28 % y un valor de área bajo la curva ROC de un 0,93. Posteriormente hemos utilizado los mismos modelos para obtener los valores aplicando las características de las aplicaciones. En este caso, hemos conseguido el mejor resultado para la precisión con el algoritmo de árboles J48 con una precisión del 86,09 %, mientras que el mayor valor para el área de la curva ROC lo hemos obtenido con el algoritmo Random Forest en todas sus configuraciones, y con IBK con valores de $k = 3$, $k = 5$, $k = 10$, con un valor de 0,91.

Finalmente, hemos combinado ambos atributos, consiguiendo el mejor resultado de precisión para IBK con valor de $K = 1$ y el mayor valor de área bajo la curva ROC para los algoritmos de IBK con el valor $k = 3$ y para Random Forest para todas sus configuraciones, con un valor de 0,98. En concreto, con la configuración de 100 árboles, consigue una precisión del 94,83 %.

A la luz de estos resultados, podemos determinar que el algoritmo IBK obtiene un muy buen resultado en este ámbito. Además, este algoritmo, dada su baja complejidad computacional, se adecúa perfectamente a las capacidades de este tipo de terminales, por lo que es la opción más adecuada para desarrollar un sistema de detección de *malware* de forma comercial en estos dispositivos. Por otro lado, el método desarrollado para la detección de anomalías arroja unos resultados sensiblemente peores que los de los algoritmos de aprendizaje automático.

LIMITACIONES IDENTIFICADAS

Hemos identificado limitaciones tanto en el modelado de amenazas como en la detección de aplicaciones maliciosas. En lo referente al modelado de amenazas, la primera limitación identificada es su carácter muy dependiente de la biblioteca de ataques. En nuestro caso, la biblioteca ha sido generada tras una

larga investigación sobre los distintos eventos que este tipo de dispositivos han sufrido en los últimos años. Sin embargo, esta biblioteca es mejorable.

Por otro lado, el otro pilar del modelo es el algoritmo que evalúa la combinación de nodos de amenazas. Este se calcula como la suma de los pesos de todos aquellos nodos a los que un nodo está conectado. Sin embargo, es posible mejorar este algoritmo de combinación, explotando más la información proveniente del grafo generado.

En lo que respecta a la detección de *malware*, la limitación más importante es el propio estado del *malware* en este momento. Según investigaciones recientes [ZJ12b], el 82 % del *malware* clasificado sobre la plataforma Android está compuesto por aplicaciones benignas en los que las atacantes introducen el comportamiento maligno y se vuelven a subir a alguna de las tiendas disponibles. Esto implica que el software malicioso disponible es muy similar al software benigno, lo que dificulta su identificación. Según se avance en el desarrollo de nuevas técnicas, los modelos aquí presentados perderán efectividad.

TRABAJO FUTURO

Esta tesis doctoral es el comienzo de una investigación de mayor recorrido.

Por un lado, respecto al modelado de amenazas, una de las limitaciones identificadas se encuentra en el banco de ataques. Para mejorar esta limitación, nos encontramos trabajando en la liberación de una herramienta que permita a la comunidad alimentar esta biblioteca de ataques. De esta forma, conseguimos mejorar los resultados de la investigación y ayudar a identificar a la sociedad, tanto investigadora como técnica, cuáles son las amenazas a las que se enfrentan a la hora de usar estos dispositivos.

Por otro lado, como hemos comentado en las debilidades, el algoritmo que combina los pesos de los nodos es uno de los pilares del modelo. Estamos investigando cómo mejorar este algoritmo, a fin de explotar toda la información posible y obtener una clasificación de las amenazas aún más potente.

Finalmente, el cálculo de los pesos en base al conocimiento experto requiere de una vigilancia constante para adaptarse a los cambios que cada situación va requiriendo. Es preciso realizar

esta labor para poder obtener clasificaciones de las amenazas rigurosas.

En lo referente a las líneas futuras de trabajo en la detección estática de muestras, debemos profundizar en la selección de las características del ejecutable. Si bien es cierto que éstas han demostrado ser eficientes con el estado del *malware* actual, la continua evolución de éste hace que sea necesaria la constante revisión de estas características.

Por otro lado, dado el crecimiento del volumen de *malware*, es necesario aplicar distintos enfoques que permitan reducir el volumen del conjunto de datos. Los modelos aquí presentados funcionan mejor cuanto mayor sea el conjunto de entrenamiento. Otros enfoques, como el aprendizaje semi-supervisado o el aprendizaje clase única, requieren de un volumen menor de conjunto de entrenamiento para llevar a cabo la clasificación. Por ello, y en previsión del crecimiento que va a experimentar este tipo de *malware*, es necesario explorar estas aproximaciones.

Además, es necesario explorar otros enfoques. Particularmente, el análisis dinámico debe ser un área donde se debe profundizar, ya que la complejidad de este tipo de software requiere de análisis más exhaustivos que el que se realiza de manera estática.

CONCLUSIONES

El desarrollo de los terminales móviles o *smartphones* en los últimos años ha sido exponencial. Nos encontramos ante una auténtica revolución en la que los usuarios no sólo ven aumentadas sus posibilidades de comunicación en cualquier lugar, sino que también ven mejorada su interacción con la realidad.

A día de hoy, millones de usuarios poseen un terminal dotado de componentes hardware muy sofisticados y con una capacidad de cálculo que hace unos pocos años no existía ni en los terminales de sobremesa. Además, al rededor de estas plataformas se están desarrollando ecosistemas de aplicaciones que permiten ampliar la funcionalidad de estos terminales hasta límites aún hoy insospechados.

Esta investigación doctoral supone un importante avance en la seguridad de este tipo de dispositivos, tanto desde el punto de vista teórico como del practico.

Desde el punto de vista teórico, el modelado de amenazas desarrollado, alimentado desde el nuevo banco de ataques, supone una nueva aportación a la categorización de las amenazas

y, más concretamente, a las que se enfrentan los terminales móviles. Con esta aportación buscamos crear un nuevo marco que permita determinar cuáles son las amenazas a las que se enfrentan estos dispositivos, dado el creciente numero de ataques que se están produciendo actualmente. En este contexto, hay que poner en valor la labor realizada en el banco de ataques, ya que supone una profunda revisión de todos aquellos aspectos que afectan a este tipo de terminales. Este banco de ataques supone un punto de partida para completar con aquellos aspectos que, dada la cambiante naturaleza de la tecnología hoy en día, surjan en los próximos años.

Esta aportaciones han dado como resultado una valoración sobre cuál es uno de los mayores peligros a los que se enfrentan estos dispositivos. Si bien es cierto que el hardware del que disponen es abundante, es en el software del mismo donde se centran gran parte de las amenazas a las que se enfrentan. La complejidad que han alcanzado los sistemas operativos de estos dispositivos, así como el rico ecosistema de aplicaciones del que disponen, hace que las amenazas a las que se enfrentan sean mucho mayores que antaño, cuando se utilizaban prácticamente para realizar llamadas y recibir mensajes y la instalación de las aplicaciones era realmente complicada.

Desde el punto de vista práctico, y teniendo en cuenta el análisis de la situación actual realizado, los dos enfoques planteados suponen un buen punto de partida para la situación actual del *malware*. Estos enfoques consiguen buenos ratios de detección con las muestras existentes hoy en día. Análisis realizados a múltiples muestras reflejan que la complejidad de estas es más bien escasa. Dadas las medidas de seguridad que plantean las distintas plataformas, los creadores de este tipo de software no emplean mucho tiempo en el desarrollo del mismo, ya que consiguen un buen rédito económico con un mínimo esfuerzo.

Sin embargo, esta situación está cambiando. Las distintas plataformas están realizado importantes esfuerzos en garantizar la seguridad y la confiabilidad de sus plataformas. Esto provocará que los creadores de *malware* realicen creaciones mucho mas complejas, las cuales dificultaran las tareas de detección. Es por ello que los aportaciones realizadas en este trabajo doctoral deben ser considerados como un punto de partida dentro de una carrera de fondo que, sin duda, se disputará en los próximos años.

CONSIDERACIONES FINALES

Hace 15 años no existía Google. Hace 10 años no existía Facebook. Hace 5 años no existía lo que conocemos hoy como teléfono inteligente. Y pese a su juventud, gran parte de la población tiene todos estos elementos como una constante en su día a día. La forma en la que hemos agregado nuevos conceptos, como por ejemplo hacer «check-in» o «twittear», a nuestra vida cotidiana demuestra que esta revolución ha llegado para quedarse.

Estamos en medio de una carrera hacia adelante en la que no nos paramos y mirar atrás; parar a pensar en las consecuencias que tienen y tendrán todos estos cambios que se producen a un ritmo vertiginoso, en la que los *smartphones* se han convertido en un parte importante de ellos.

La realidad es que nos encontramos con dispositivos portátiles de pequeño tamaño con una gran capacidad de cómputo en los que almacenamos gran parte de nuestra información. Toda esa información, nuestras fotos, nuestros contactos, están en un dispositivo de un tamaño minúsculo que es fácil de perder o robar, que se puede conectar a mil fuentes de información, las cuales tienen vulnerabilidades conocidas y son explotables, quedando a la espera de que un atacante le saque provecho.

La percepción de falta de seguridad que durante los últimos años se ha desarrollado en el entorno de escritorio aún falta en el entorno del terminal móvil. Es por ello que, desde la comunidad científica y desde la industria especializada, debemos aunar esfuerzos en el desarrollo de herramientas que permitan reducir la distancia entre la percepción de seguridad y la situación real.

Sin embargo, estas herramientas nunca serán eficientes si las personas que las manejan no son conscientes del peligro. Desde el ámbito universitario, en virtud del compromiso ético adquirido con la sociedad, es necesario impulsar esta formación de los usuarios para poder disfrutar de este terremoto de información al que nos enfrentamos hoy en día.

Citando a Bruce Schneier, «si piensas que la tecnología puede solucionar tus problemas de seguridad, está claro que ni entiendes los problemas ni entiendes la tecnología». En esta investigación doctoral he avanzado en el entendimiento de los problemas y en la generación de conocimiento de las tecnologías utilizadas, estando de esta forma un poco más cerca de ayudar a solucionar algunos de estos problemas.

A partir de ahora queda un largo y emocionante camino por delante.

«*Stay Hungry. Stay Foolish.*»

S. Jobs (1955 – 2011)

BIBLIOGRAFÍA

[AL09] O. Amft y P. Lukowicz. From backpacks to smartphones: Past, present, and future of wearable computers. *Pervasive Computing, IEEE*, 8(3):8–13, 2009.

[Alv11] G. Alvarez. Hackonomics: todo sobre la economía sumergida del hacking. *PC World*, (290):48–56, 2011.

[And72] J. P. Anderson. Computer security technology planning study. Informe Técnico ESDTR73-51, Air Force Electronic Systems Division, Hanscom AFB, Bedford, MA, Fort Washington, Pa. 19034, 1972. También disponible como Vol. I, DITCAD-758206. Vol. II DITCAD-772806.

[ANN05] N. Asokan, V. Niemi, y K. Nyberg. Man-in-the-middle in tunnelled authentication protocols. En *Security Protocols*, páginas 28–41. Springer, 2005.

[AW99] S. Amari y S. Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789, 1999.

[BBS95] A. Berman, V. Bourassa, y E. Selberg. Tron: Process-specific file protection for the unix operating system. En *Proceedings of the 1995 Winter USENIX Conference*. 1995.

[BDD+01] J. Bergeron, M. Debbabi, J. Desharnais, M.M. Erhioui, Y. Lavoie, y N. Tawbi. Static detection of malicious code in executable programs. *Int. J. of Req. Eng*, 2001:184–189, 2001.

[BDD+11] S. Bugiel, L. Davi, A. Dmitrienko, T. Fischer, A.R. Sadeghi, y B. Shastry. Poster: the quest for security against privilege escalation attacks on android. En *Proceedings of the 18$^{\text{th}}$ ACM conference on Computer and communications security*, páginas 741–744. ACM, 2011.

[Bel06] C.M. Bishop y SpringerLink (Service en ligne). *Pattern recognition and machine learning*, tomo 4. springer New York, 2006.

[BGLL08] V.D. Blondel, J.L. Guillaume, R. Lambiotte, y E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008:P10008, 2008.

[BHSP08] A. Bose, X. Hu, K.G. Shin, y T. Park. Behavioral detection of malware on mobile handsets. En *Proceeding of the 6th international conference on Mobile systems, applications, and services*, páginas 225–238. ACM, 2008.

[Bib77] K. J. Biba. Integrity considerations for secure computer systems. Informe Técnico MTR-3153, Volume I, MITRE Corp., Bedford, MA, Abril 1977.

[BKvOS10] D. Barrera, H.G. Kayacik, P.C. van Oorschot, y A. Somayaji. A methodology for empirical analysis of permission-based security models and its application to android. En *Proceedings of the 17$^{\text{th}}$ ACM conference on Computer and communications security*, páginas 73–84. ACM, 2010.

[BL73] D. E. Bell y L. J. LaPadula. Secure computer systems: Mathematical foundations. Informe Técnico MTR-2547, Volume I, MITRE Corp., Bedford, MA, 1973.

[BNC+08] T.K. Buennemeyer, T.M. Nelson, L.M. Clagett, J.P. Dunning, R.C. Marchany, y J.G. Tront. Mobile device profiling and intrusion detection using smart batteries. En *Proceedings of the 41$^{\text{st}}$ Annual Hawaii International Conference on System Sciences.*, páginas 296–296. IEEE, 2008.

[BP63] Mr. Bayes y M. Price. An essay towards solving a problem in the doctrine of chances. by the late rev. mr. bayes, frs communicated by mr. price, in a letter to john canton, amfrs. *Philosophical Transactions (1683-1775)*, páginas 370–418, 1763.

[Bre01] L. Breiman. Random forests. *Machine learning*, 45(1):5–32, 2001.

[BSBV11] F. Brezo, I. Santos, P.G. Bringas, y J.L. Val. Challenges and limitations in current botnet detection. En *Database and Expert Systems Applications (DEXA), 2011 22nd International Workshop on*, páginas 95–101. IEEE, 2011.

[BSGBdV11] F. Brezo, I. Santos, P. G. Bringas, y J. L. del Val. Challenges and limitations in current botnet detection. En *In Proceedigns of the 6th International Workshop on Flexible Database and Information System Technology (FlexDBIST'11)*. 2011.

[CAJ07] M. Christodorescu y S. Adviser-Jha. *Behavior-based malware detection*. University of Wisconsin at Madison, 2007.

[Can09] R. Cannings. Android: Securing a mobile platform from the ground up. En *Proceedings of the 18th Usenix Security Symposium*. 2009.

[CB07] Y. Chen y BW Boehm. Stakeholder value driven threat modeling for off the shelf based systems. En *Software Engineering-Companion, 2007. ICSE 2007 Companion. 29th International Conference on*, páginas 91–92. IEEE, 2007.

[CBS05] A. Chaturvedi, S. Bhatkar, y R. Sekar. Improving attack detection in host-based ids by learning properties of system call arguments. En *Proceedings of the IEEE Symposium on Security and Privacy*. Citeseer, 2005.

[CBS07] Y. Chen, B. Boehm, y L. Sheppard. Value driven security threat modeling based on attack path analysis. 2007. ISSN 1530-1605.

[CGH97] E. Castillo, J.M. Gutiérrez, y A.S. Hadi. *Expert systems and probabilistic network models*. Springer Verlag, 1997.

[CH91] G.F. Cooper y E. Herskovits. A bayesian method for constructing bayesian belief networks from databases. En *Proceedings of the Seventh Conference on Uncertainty in Artificial Intelligence*, páginas 86–94. Morgan Kaufmann Publishers Inc., 1991.

[Cha09] A. Chaudhuri. Language-based security on android. En *Proceedings of the ACM SIGPLAN fourth workshop on programming languages and analysis for security*, páginas 1–7. ACM, 2009.

[CJ06] M. Christodorescu y S. Jha. Static analysis of executables to detect malicious patterns. Informe técnico, DTIC Document, 2006.

[CJ08] G.W. Chow y A. Jones. A Framework for Anomaly Detection in OKL4-Linux Based Smartphones. En *Australian Information Security Management Conference*, página 49. Citeseer, 2008.

[CJS+05] M. Christodorescu, S. Jha, S.A. Seshia, D. Song, y R.E. Bryant. Semantics-aware malware detection. En *Security and Privacy, 2005 IEEE Symposium on*, páginas 32–46. IEEE, 2005.

[CLP+11] W.B. Cheon, W.G. Lim, W.H. Park, T.M. Chung, et al. The new vulnerability of service set identifier (ssid) using qr code in android phone. En *Information Science and Applications (ICISA), 2011 International Conference on*, páginas 1–6. IEEE, 2011.

[CNC11] M. Conti, V. Nguyen, y B. Crispo. Crepe: Context-related policy enforcement for android. *Information Security*, páginas 331–345, 2011.

[CO07] NJ Croft y MS Olivier. A silent sms denial of service (dos) attack. *Information and Computer Security Architectures (ICSA) Research Group South Africa*, 2007.

[Coh95] W.W. Cohen. Fast e ective rule induction. En *Proceedings of the 12th International Conference on Machine Learning, Lake Tahoe, California*. 1995.

[CW89] D.D. Clark y D.R. Wilson. A comparison of commercial and military computer security policies. *NIST SPECIAL PUBLICATION SP*, 1989.

[CW00] D.M. Chess y S.R. White. An undetectable computer virus. En *Proceedings of Virus Bulletin Conference*, tomo 5. 2000.

[CWYL07] J. Cheng, S.H.Y. Wong, H. Yang, y S. Lu. Smartsiren: virus detection and alert for smartphones. En *Proceedings of the 5th international conference on Mobile systems, applications and services*, páginas 258–271. ACM, 2007.

[DDSW11] L. Davi, A. Dmitrienko, A.R. Sadeghi, y M. Winandy. Privilege escalation attacks on android. *Information Security*, páginas 346–360, 2011.

[Den76] D.E. Denning. A lattice model of secure information flow. *Communications of the ACM*, 19(5):236–243, 1976.

[DK82] P.A. Devijver y J. Kittler. *Pattern recognition: A statistical approach*. Prentice/Hall International, 1982.

[DKS10] O. Dunkelman, N. Keller, y A. Shamir. A practical-time attack on the a5/3 cryptosystem used in third generation gsm telephony. En *Proceedings of the 30th Annual Cryptology Conference (CRYPTO 2010)*. 2010.

[DP07] M. Dalla Preda. *Code Obfuscation and Malware Detection by Abstract Interpretation*. Tesis Doctoral, PhD thesis, Università degli Studi di Verona Dipartimento di Informatica, 2007. 6, 23, 2007.

[DSW09] L. Davi, A.R. Sadeghi, y M. Winandy. Dynamic integrity measurement and attestation: towards defense against return-oriented programming attacks. En *Proceedings of the 2009 ACM workshop on Scalable trusted computing*, páginas 49–54. ACM, 2009.

[Edd96] S.R. Eddy. Hidden markov models. *Current opinion in structural biology*, 6(3):361–365, 1996.

[EGC+10a] W. Enck, P. Gilbert, B.G. Chun, L.P. Cox, J. Jung, P. McDaniel, y A.N. Sheth. TaintDroid: An Information-Flow Tracking System for Realtime Privacy Monitoring on Smartphones. 2010.

[EGC+10b] W. Enck, P. Gilbert, B.G. Chun, L.P. Cox, J. Jung, P. McDaniel, y A.N. Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. En *Proceedings*

of the 9th USENIX conference on Operating systems design and implementation, páginas 1–6. USENIX Association, 2010.

[Ehr10] D. Ehringer. The dalvik virtual machine architecture, March 2010.

[EKV+05] P. Efstathopoulos, M. Krohn, S. VanDeBogart, C. Frey, D. Ziegler, E. Kohler, D. Mazieres, F. Kaashoek, y R. Morris. Labels and event processes in the asbestos operating system. *ACM SIGOPS Operating Systems Review*, 39(5):17–30, 2005.

[EMJ08] W. Enck, P. McDaniel, y T. Jaeger. Pinup: Pinning user files to known applications. En *Proceedings of the 2008 Annual Computer Security Applications Conference (ACSAC).*, páginas 55–64. IEEE, 2008.

[EOM08] W. Enck, M. Ongtang, y P. McDaniel. Mitigating android software misuse before it happens. 2008.

[EOM09] W. Enck, M. Ongtang, y P. McDaniel. Understanding android security. *IEEE Security & Privacy*, 7(1):50–57, 2009.

[ERS+07] W. Enck, S. Rueda, J. Schiffman, Y. Sreenivasan, L. St Clair, T. Jaeger, y P. McDaniel. Protecting users from themselves. En *Proceedings of the 2007 ACM workshop on Computer security architecture*, páginas 29–36. ACM, 2007.

[ETMLP05] W. Enck, P. Traynor, P. McDaniel, y T. La Porta. Exploiting open functionality in sms-capable cellular networks. En *Proceedings of the 12th ACM conference on Computer and communications security*, páginas 393–404. ACM, 2005.

[FCF09] A.P. Fuchs, A. Chaudhuri, y J.S. Foster. Scandroid: Automated security certification of android applications. *Manuscript, Univ. of Maryland, http://www. cs. umd. edu/˜ avik/projects/scandroidascaa*, 2009.

[FCH+11] A.P. Felt, E. Chin, S. Hanna, D. Song, y D. Wagner. Android permissions demystified. En *Proceedings of the 18th ACM conference on Computer and communications security*, páginas 627–638. ACM, 2011.

[FGG97] N. Friedman, D. Geiger, y M. Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2):131–163, 1997.

[FGW11] A.P. Felt, K. Greenwood, y D. Wagner. The effectiveness of application permissions. En 2^{nd} *USENIX Conference on Web Application Development*, página 75. 2011.

[FHJ52] E. Fix y J.L. Hodges Jr. Discriminatory analysis-nonparametric discrimination: small sample performance. Informe técnico, DTIC Document, 1952.

[FHSL96] S. Forrest, S.A. Hofmeyr, A. Somayaji, y T.A. Longstaff. A sense of self for unix processes. En *Proceedings of the 1996 IEEE Symposium on Security and Privacy.*, páginas 120–128. IEEE, 1996.

[FKC+92] DF Ferraiolo, R. Kuhn, R. Chandramouli, J. Reynaldo, y C. CISM. Role-based access control (rbac). En *Proc. of* 15^{th} *NIST-NSA National Computer Security Conference*. 1992.

[GKF02] V. Gupta, S. Krishnamurthy, y M. Faloutsos. Denial of service attacks at the mac layer in wireless ad hoc networks. En *Proceedings of Military Communications Conference (MILCOM).*, tomo 2, páginas 1118–1123. IEEE, 2002.

[HB06] E. Haselsteiner y K. Breitfuß. Security in near field communication (nfc). En *Workshop on RFID Security RFIDSec*, tomo 6. 2006.

[Hei99] G. Heine. *GSM networks: protocols, terminology, and implementation*, tomo 213. Artech House, 1999.

[HFH+09] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, y I.H. Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[HHT04] T. Harada, T. Horie, y K. Tanaka. Task oriented management obviates your onus on linux. En *Linux Conference*. 2004.

[HLOF03] M. Howard, D. LeBlanc, Safari Tech Books Online, y Safari Books Online (Firme). *Writing secure code*, tomo 2. Microsoft press Redmond, WA, 2003.

[HRU76] M.A. Harrison, W.L. Ruzzo, y J.D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[Idc11] Android explodes in western europe, drives market growth and becomes the biggest smartphone operating system in 4q10, says idc, 03 2011.

[IM07] N. Idika y A.P. Mathur. A survey of malware detection techniques. *Purdue University*, página 48, 2007.

[Jae08] T. Jaeger. Operating system security. *Synthesis Lectures on Information Security, Privacy and Trust*, 1(1):1–218, 2008.

[JD04] G.A. Jacoby y N.J. Davis. Battery-based intrusion detection. En *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, tomo 4, páginas 2250–2255. IEEE, 2004.

[JMD06] G.A. Jacoby, R. Marchany, y N.J. DAVIS. How mobile host batteries can improve network security. *IEEE security & privacy*, 4(5):40–49, 2006.

[Ken83] J.T. Kent. Information gain and a general measure of correlation. *Biometrika*, 70(1):163–173, 1983.

[KFH05] D.K. Kang, D. Fuller, y V. Honavar. Learning classifiers for misuse and anomaly detection using a bag of system calls representation. En *Proceedings from the 6th Annual IEEE SMC Information Assurance Workshop (IAW).*, páginas 118–125. IEEE, 2005.

[KKB+06] E. Kirda, C. Kruegel, G. Banks, G. Vigna, y R. Kemmerer. Behavior-based spyware detection. En *Usenix Security Symposium*. 2006.

[Kle96] EM Kleinberg. An overtraining-resistant stochastic modeling method for pattern recognition. *The annals of statistics*, 24(6):2319–2349, 1996.

[Koh90] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[Koh95] R. Kohavi. A study of cross-validation and bootstrap for accuracy estimation and model selection.

En *International joint Conference on artificial intelligence*, tomo 14, páginas 1137–1145. LAWRENCE ERLBAUM ASSOCIATES LTD, 1995.

[KRV04] C. Kruegel, W. Robertson, y G. Vigna. Detecting kernel-level rootkits through binary analysis. En *Computer Security Applications Conference, 2004. 20th Annual*, páginas 91–100. IEEE, 2004.

[KRVV04] C. Kruegel, W. Robertson, F. Valeur, y G. Vigna. Static disassembly of obfuscated binaries. En *Proceedings of the 13th conference on USENIX Security Symposium*, tomo 13, página 18. 2004.

[KSS08] H. Kim, J. Smith, y K.G. Shin. Detecting energy-greedy anomalies and mobile malware variants. En *Proceeding of the 6th international conference on Mobile systems, applications, and services*, páginas 239–252. ACM, 2008.

[KYB+07] M. Krohn, A. Yip, M. Brodsky, N. Cliffer, M.F. Kaashoek, E. Kohler, y R. Morris. Information flow control for standard os abstractions. En *ACM SIGOPS Operating Systems Review*, tomo 41, páginas 321–334. ACM, 2007.

[KZP07] SB Kotsiantis, ID Zaharakis, y PE Pintelas. Supervised machine learning: A review of classification techniques. *Frontiers in Artificial Intelligence and Applications*, 160:3, 2007.

[Lab] Open Kernel Labs. Ok:android. Accesed 25-03-2011.

[Lab11] Panda Labs. El mercado negro del cibercrimen al descubierto. Informe técnico, Panda Security, 2011.

[Lam74] B.W. Lampson. Protection. *ACM SIGOPS Operating Systems Review*, 8(1):18–24, 1974.

[LBV05] Z. Liu, S.M. Bridges, y R.B. Vaughn. Combining static analysis and dynamic learning to build accurate intrusion detection models. En *Proceedings 3rd IEEE International Workshop on Information Assurance*, páginas 164–177. IEEE, 2005.

[Lew98] D. Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. *Machine Learning: ECML-98*, páginas 4–15, 1998.

[Lin11] P. Linz. *An introduction to formal languages and automata*. Jones & Bartlett Publishers, 2011.

[LM08] M. Latapy y C. Magnien. Complex network measurements: Estimating the relevance of observed properties. En *The 27th IEEE Conference on Computer Communications (INFOCOM).*, páginas 1660–1668. IEEE, 2008.

[LMC07] N. Li, Z. Mao, y H. Chen. Usable mandatory integrity protection for operating systems. En *Security and Privacy, 2007. SP'07. IEEE Symposium on*, páginas 164–178. IEEE, 2007.

[LRW10] A. Lineberry, D. L. Richardson, y T Wyatt. These aren't the permissions you're looking for. *Blackhat*, 2010.

[LSAGB10] C. Laorden, B. Sanz, G. Alvarez, y P. G. Bringas. A threat model approach to threats and vulnerabilities in on-line social networks. En *Advances in Intelligent and Soft Computing*, tomo 85, páginas 135–142. 2010.

[LSJM08] S.M. Lee, S. Suh, B. Jeong, y S. Mo. A multilayer mandatory access control mechanism for mobile devices based on virtualization. En *5th IEEE Consumer Communications and Networking Conference (CCNC).*, páginas 251–256. IEEE, 2008.

[LSS00] W. Lee, S.J. Stolfo, y COLUMBIA UNIV NEW YORK DEPT OF COMPUTER SCIENCE. *Data mining approaches for intrusion detection*. Defense Technical Information Center, 2000.

[LUPS+12] C. Laorden, X. Ugarte-Pedrero, I. Santos, B. Sanz, J. Nieves, y P.G. Bringas. On the study of anomaly-based spam filtering using spam as representation of normality. En *Proceedings of the 3th Consumer Communications and Network Conference (CCNC) Research Student Workshop*. 2012.

[LYZC09] L. Liu, G. Yan, X. Zhang, y S. Chen. Virusmeter: Preventing your cellphone from spies. En *Recent Advances in Intrusion Detection*, páginas 244–264. Springer, 2009.

[McK99] D. McKinney. Impact of commercial off-the-shelf (cots) software on the interface between systems and software engineering. En *Proceedings of the 1999 International Conference on Software Engineering.*, páginas 627–628. IEEE, 1999.

[MHH06] M. Miettinen, P. Halonen, y K. Hatonen. Host-based intrusion detection for advanced mobile devices. En *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, tomo 2, páginas 72–76. Ieee, 2006.

[MKK07] A. Moser, C. Kruegel, y E. Kirda. Limits of static analysis for malware detection. En *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC).*, páginas 421–430. IEEE, 2007.

[ML00] A.C. Myers y B. Liskov. Protecting privacy using the decentralized label model. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 9(4):410–442, 2000.

[MM00] G. McGraw y G. Morrisett. Attacking malicious code: A report to the infosec research council. *Software, IEEE*, 17(5):33–41, 2000.

[Mor07] B. Morris. *The Symbian OS architecture sourcebook: design and evolution of a mobile phone OS*. Wiley, 2007.

[MPV07] D.C. Montgomery, E.A. Peck, y G.G. Vining. *Introduction to linear regression analysis*, tomo 49. John Wiley & Sons, 2007.

[MSR07] P. Mell, K. Scarfone, y S. Romanosky. A complete guide to the common vulnerability scoring system (cvss), version 2.0, forum of incident response and security teams, 2007.

[MVVK06] D. Mutz, F. Valeur, G. Vigna, y C. Kruegel. Anomalous system call detection. *ACM Transactions on In-*

formation and System Security (TISSEC), 9(1):61–93, 2006.

[NKZ10] M. Nauman, S. Khan, y X. Zhang. Apex: Extending android permission model and enforcement with user-defined runtime constraints. En *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, páginas 328–332. ACM, 2010.

[NKZS10] M. Nauman, S. Khan, X. Zhang, y J.P. Seifert. Beyond kernel-level integrity measurement: Enabling remote attestation for the android platform. *Trust and Trustworthy Computing*, páginas 1–15, 2010.

[NMHH05] D.C. Nash, T.L. Martin, D.S. Ha, y M.S. Hsiao. Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices. En *3rd IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom).*, páginas 141–145. Ieee, 2005.

[OMEM09a] M. Ongtang, S. McLaughlin, W. Enck, y P. McDaniel. Semantically rich application-centric security in android. En *2009 Annual Computer Security Applications Conference ACSAC'09*, páginas 340–349. IEEE, 2009.

[OMEM09b] Machigar Ongtang, Stephen McLaughlin, William Enck, y Patrick McDaniel. Semantically Rich Application-Centric Security in Android. *2009 Annual Computer Security Applications Conference ACSAC'09*, páginas 340–349, diciembre 2009. doi: 10.1109/ACSAC.2009.39.

[PBMW99] L. Page, S. Brin, R. Motwani, y T. Winograd. The pagerank citation ranking: Bringing order to the web. 1999.

[Pro07] The Android Open Source Project. Bytecode for the dalvik vm, 2007.

[Pro11] The Android Open Source Project. .dex — dalvik executable format. Informe técnico, Google, 2011.

[PSY10] P. Porras, H. Saïdi, y V. Yegneswaran. An analysis of the ikee. b iphone botnet. *Security and Privacy in*

Mobile Information and Communication Systems, páginas 141–152, 2010.

[Qui86] J.R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[Qui93] J.R. Quinlan. *C4. 5: programs for machine learning*. Morgan kaufmann, 1993.

[RKLC03] J.C. Rabek, R.I. Khazan, S.M. Lewandowski, y R.K. Cunningham. Detection of injected, dynamically generated, and obfuscated malicious code. En *Proceedings of the 2003 ACM workshop on Rapid malcode*, páginas 76–82. ACM, 2003.

[RLPS+00] R. Ramjee, TF La Porta, L. Salgarelli, S. Thuel, K. Varadhan, y L. Li. Ip-based access network infrastructure for next-generation wireless data networks. *Personal Communications, IEEE*, 7(4):34–41, 2000.

[Rut06] J. Rutkowska. Introducing stealth malware taxonomy. *COSEINC Advanced Malware Labs*, 2006.

[Sal74] J.H. Saltzer. Ongoing research and development on information protection. *ACM SIGOPS Operating Systems Review*, 8(3):8–24, 1974.

[SBN+10] I. Santos, F. Brezo, J. Nieves, Y. Penya, B. Sanz, C. Laorden, y P. G. Bringas. Idea: Opcode-sequence-based malware detection. En *Engineering Secure Software and Systems*, tomo 5965 de *LNCS*, páginas 35–43. 2010. 10.1007/978-3-642-11747-3_3.

[SBUP+] Igor Santos, Felix Brezo, Xabier Ugarte-Pedrero, Pablo García Bringas, y J.M. Gómez-Hidalgo. Noa: An information retrieval based malware detection system. *Information Sciences*, ??(??):?? In press.

[SCCA09] A.D. Schmidt, J.H. Clausen, A. Camtepe, y S. Albayrak. Detecting symbian os malware through static function call analysis. En 4^{th} *International Conference on Malicious and Unwanted Software (MALWARE)*, páginas 15–22. IEEE, 2009.

[SCFY96] R.S. Sandhu, E.J. Coyne, H.L. Feinstein, y C.E. Youman. Role-based access control models. *Computer*, 29(2):38–47, 1996.

[SEZS01] M.G. Schultz, E. Eskin, F. Zadok, y S.J. Stolfo. Data mining methods for detection of new malicious executables. En *Proceedings of the 2001 IEEE Symposium on Security and Privacy, 2001. S&P*, páginas 38–49. IEEE, 2001.

[Sha51] CE Shannon. Entropy of printed english bell system tech. *J*, 30(1):50–64, 1951.

[Sha03] A. Shapiro. Monte carlo sampling methods. *Handbooks in Operations Research and Management Science*, 10:353–425, 2003.

[SHG02] M. Schmid, F. Hill, y A.K. Ghosh. Protecting data from malicious software. En *Proceedings of the 18$^{\text{th}}$ Annual Computer Security Applications Conference*, páginas 199–208. IEEE, 2002.

[SJS06] U. Shankar, T. Jaeger, y R. Sailer. Toward automated information-flow integrity verification for security-critical applications. En *Proceedings of the 2006 ISOC Networked and Distributed Systems Security Symposium (NDSS'06), San Diego, CA, USA*. 2006.

[SKE+12] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, y Y. Weiss. "andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, páginas 1–30, 2012.

[SKFT09] W. Shin, S. Kiyomoto, K. Fukushima, y T. Tanaka. Towards formal analysis of the permission-based security model for android. En *Wireless and Mobile Communications, 2009. ICWMC'09. Fifth International Conference on*, páginas 87–92. IEEE, 2009.

[SKFT10] W. Shin, S. Kiyomoto, K. Fukushima, y T. Tanaka. A formal model to analyze the permission authorization and enforcement in the android framework. En *Proceedings of the 2010 IEEE Second International Conference on Social Computing, SOCIALCOM*, tomo 10, páginas 944–951. 2010.

[SKK+10] W. Shin, S. Kwak, S. Kiyomoto, K. Fukushima, y T. Tanaka. A small but non-negligible flaw in the

android permission scheme. En *Policies for Distributed Systems and Networks (POLICY), 2010 IEEE International Symposium on*, páginas 107–110. IEEE, 2010.

[SKM09] Y. Singh, A. Kaur, y R. Malhotra. Comparative analysis of regression and machine learning methods for predicting fault proneness models. *International Journal of Computer Applications in Technology*, 35(2):183–193, 2009.

[SKY+06] M. Stiegler, A.H. Karp, K.P. Yee, T. Close, y M.S. Miller. Polaris: virus-safe computing for windows xp. *Communications of the ACM*, 49(9):83–88, 2006.

[SLAB10] Borja Sanz, Carlos Laorden, Gonzalo Alvarez, y Pablo G. Bringas. A threat model approach to attacks and countermeasures in on-line social networks. En *Proceedings of the 11th Reunion Española de Criptografía y Seguridad de la Información (RECSI), 7-10th September, Tarragona (Spain), in press.*, páginas 343–348. 2010.

[SLTW04] V. Svetnik, A. Liaw, C. Tong, y T. Wang. Application of breiman's random forest to modeling structure-activity relationships of pharmaceutical molecules. *Multiple Classifier Systems*, páginas 334–343, 2004.

[SM97] D. Samfat y R. Molva. Idamn: an intrusion detection architecture for mobile networks. *Selected Areas in Communications, IEEE Journal on*, 15(7):1373–1380, 1997.

[SM09] K. Scarfone y P. Mell. An analysis of cvss version 2 vulnerability scoring. En *Empirical Software Engineering and Measurement, 2009. ESEM 2009. 3rd International Symposium on*, páginas 516–525. IEEE, 2009.

[SPDB09] I. Santos, Y.K. Penya, J. Devesa, y P.G. Bringas. N-Grams-based file signatures for malware detection. En *Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS), Volume AIDSS*, páginas 317–320. 2009.

[SPL+09] A.D. Schmidt, F. Peters, F. Lamour, C. Scheel, S.A. Çamtepe, y Ş. Albayrak. Monitoring smartphones for anomaly detection. *Mobile Networks and Applications*, 14(1):92–106, 2009.

[SSB+09] Aubrey-Derrick Schmidt, Hans-Gunther Schmidt, Leonid Batyuk, Jan Hendrik Clausen, Seyit Ahmet Camtepe, Sahin Albayrak, y Can Yildizli. Smartphone malware evolution revisited: Android next target? *2009 4th International Conference on Malicious and Unwanted Software (MALWARE)*, páginas 1–7, octubre 2009. doi:10.1109/MALWARE.2009.5403026.

[SSL+12] B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, y P.G. Bringas. On the automatic categorisation of android applications. En *Proceedings of the 9th IEEE Consumer Communications and Networking Conference (CCNC2012)*. 2012. In press.

[SSPK08] W. Sun, R. Sekar, G. Poothia, y T. Karandikar. Practical proactive integrity preservation: A basis for malware defense. En *IEEE Symposium on Security and Privacy.*, páginas 248–262. IEEE, 2008.

[Sta03] W. Stallings. *Cryptography and network security*, tomo 2. Prentice hall, 2003.

[SWS+00] A.W.M. Smeulders, M. Worring, S. Santini, A. Gupta, y R. Jain. Content-based image retrieval at the end of the early years. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 22(12):1349–1380, 2000.

[SZZ+11] R. Schlegel, K. Zhang, X. Zhou, M. Intwala, A. Kapadia, y X. Wang. Soundcomber: A stealthy and context-aware sound trojan for smartphones. En *Proceedings of the Network and Distributed System Security Symposium*. 2011.

[Tea11] Android Team. Android developers references. Informe técnico, Google, 2011.

[TGC07] Tcg mobile reference architecture specification version 1.0. Informe técnico, Trusted Computing Group, 2007.

[TM02] K.M.C. Tan y R.A. Maxion. Why 6? defining the operational limits of stide, an anomaly-based intrusion detector. En *Proceedings of the 2002 IEEE Symposium on Security and Privacy.*, páginas 188–201. IEEE, 2002.

[TMP08] P. Traynor, P. McDaniel, y T.L. Porta. Vulnerabilities in cellular data networks. *Security for Telecommunications Networks*, páginas 109–131, 2008.

[UPSB11] X. Ugarte-Pedrero, I. Santos, y P.G. Bringas. Boosting scalability in anomaly-based packed executable filtering. En *In Proceedings of the 7th International Conference on Information Security and Cryptology (INSCRYPT)*. 2011. In press.

[VH08] D. Venugopal y G. Hu. Efficient signature based malware detection on mobile devices. *Mobile Information Systems*, 4(1):33–49, 2008.

[VMw] Inc. VMware. Vmware mobile virutalization platform, virtual appliances for mobile phones. Accesed 25-03-2011.

[WCO+90] D. Wichers, D.M. Cook, R.A. Olsson, J. Crossley, P. Kerchen, K. Levitt, y R. Lo. Pacl's: An access control list approach to anti-viral security. En *Proceedings of the 13th National Computer Security Conference*, páginas 340–349. 1990.

[WD01] D. Wagner y R. Dean. Intrusion detection via static analysis. En *Proceedings of the 2001 IEEE Symposium on Security and Privacy S&P*, páginas 156–168. IEEE, 2001.

[WFP99] C. Warrender, S. Forrest, y B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. En *Proceedings of the 1999 IEEE Symposium on Security and Privacy.*, páginas 133–145. IEEE, 1999.

[Win08] J. Winter. Trusted computing building blocks for embedded linux-based arm trustzone platforms. En *Proceedings of the 3rd ACM workshop on Scalable trusted computing*, páginas 21–30. ACM, 2008.

[WN07] B. Witten y C. Nachenberg. Malware evolution: A snapshot of threats and countermeasures in 2005. *Malware Detection*, páginas 3–15, 2007.

[WS96] Y. Wilks y M. Stevenson. The grammar of sense: Is word-sense tagging much more than part-of-speech tagging? *Arxiv preprint cmp-lg/9607028*, 1996.

[WWH08] T.Y. Wang, C.H. Wu, y C.C. Hsieh. A virus prevention model based on static analysis and data mining methods. En *IEEE 8th International Conference on Computer and Information Technology Workshops. CIT Workshops*, páginas 288–293. IEEE, 2008.

[XBL+11] C. Xiang, F. Binxing, Y. Lihua, L. Xiaoyi, y Z. Tianning. Andbot: towards advanced mobile botnets. En *Proceedings of the 4th USENIX conference on Large-scale exploits and emergent threats*, páginas 11–11. USENIX Association, 2011.

[Xen06] C. Xenakis. Malicious actions against the gprs technology. *Journal in Computer Virology*, 2(2):121–133, 2006.

[XMTZ06] W. Xu, K. Ma, W. Trappe, y Y. Zhang. Jamming sensor networks: attack and defense strategies. *Network, IEEE*, 20(3):41–47, 2006.

[YE05] T. Yap y H. Ewe. A mobile phone malicious software detection model with behavior checker. *Web and Communication Technologies and Internet-Related Social Issues-HSI 2005*, páginas 920–921, 2005.

[ZAS09] X. Zhang, O. Acıiçmez, y J.P. Seifert. Building efficient integrity measurement and attestation for mobile phone platforms. *Security and Privacy in Mobile Information and Communication Systems*, páginas 71–82, 2009.

[ZBWKM06] N. Zeldovich, S. Boyd-Wickizer, E. Kohler, y D. Mazières. Making information flow explicit in histar. En *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation*, páginas 19–19. 2006.

[ZJ12a] Y. Zhou y X. Jiang. Dissecting android malware: Characterization and evolution. 2012.

[ZJ12b] Y. Zhou y X. Jiang. Dissecting android malware: Characterization and evolution. En *Proceedings of the 33rd IEEE Symposium on Security and Privacy (S&P 2012)*. 2012.

ÍNDICE ALFABÉTICO

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z