



UNIVERSIDAD DE DEUSTO

UN NUEVO ENFOQUE PARA LA DETECCIÓN DE MALWARE EN ANDROID A TRAVÉS DEL ANÁLISIS DEL COMPORTAMIENTO

Tesis doctoral presentada por José Gaviria de la Puerta
dentro del Programa de Doctorado en Informática y Telecomunicaciones

Dirigida por Dr. Pablo García Bringas
y Dr. Borja Sanz Urquijo



UNIVERSIDAD DE DEUSTO

UN NUEVO ENFOQUE PARA LA DETECCIÓN DE MALWARE EN ANDROID A TRAVÉS DEL ANÁLISIS DEL COMPORTAMIENTO

Tesis doctoral presentada por José Gaviria de la Puerta
dentro del Programa de Doctorado en Informática y Telecomunicaciones

Dirigida por Dr. Pablo García Bringas
y Dr. Borja Sanz Urquijo

El doctorando

El director

El director

Bilbao, julio 2015

*A mis padres, a mis hermanas, a mi cuñado y a mis sobrinas.
Sois la motivación para seguir.*

*A Izaskun.
Un paso más en nuestro camino.*

«Tu enfoque determina tu realidad.»

Qui-Gon Jinn, creado por George Lucas

Abstract

Mobile devices or *smartphones* have occupied an important position in the field of communication. The widespread use of which has led to complex operating systems that manage and store private information of users. It is for this reason that the security of these systems has become a priority for companies and users and the scientific community.

One of the most important areas within the security of these systems is the detection of malicious software (*malware*), which consists to detect malicious applications within various platforms.

In this PHD thesis, we focus on the detection of *malware* in *Android* operating system. This system is today the dominant operating system on mobile devices; so that, *malware* developers have focused their efforts on him.

Specifically, in this PHD we propose two *malware* detection techniques based on approaches that have proved their worth in the desktop environment. In order to demonstrate the validity of these techniques, we develop two approaches using the static and the dynamic approach for *malware* detection.

In the static approach, we conducted a comprehensive study of operational codes which are extracted from applications. Through these characteristics and using machine learning algorithms, we seek to predict whether an application is malicious or benign.

On the other hand, we analyse the dynamic detection techniques, proposing a system of automation of virtual machines for *Android* system. With this system, we obtain the network traffic generated by the execution of applications.

Subsequently, using a methodology previously validated in desktop environments, combined with supervised machine learning techniques, we classify applications and compare the results with those obtained in the desktop approximation.

Resumen

Los terminales móviles inteligentes o *smartphones* han ocupado un puesto importante en el ámbito de la comunicación. El uso extendido de los mismos ha dado lugar a sistemas operativos complejos, que gestionan y almacenan información de carácter privado y sensible de los usuarios. Es por ello, que la seguridad en estos sistemas se ha convertido en una prioridad tanto para empresas y usuarios como para la comunidad científica.

Uno de los ámbitos más importantes dentro de la seguridad de estos sistemas es la detección de *software* malicioso (*malware*), que consiste en detectar aplicaciones de este tipo dentro de las diversas plataformas.

En esta tesis doctoral, nos centramos en la detección de *malware* en el sistema operativo *Android*. Este sistema operativo, es a día de hoy el predominante en los dispositivos móviles; lo que ha provocado que los desarrolladores de *malware* hayan centrado sus esfuerzos en él.

En concreto, proponemos dos técnicas de detección de *malware*, basados en los enfoques que han demostrado su validez en entorno de escritorio. Con el fin de demostrar la validez de estas técnicas, desarrollamos dos metodologías consistentes en un enfoque de detección estática y en otro de detección dinámica de *malware*.

En la aproximación de detección estática, realizamos un estudio exhaustivo de los códigos operacionales extraídos de las aplicaciones. Mediante estas características y el uso de algoritmos de aprendizaje automático, buscamos predecir si una aplicación es maliciosa o benigna.

Por otro lado, analizamos las técnicas de detección dinámica, proponiendo un sistema de automatización de máquinas virtuales para el sistema *Android*. Con esta plataforma, obtenemos el tráfico de red generado por la ejecución de las aplicaciones.

Posteriormente, empleando una metodología validada en computadores de sobremesa basada en técnicas de aprendizaje automático supervisado, clasificamos las aplicaciones y comparando los resultados obtenidos en ambos entornos.

Laburpena

Terminal mugikor adimenduek edo *smartphone*-ek komunikazio eremuan garrantzi handia hartu dute. Hauen erabileraren hazkundeak, sistema eragile konplexuen agerpena ekarri du. Sistema eragile hauek, erabiltzaileen informazio pribatu eta sentikorra kudeatzen eta gordezen dute. Hori-dela-eta, enpresa eta zientzia elkarteentzako sistema hauen segurtasuna lehentasuna bat bilakatu da.

Sistema hauen segurtasunaren eremu garrantzitsuenetako bat *software* maltzurren (malware) hautematea da, hau da, plataforma desberdinen artean aplikazio maltzurak hautematean oinarritzen da. Tesilan honetan, *Android* sistema eragilean *malware*-a hautematean zentratuko gara. Gaur egun sistema hau, gailu mugikorren artean sistema eragile nagusia da, ondorioz, *malware* garatzaileak sistema honetan murgildu dira.

Bereziki, tesi-lan honetan *malware* detekziorako bi teknika proposatzen ditugu, idatz-mahaiko sistemetan eraginkorrak direla. Eraginkortasuna frogatzeko *malware* detektatzeko ikuspuntu estatiko eta dinamikoetan oinarritutako bi metodologia garatu ditugu. Detekzio estatikoko hurbilketa, aplikazioetatik ateratako kode-operazionalen azterketa sakona egingo dugu. Ezaugarri hauen eta ikasketa automatikoetan oinarritutako algoritmoen bitartez, aplikazio bat ona ala maltzurra den jakitea bilatzen dugu.

Aldiz, detekzio dinamikoko hurbilketa, *Android* sistemarako makina birtualen automatizazioan oinarritutako sistema bat proposatzen dugu. Sistema honekin, aplikazioen exekuzioaren bidez sortutako sare-trafficoa lortzen dugu.

Bukatzeko, idatz-mahaiko ordenagailu sistemetan jadanik balioztatutako metodologia bat erabiliz, eta gainbegiratutako ikasketa automatikoan oinarritutako sistema batekin konbinatuta, aplikazioak sailkatuko ditugu. eta azkenik lortutako emaitzak idatz-mahaiko hurbilketarako emaitzekin alderatuko ditugu.

Agradecimientos

En este pequeño apartado quiero dejar constancia de todas las personas que me han ayudado con su trabajo y apoyo, pasando a formar parte de esta tesis doctoral. Quiero aclarar que el orden que he seguido no implica mayor o menor importancia.

Esta tesis doctoral no podía haber sido de otra forma si no es gracias a mis dos directores. Primero *Pablo García Bringas*, el cual me brindó la posibilidad de entrar en el laboratorio *S3Lab*, después llamado *DeustoTech-Computing*. Me acuerdo de ese día entrando por su despacho y diciéndole que quería ser parte de la historia del laboratorio. Segundo a *Borja Sanz Urquijo*, director, jefe y ante todo, amigo. Desde que te conocí allá por el 2008 supe que siempre estarías ahí y así ha sido.

No me puedo olvidar del resto de gente del laboratorio que me ha estado apoyando durante tiempo inmemorable. *Asier, Sendoa, Sergio, Xabi, Igor, Javi, Iskander, Carlos, Irene, Patxi, Iván, Iñigo, Nora, Mikel*, esta tesis ha sido con la colaboración de cada uno de vosotros. Dentro de este grupo, especial agradecimiento a *Jorge de la Peña*, gracias por los consejos sobre Bérnago, las horas pasadas tomando cañas y ser buen amigo.

A *Félix Brezo*, esto empezó con *BRIANA* y terminó en dos tesis doctorales. Gracias por compartir el camino aunque haya terminado siendo en la distancia.

A *Iker Pastor*, gran amigo, mejor persona, gracias por tirar de mí cuando no sabía dónde estaba. Bien sabes que esta tesis doctoral ha llegado a buen puerto por tu perseverancia en levantarme cada vez que caía. Muchas gracias.

También me acuerdo de esos primeros días hablando con Pablo y un chico alegre me dijo que a ver que día me iba a incorporar, que tenía ganas de conocerme, y no sólo me ha conocido, se ha convertido en un

mentor, gracias *Agustín Zubillaga*, todavía me falta mucho de aprender de ti.

A *Giuseppe Psaila*, gran anfitrión en mi estancia en la Universidad de *Bérgamo*. Gracias por darme la oportunidad de poder conocer el trabajo vuestro en los ámbitos de la seguridad y por hacerme disfrutar de la estancia allí. *Grazie mille per tutto, amico. Siete sempre i benvenuti a Bilbao.*

A mis *Kamaradas*, los cuales sois una parte fundamental de mí. Habéis aguantado varios años duros para mí, pero ahí me habéis apoyado. Muchas gracias por no darme por perdido.

A *Aitor Esquibel* por demostrar lo que es un amigo. Ha habido momentos malos y ahí has sabido estar, sabido apoyar y sacar lo mejor de cada momento. Muchas gracias por tu apoyo incondicional y ser buen amigo.

Otra de las personas a las que tengo que agradecer todo este tiempo, la tesis, y ser la conversación más larga que haya tenido, es a *Alba March*. Gracias por ayudarme todos los días un poco más, escucharme y leerme cuando me quejo. No hace falta que te diga que has sido un apoyo en Bilbao y en *Bérgamo*. Esta tesis también tiene parte de ti. Siempre quedará Copenhague. Nunca dejes de soñar.

También quiero agradecer esta tesis a *Leire Gonzalez Peña*, amiga incansable que ha estado ahí peleando para que sacara una sonrisa en los peores momentos. Muchas gracias. A *Isabel Monreal*, tu apoyo desde el primer momento ha sido uno de los pilares para que esta tesis salga adelante.

A *Aritz*, *Iratxe* y *Calvin*, grandes amigos. Os he dejado abandonados mucho tiempo pero aquí estoy de nuevo. A *Gaizka Errasti*, gracias por obligarme a hacer descansos para que me aireara un rato.

A otra de mis cuadrillas, *Natxo*, *Aiala*, *Aitziber* y *Borja*, gracias por esas maravillosas excursiones y tardes de juegos que hacían despejar mi cabeza de esta tesis, ¿Para cuándo otro *Cachopo*?.

Por supuesto, no me puedo olvidar de mi nueva cuadrilla que ha unido la distancia, gracias *Bergaños*, os llevo siempre en el corazón.

A mi familia política, por aceptarme desde el primer momento como uno más de la familia. Es un placer poder agradeceros todo lo que habéis hecho por mí. Muchas gracias.

A mi familia, han sido años duros que gracias a vuestro apoyo incondicional, a saber llevarme, a levantarme cuando me caigo, esta tesis está escrita. Muchísimas gracias *Ama, Aita, Marta, Sara, Diego* y mis dos princesitas, *Daniela y Martina*. A todo el resto de mi familia, a todos y cada uno de los miembros de ella, pero en concreto a mi abuelo, imagen a seguir, aguantando hasta que la terminé para marcharse, muchas gracias abuelo, te echaré de menos.

También, agradecer a *Izaskun Aiarzaguena Martínez* por todo. Gracias por aguantar, por recibir malas contestaciones, por estar ahí en lo bueno y en lo malo. Esta tesis no es mía, es tuya, sin tu apoyo no podría estar escrita.

Por último agradecer a toda la gente que ha estado conmigo en este camino. Gente que aunque no mencione no significa que no sea importante para mí. Gracias a todos y lo siento si no os he mencionado.

Eskerrik asko,

Sete

Julio del 2015

Índice general

Índice de figuras	xv
Índice de tablas	xvii
1 Introducción	1
1.1 Estado actual de los <i>smartphones</i>	1
1.1.1 Definición de <i>smartphone</i>	2
1.1.2 Plataformas de <i>smartphones</i>	2
1.1.2.1 <i>Android</i>	2
1.1.2.2 <i>iOS</i>	3
1.1.2.3 <i>Windows Phone</i>	4
1.1.2.4 <i>BlackBerry 10</i>	4
1.1.2.5 Otras plataformas de <i>smartphone</i>	4
1.2 Estado actual del cibercrimen	5
1.3 Estado actual del cibercrimen en los <i>smartphones</i>	6
1.4 Hipótesis y objetivos	7
1.5 Estructura de la Tesis	9
2 Historia de las amenazas	11
2.1 Evolución del <i>malware</i>	11
2.1.1 <i>Virus</i>	12
2.1.2 <i>Gusanos</i>	12
2.1.3 <i>Troyanos</i>	12
2.1.4 <i>Spyware</i>	13

ÍNDICE GENERAL

2.1.5	<i>Puerta trasera</i>	13
2.1.6	<i>Rootkit</i>	13
2.1.7	<i>Bots</i>	13
2.2	La tecnología cambia	16
2.2.1	El <i>malware</i> en las nuevas tecnologías	19
2.2.2	Vectores de amenaza	23
2.2.2.1	Basados en el <i>hardware</i>	23
2.2.2.2	Independientes del dispositivo	23
2.2.2.3	Basadas en el software	24
2.2.2.4	Basada en el usuario	24
2.3	Sumario	24
3	Estado de la técnica en la plataforma <i>Android</i>	27
3.1	Evolución del <i>malware</i> en <i>smartphones</i>	27
3.2	La arquitectura de <i>Android</i>	30
3.2.1	La seguridad en <i>Android</i>	33
3.2.2	Las características de la seguridad del usuario	37
3.2.3	La seguridad <i>Android</i> a nivel de aplicación	37
3.2.3.1	El modelo de permisos de <i>Android</i>	38
3.2.3.2	La información personal	41
3.2.3.3	Los metadatos <i>Android</i>	41
3.3	Vectores de infección	41
3.3.1	Ataques centrado en el <i>hardware</i>	42
3.3.2	Ataques independientes al dispositivo	43
3.3.3	Ataques centrados en el <i>software</i>	44
3.3.4	Ataques centrados en el usuario	46
3.4	Técnicas de detección de <i>malware</i>	47
3.4.1	Análisis estático	47
3.4.1.1	Análisis de los permisos de aplicación	51
3.4.2	Análisis dinámico	52
3.4.2.1	Detección de anomalías	53
3.4.2.2	Detección de comportamientos maliciosos	54
3.4.2.3	Detección de consumos energéticos	57

3.5	Sumario	58
4	Utilización de opcodes como predictor de la detección	61
4.1	Trabajo relacionado	62
4.2	Alcance de la experimentación	64
4.2.1	Recolección de las muestras	65
4.2.2	Obtención de la información	67
4.3	Aprendizaje supervisado	67
4.3.1	Máquinas de soporte vectorial	67
4.3.1.1	Funciones de kernel	68
4.3.2	Árboles de decisión	69
4.3.3	Redes bayesianas	72
4.3.4	K vecinos más cercanos	74
4.3.5	Simple Logistic	76
4.3.6	Validación cruzada	76
4.4	Validación experimental	77
4.5	Sumario	80
5	Análisis del tráfico de red en la detección de <i>malware</i>	83
5.1	Protocolos de comunicación	84
5.1.0.1	<i>User Datagram Protocol</i>	86
5.1.0.2	<i>Transmission Control Protocol</i>	87
5.2	Estado del arte	87
5.3	Metodología creada	93
5.3.1	Preparación del entorno	93
5.3.2	Proceso de automatización	94
5.4	Aprendizaje supervisado	95
5.4.1	Máquinas de soporte vectorial	96
5.4.2	Método <i>Bagging</i>	97
5.4.3	Redes bayesianas	98
5.4.4	K vecinos más cercanos	99
5.5	Evaluación general	99
5.6	Validación empírica	101
5.6.1	Capacidad del clasificador para la clasificación de tráfico	102

ÍNDICE GENERAL

5.7	Sumario	105
6	Conclusiones	109
6.1	Contribuciones principales	109
6.2	Trabajo futuro	113
6.2.1	Mejora del método de detección dinámica	113
6.2.2	Combinación de características en la detección estática . .	114
6.2.3	Creación de un sistema híbrido para la detección de <i>malware</i>	114
6.3	Consideraciones finales	115
7	Conclusions	117
7.1	Summary of results	117
7.2	Future Work	121
7.2.1	Improvement of the dynamic detection method	121
7.2.2	Combination of features in static detection	121
7.2.3	Creating a hybrid system for the detection of <i>malware</i> . .	121
7.3	Final considerations	122
	Bibliografía	127
	Índice alfabético	151

Índice de figuras

1.1	Número de aplicaciones maliciosas aparecidas por mes para la plataforma <i>Android</i> y número de aplicaciones maliciosas totales. . . .	7
2.1	Evolución de las ventas de dispositivos móviles a lo largo de los años.	18
2.2	Evolución de S.O. móvil	19
2.3	Características específicas de la seguridad móvil.	20
3.1	Cuota de mercado por dispositivo móvil.	30
3.2	Incremento de muestras en <i>malware</i> bancario para dispositivos móviles en el año 2013.	31
3.3	Arquitectura del sistema operativo <i>Android</i>	32
3.4	Permisos de aplicación pedidos por una aplicación en tiempo de instalación.	40
4.1	Ejemplo de máquina de soporte vectorial.	68
4.2	Ejemplo de árbol de decisión.	70
4.3	Ejemplo de Red Bayesiana.	72
4.4	Ejemplo del funcionamiento de <i>KNN</i> en un espacio bidimensional para $k = 4$	75
4.5	Ejemplo de los conjuntos de entrenamiento en la validación cruzada para una división de $K = 10$ conjuntos	76
4.6	Example of operational codes with variables used in an <i>Android</i> application.	78
5.1	Diferencia entre la pila de protocolos OSI y pila TCP/IP	85

ÍNDICE DE FIGURAS

5.2	Proceso de automatización de las máquinas virtuales	95
-----	---	----

Índice de tablas

2.1	Tipos de Malware y sus características.	15
4.1	Cantidad de aplicaciones por categoría descargadas de la <i>Play Store</i> de <i>Google</i>	66
4.2	Capacidad de los clasificadores usando los códigos operacionales como característica de la clasificación.	78
4.3	Capacidad de los clasificadores usando los permisos como característica de la clasificación.	80
5.1	Tabla comparativa de ventajas y desventajas entre los protocolos de comunicación <i>UDP</i> y <i>TCP(1/2)</i>	88
5.2	Tabla comparativa de ventajas y desventajas entre los protocolos de comunicación <i>UDP</i> y <i>TCP(2/2)</i>	89
5.3	Resultados obtenidos por los clasificadores para el análisis de tráfico de red en aplicaciones de <i>Android</i>	103
5.4	Tabla comparativa del mejor y peor clasificador en término de <i>Accuracy</i> de los resultados obtenidos por Brezo y nuestro enfoque.	104
5.5	Tabla comparativa del mejor y peor clasificador en término de <i>TPR</i> y <i>Precision</i> de los resultados obtenidos por Brezo y nuestro enfoque.	105
5.6	Tabla comparativa del mejor y peor clasificador en término de área bajo la curva <i>ROC</i> de los resultados obtenidos por Brezo y nuestro enfoque.	106

«Busco hombres que crean que no
hay cosas imposibles»

Henry Ford (1863 – 1947)

CAPÍTULO

1

Introducción

A lo largo de los últimos años, el crecimiento experimentado por los llamados teléfonos inteligentes o *smartphones* está siendo extraordinario. Sin embargo desde el punto de vista de la seguridad, este tipo de dispositivos ha traído consigo una serie de problemáticas, algunas existentes en entornos de escritorio, así como otras nuevas.

Este capítulo continua de la siguiente forma: en la sección 1.1 veremos el estado actual de los *smartphones*, basándonos en la definición formal de los mismos. Más tarde en la sección 1.2 haremos un resumen del cibercrimen a lo largo de los años y su implicación de cara al usuario. En la sección 1.3 expondremos la unión de la problemática del cibercrimen en los *smartphones*. Para continuar, en la sección 1.4 estableceremos la hipótesis fundamental de esta tesis doctoral, desarrollando a su vez los objetivos específicos y operacionales. Por último, en la sección 1.5 explicamos como se estructura el resto del documento que conforma la tesis doctoral.

1.1 Estado actual de los *smartphones*

En esta sección plantearemos una definición de dispositivos móviles o *smartphones* para después ver las diferentes plataformas dentro de estos dispositivos. Además podremos ver el alcance y penetración de los mismos en el mercado.

1. INTRODUCCIÓN

1.1.1 Definición de *smartphone*

La definición tradicional de *smartphone* nos habla del terminal de telefonía móvil que ejecuta un sistema operativo con determinadas capacidades avanzadas [Smi13], tales como: (i) instalación de *software* de terceros, (ii) disponibilidad de una conectividad a Internet mediante un navegador y, por último, (iii) correo electrónico.

Dichas capacidades impulsaron la adopción de una serie de características *hardware* como métodos de entrada mejorados (pantallas táctiles, teclados *QUERTY*), módulos de conectividad inalámbrica, y sensores de localización.

Estas características diferenciaban claramente a los *smartphones* de los *feature phones*¹ que ejecutaban un sistema operativo cerrado. Sin embargo, la barrera entre estas dos categorías se ha vuelto difusa, puesto que los *feature phones* de la actualidad poseen características *hardware* y *software* propias de los *smartphones*.

1.1.2 Plataformas de *smartphones*

Aunque *Android* e *iOS* acaparan entre ambas más del 90 % de la cuota de mercado, *Windows Phone* y *Blackberry 10* se califican como plataformas de *smartphones* principales debido al fuerte soporte de empresas sólidas en el mundo IT y la disponibilidad de la gran mayoría de aplicaciones de mayor difusión y relevancia en sus catálogos de *software* de terceros.

A continuación vamos a ver más de cerca cada uno de los sistemas operativos más representativos en cuanto a *smartphones* se refiere: (i) *Android*, (ii) *iOS*, (iii) *Windows Phone*, (iv) *BlackBerry 10* y, por último, (v) otros sistemas.

1.1.2.1 *Android*

El sistema operativo *Android* fue presentado en noviembre del 2007 junto a la *Open Handset Alliance*, o *OHA*, tras la adquisición por *Google* de su desarrollador original *Android Inc.* en el 2005 [All07]. La *OHA* es un consorcio que reúne a los principales fabricantes y desarrolladores con el fin de impulsar los estándares abiertos en la telefonía móvil [All11].

¹Es un término aplicado a ciertos teléfonos móviles de baja gama o de características limitadas

Android es un sistema operativo con *kernel* monolítico¹ *Linux*², publicado bajo una licencia *open source* y mantenido por la AOSP (*Android Open Source Project*). Sin embargo contiene *middleware* y *software* propietarios, mayormente desarrollados y licenciados por *Google*. También es posible encontrar *software* de terceros (principalmente incluido por el fabricante del terminal) y algunos controladores de código privativo [GS11].

En este sentido, *Google* ofrece acceso a todo su ecosistema de servicios, destacando los de buscador *Google*, correo *Gmail*, cartografía *Google Maps*, así como a su tienda de aplicaciones *Google Play*. Sin embargo ofrece la libertad de instalar *software* desde *markets* de terceros o manualmente con su gestor de paquetes *APK*.

Es destacable, que existen numerosos proyectos paralelos de *Android*, derivados del código fuente del mismo, como por ejemplo en el mercado chino donde los servicios de *Google* están bloqueados por la legislación del país, algunos con el soporte de empresas como *Amazon*.

1.1.2.2 *iOS*

El sistema operativo *iOS* está desarrollado y mantenido por *Apple* desde su aparición en julio de 2007. Implementa una versión restringida del *kernel* monolítico *Match (Darwin)* presente en los sistemas operativos de escritorio de *Apple*, eliminando el acceso *shell* al usuario y limitando el de aplicaciones [Dev07].

iOS se completa con el acceso a varios servicios proporcionados por su fabricante, como la tienda de aplicaciones *AppStore*, la de música *iTunes*, ó la sincronización en la nube *iCloud*. Como punto centralizado de distribución de *software*, la *AppStore* ofrece exclusivamente aplicaciones filtradas y validadas por *Apple*, incrementando la sensación de seguridad (aunque con algunas apariciones de *malware*, que son anecdóticas) .

Los terminales de referencia son los *iPhone*, desarrollados por *Apple*. Estos, están diseñados teniendo en cuenta las características del sistema operativo y viceversa, consiguiendo una buena integración entre *software* y *hardware*.

¹EL *kernel* monolítico es una arquitectura de sistema operativo en la que el propio sistema operativo tiene control total sobre su núcleo.

²Linux es un sistema operativo libre.

1. INTRODUCCIÓN

1.1.2.3 Windows Phone

Windows Phone es el sistema operativo basado en el *kernel* monolítico de *Windows NT* en su versión actual, y está previsto el cambio a *OneCore* en su inminente iteración a *Windows 10* en el segundo semestre de 2015 [GG14].

Microsoft decidió iniciar un camino hacia la convergencia de sus plataformas, tomando como primer paso la unificación de interfaces de usuario con *ModernUI*, basada en baldosas interactivas de diseño geométrico simple sobre colores planos. En una segunda etapa, *Microsoft* presenta un núcleo común con la premisa de facilitar la creación de aplicaciones universales. Desde un mismo código fuente el compilador creará aplicaciones para cada plataforma de la familia *Windows*, siendo sólo necesario programar y diseñar la interfaz para cada plataforma (PC, *smartphone*, Xbox) [GG14].

Windows integra la misma plataforma de servicios en el ecosistema completo, con el buscador y cartografía *Bing*, suscripciones a la suite ofimática *Office 365* y *Onedrive*.

1.1.2.4 BlackBerry 10

Blackberry 10 es un sistema operativo presentado en enero del 2013 y basado en el *microkernel QNX*, un robusto sistema presente en maquinaria industrial y ECUs de automóvil que fue adquirido por *Blackberry* en el año 2010. Al contrario que un *kernel* monolítico, este último se ejecuta como una serie de servidores independientes que permiten ajustar la funcionalidad del sistema sin hacer cambios en el propio *kernel* [Aus12].

Blackberry es la empresa que mejor ha capitalizado las necesidades de seguridad de las empresas e instituciones en el pasado, haciendo que sus terminales incluyan características de seguridad únicas [San08].

Los *smartphones* de referencia de esta plataforma están abandonando su característico teclado físico para ofrecer un diseño similar a otras plataformas, basadas completamente en interfaces táctiles, así como en los movimientos predefinidos obtenidos por los diferentes sensores del dispositivo, tales como el giroscopio.

1.1.2.5 Otras plataformas de *smartphone*

Existen otras plataformas de *smartphone* reconocidas pero que no disfrutaban de una cuota de mercado significativa en la actualidad, sin embargo están orientadas a proporcionar *smartphones* en los mercados emergentes.

- **FirefoxOS**, sistema de *kernel* monolítico *Linux* completamente abierto impulsado por *Mozilla Foundation* que apuesta por las aplicaciones desarrolladas con estándares web como *HTML5*, *CSS3* y *JavaScript* [NC14].
- **Tizen**, sistema de *kernel* monolítico *Linux* que recoge el testigo de la plataforma *LiMo* desde el año 2012 [GMZ14].

1.2 Estado actual del cibercrimen

Nuestra sociedad ha evolucionado hasta el punto de que la mayoría de las personas pasamos una parte importante de nuestras vidas conectados a *Internet*. En muchos sentidos, este mundo virtual refleja nuestro mundo real; es por ello que los delincuentes, que son una parte de nuestra sociedad, también han aparecido en el mundo virtual. La presencia de estos delincuentes cibernéticos se ha desarrollado de forma paralela al creciente intercambio de dinero y datos que circulan a través de la red, convirtiendo a estos últimos en un claro objetivo cada vez más tentador.

Dado este hecho, el ecosistema de la ciberdelincuencia está llegando a la madurez, definiendo claramente las relaciones y modelos de negocio que van a seguir [Ksh10]. Dentro de estos últimos, nos podemos encontrar con dos tipos de cibercriminales: i) pequeños estafadores que roban pequeñas sumas de dinero pero a una cantidad ingente de personas, o ii) estafadores que intentan robar grandes cantidades de dinero de una única vez.

Aún así, cuando hablamos sobre una actividad delictiva en *Internet*, no nos referimos a mafias geolocalizadas, sino más bien a grupos con actividades complementarias [BI09]. Por ejemplo, el individuo, o grupo, dueño de una *botnet*¹ capaz de lanzar ataques de denegación de servicio (*DDoS*) o distribuir correo no deseado, necesita direcciones de correo electrónico. Por otro lado, otra persona, a quien el

¹Un conjunto o red de robots informáticos o bots, que se ejecutan de manera autónoma y automática

1. INTRODUCCIÓN

propietario de la *botnet* no tiene por qué conocer, cubre esta necesidad mediante el robo y venta de las direcciones necesarias. Como podemos ver, los patrones de negocio de la vida real, se extienden a este mundo virtual.

Por ello, el cibercrimen es como cualquier otro negocio. Se comporta de acuerdo a los principios de negocio tradicionales, tales como la rentabilidad, facilidad de uso, la gestión del riesgo, y los mercados emergentes.

La avalancha de nuevos servicios que están disponibles a través de Internet, además de una población mundial que ahora adopta más fácilmente estos nuevos servicios, también contribuye al éxito de la ciberdelincuencia. Las áreas que son especialmente vulnerables a los ataques incluyen [McQ08]: i) la banca en línea, obligando a los distintos agentes económicos a securizar sus servicios a través de Internet; ii) las instalaciones de almacenamiento de datos y aplicaciones en la nube, donde las empresas externalizan parte de su infraestructura tecnológica; iii) los juegos en línea, que generan un gran volumen económico; iv) las agencias bursátiles en línea, permitiendo transacciones monetarias desde cualquier localización; y v) la *Web 2.0* con una generación ingente de datos diaria.

Los cibercriminales actuales hacen uso de herramientas para llevar a cabo sus acciones. En los últimos tiempos se han venido utilizando los llamados troyanos como armas preferidas por los atacantes. Mediante estas herramientas son capaces de construir *botnets*, robar contraseñas, datos confidenciales, realizar ataques de denegación de servicio o los más actuales de cifrar el contenido del disco para pedir una recompensa a cambio [ATS14]. Una característica de los ataques actuales es que buscan mantener una presencia en la máquina infectada durante el mayor tiempo posible.

1.3 Estado actual del cibercrimen en los *smartphones*

Desde hace décadas, los ordenadores han sido los únicos dispositivos que se usaban dentro de la informática. A día de hoy, la tecnología móvil ha servido de apoyo para facilitar las tareas que antes únicamente se desarrollaban en los ordenadores de sobremesa.

Con el aumento de las capacidades de este tipo de teléfonos, el *malware*, una de las amenazas más extendidas en los sistemas tradicionales, también se ha trasladado

1.3 Estado actual del cibercrimen en los *smartphones*

a estos nuevos sistemas. Según *Kaspersky Labs*, el número de muestras de *malware* nuevas encontradas en el año 2014 era de 4.643.582 [Emm14]. En la Figura 1.1 se puede ver el incremento de las muestras totales de *malware* para *Android* a lo largo de los últimos años.

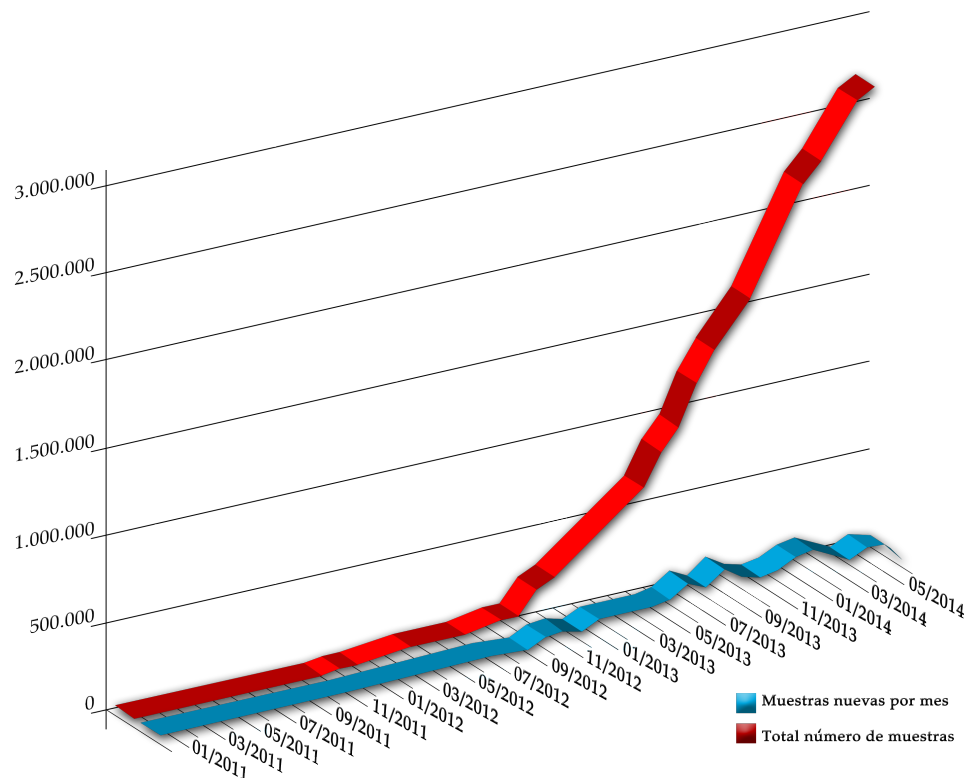


Figura 1.1: Número de aplicaciones maliciosas aparecidas por mes para la plataforma *Android* y número de aplicaciones maliciosas totales.

El *malware* en sistemas móviles no es inocuo y su capacidad para hacer acciones maliciosas incluye desde el robo de datos confidenciales en el dispositivo, las escuchas de las conversaciones en curso por parte del atacante, o incurrir en gastos adicionales para el usuario mediante la suscripción en sistemas de tarificación adicional.

1. INTRODUCCIÓN

1.4 Hipótesis y objetivos

Frente a estos hechos podemos formular la siguiente hipótesis fundamental:

«La representación de las aplicaciones móviles obtenida a través de la caracterización tanto de su naturaleza como de su comportamiento permite una detección efectiva de aplicaciones maliciosas.»

La intención que subyace en dicha hipótesis es la de, mediante el uso de técnicas de desensamblado y de técnicas dinámicas de análisis, unidas al uso de aprendizaje automático supervisado, crear un método de detección temprana de aplicaciones maliciosas en estos dispositivos móviles.

A partir de esta hipótesis podemos ver los siguientes objetivos específicos:

- Analizar aquellos aspectos susceptibles de ser representativos de la intencionalidad de una aplicación móvil.
- Extraer una representación de los códigos operacionales más usados para el comportamiento malicioso en dispositivos móviles para la detección del *malware*.
- Extraer una representación del comportamiento del *malware* en la red, para así poder detectarlo no sólo de forma estática, sino también de forma dinámica.
- Sistematizar la extracción y evaluación de las representaciones a fin de determinar su viabilidad en un entorno fuera del laboratorio de experimentación.

Los siguientes objetivos operacionales servirán para dirigir la tesis hacia los objetivos descritos:

- **Estudio de las diferentes técnicas usadas para la detección de *malware* en dispositivos móviles.** Realizaremos un estudio de las técnicas usadas para la detección de *malware* en los dispositivos móviles, así como de la evolución que han sufrido estas técnicas hasta nuestros días.

- **Caracterizar la naturaleza de las aplicaciones en base a atributos estáticos.** Analizaremos nuevas características que faciliten la caracterización de las aplicaciones en entornos móviles.
- **Análisis de las comunicaciones en aplicaciones maliciosas.** Estudiaremos el tráfico de red generado por distintas muestras de *malware*, a fin de detectar patrones o comportamientos comunes que faciliten su caracterización.
- **Desarrollar nuevas técnicas de representación.** Desarrollaremos nuevas técnicas de representación de *malware* teniendo usando las características tanto estáticas como dinámicas anteriormente obtenidas.
- **Desarrollar nuevas técnicas de detección.** Estudiaremos nuevas técnicas de detección de *malware* a partir esta representación.

1.5 Estructura de la Tesis

Esta tesis doctoral se organiza de la siguiente manera.

En el *capítulo 1* hemos 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.

En el *capítulo 2* realizaremos un resumen de la historia del *malware* en terminos generales. En las primeras secciones del documento explicaremos todos los tipos de *malware* conocido. A su vez haremos un breve estudio de la evolución de la tecnología y como esta ha ido direccionándose a los dispositivos móviles.

En el *capítulo 3* veremos una introducción al sistema *Android* y a las técnicas de detección llevadas para el sistema. En una primera parte, explicaremos en profundidad toda la arquitectura del sistema. La segunda parte, recopilaremos las investigaciones más relevantes sobre la detección de *malware* en la plataforma *Android* a lo largo de los años. Dentro de esta parte también veremos una referencia a las diferentes técnicas de detección que se han llevado a cabo por parte de la comunidad científica.

1. INTRODUCCIÓN

El *capítulo 4* es la experimentación mediante el uso de los códigos operacionales de los dataset obtenidos a través de la comunidad científica. En dicho experimento presentaremos nuestro enfoque, mediante el cual haciendo uso de los *opcodes* y técnicas de *machine learning* o aprendizaje supervisado, somos capaces de clasificar una aplicación como maliciosa o benigna.

En el *capítulo 5* mostraremos como haciendo uso de las trazas que nos van generando las aplicaciones en la red podemos detectar si una aplicación móvil es maliciosa o benigna. Además en este capítulo haremos una comparativa entre los resultados obtenidos en este experimento con los obtenidos en estudios anteriores.

Los *capítulos 6 y 7* son las conclusiones finales de esta tesis doctoral en castellano y en inglés. Además dentro de estos capítulos nos encontraremos con las vías de trabajo futuro posibles.

«Según vamos adquiriendo conocimiento, las cosas no se hacen más comprensibles, sino más misteriosas.»

Albert Schweitzer (1875-1965)

CAPÍTULO

2

Historia de las amenazas

LOS dispositivos móviles, han adquirido cierta popularidad en los últimos años que ha derivado en nuevos estudios en referencia a la seguridad en ellos. Por esa razón, el número de ataques contra estos dispositivos ha aumentado a lo largo de los últimos años. Debido al incremento tanto en cantidad como en capacidad del *malware* en los últimos años [ASH⁺14], las investigaciones en detección de *malware* para dispositivos móviles se ha convertido en un tema recurrente. Este capítulo presenta varios conceptos básicos acerca del *malware*, tales como los diferentes tipos existentes y sus diferencias.

Seguido, en la sección 2.1 se verá hacia donde ha ido evolucionando la tecnología, siendo punto de partida para los nuevos ataques, haciendo especial énfasis en los vectores de ataque más importantes en este ámbito. En la sección 2.2 haremos una presentación de los dos enfoques más extendidos para la detección de *malware*. Finalmente, en la sección 2.3 resumiremos los principales aspectos de este capítulo.

2.1 Evolución del *malware*

La definición formal de *malware*, palabra compuesta por los términos anglosajones *MALicious* y *softWARE*, hace referencia a:

2. HISTORIA DE LAS AMENAZAS

Cualquier programa informático diseñado con la intención de dañar ordenadores, redes o información [SPDB09].

Si bien en un comienzo se denominaba virus a un *software* maligno, con el paso del tiempo, y debido principalmente a la heterogeneidad de las muestras de este tipo de aplicaciones, se acuñó el término *malware* para agruparlas.

A continuación, vamos a realizar una exposición de los tipos más significativos de lo que se conoce como *malware*: (i) virus, (ii) gusanos, (iii) troyanos, (iv) *spyware*, (v) puerta trasera, (vi) *rootkit* y (vii) *botnet* [DSK11].

2.1.1 *Virus*

El término virus informático lo definió Frederick Cohen en el año 1984 como «Programa que puede infectar a otros programas incluyendo una copia posiblemente evolucionada de sí mismo» [Coh84].

También Mihai Christodorescu definió un virus como «un programa que se replica infectando a otro programa, un sector de arranque o de partición, o un documento que permita macros. Normalmente, añade una copia de sí mismo a los ficheros de la víctima.» [CAJ07].

Concretamente, un virus informático es un tipo de *malware* que tiene por objeto alterar el normal funcionamiento de una computadora, sin el permiso o conocimiento del usuario de la misma.

2.1.2 *Gusanos*

Un gusano es un *malware* que tiene la propiedad de duplicarse a sí mismo y que hace uso de las partes automáticas de un sistema operativo, que generalmente permanecen invisibles al usuario [Erb04].

A diferencia de un virus, un gusano no altera necesariamente los archivos de programas, sino que reside en la memoria y se duplica a sí mismo. Los gusanos casi siempre causan problemas en la red (aunque sea simplemente consumiendo ancho de banda), mientras que los virus siempre infectan o corrompen los archivos de la computadora que atacan.

2.1.3 *Troyanos*

El término troyano o caballo de Troya hace referencia a la historia de Ulises y el caballo de Troya¹. Se trata de un programa ejecutable, generalmente con fines maliciosos, que, aunque no se propaga por sí solo, tiene como objetivo alojarse en las computadoras, permitiendo el acceso a usuarios externos a través de una red local o de Internet [Erb04].

Para que un programa sea catalogado como troyano tiene que acceder y controlar la máquina anfitriona sin ser advertido, normalmente tras instalarse bajo una apariencia inocua como puede ser una aplicación conocida, una imagen o un archivo de música entre otros.

2.1.4 *Spyware*

Por *spyware* se entiende aquel programa que se instala en un ordenador furtivamente para recopilar información acerca del uso del equipo y enviar dichos contenidos a terceros[Erb04].

La información que estos habitualmente recogen va desde mensajes y contactos extraídos del correo electrónico, hasta datos sobre la conexión a Internet, historiales y hábitos de navegación, o incluso, información insertada en formularios (como claves de correo, números de tarjeta de crédito, contraseñas, etc.).

2.1.5 *Puerta trasera*

Una puerta trasera es un programa informático diseñado para superar las políticas de seguridad con el fin de permitir a entidades externas tener control sobre una máquina o una red de forma remota [FS05]. Las puertas traseras pueden ser programas únicos o alojarse dentro de versiones corruptas de programas benignos [DP07].

¹La historia del Caballo de Troya relata la historia de la guerra de Troya en la que los griegos se escondieron dentro de un caballo de madera para asaltar la ciudad de Troya. Los soldados griegos salieron a la noche de dentro del mismo atacando la ciudad mientras los troyanos dormían.

2. HISTORIA DE LAS AMENAZAS

2.1.6 *Rootkit*

Los *rootkits* son un tipo especial de *malware*, que se oculta en archivos específicos o procesos, poniendo en peligro el dispositivo. Para obtener este acceso, normalmente hace uso de vulnerabilidades del sistema operativo para conseguir acceso como administrador del mismo y poder llevar a cabo las acciones maliciosas [HB06].

2.1.7 *Bots*

Un *bot* es un tipo de puerta trasera que participa en un sistema de control a gran escala de máquinas víctima [LJZ09]. Las máquinas infectadas con el programa *bot* se conectan a un canal de control, formando una *botnet* o red de *bots*, una red de máquinas controladas fácilmente al unísono por un solo atacante[CAJ07].

El fenómeno *botnet* (fruto de la combinación de las voces inglesas *software robot networks*) no es casual ni tampoco especialmente reciente. El desarrollo de complejas redes de equipos conectados y las posibilidades computacionales y funcionales que su coordinación puede proveer, no ha pasado por alto para el conjunto de profesionales de la informática ni tampoco para los escritores de *malware*.

En la Tabla 2.1 mostramos los tipos de *malware* más relevantes de los últimos tiempos. En dicha tabla se puede apreciar los diferentes subtipos de cada uno de ellos. Por ejemplo, vemos que ha habido una evolución, desde un sistema parasitario en nuestro ordenador como podían ser los virus convencionales, hasta ya hoy en día ir disfrazados como archivos legítimos, los cuales internamente tienen el código malicioso escondido.

A su vez, en los medios de propagación, vemos como anteriormente los virus más convencionales, necesitaban de un fichero específico o un medio de comunicación infectado para poder propagarse. Posteriormente, surgieron los llamados gusanos generando sus propios medios de autoreplicación en las redes corporativas de empresas, hasta hoy en día que usan técnicas de infección únicamente con el acceso a una web que ya este infectada o que al menos contenga el *payload*¹.

¹Este término se refiere a los efectos destructivos, nocivos o molestos que cualquier virus puede producir cuando ya ha tenido lugar su infección, además de los efectos secundarios de dicha infección (cambios en la configuración del sistema, reenvío de e-mail, ejecución del virus en el arranque del sistema o de *Windows*, etc).

Tabla 2.1: Tipos de Malware y sus características.

	Virus	Gusano	Troyano	Puerta Trasera	Spyware	Rootkit	Botnet
Forma de existencia	Parásito	Entidad independiente	Oculto en otros archivos	Oculto en otros archivos	Oculto en otros archivos	Oculto en otros archivos	Oculto en otros archivos
Método de propagación	Depende del fichero host o los medios de comunicación	Auto replicación	Medios engañosos	Medios engañosos	Medios engañosos	Medios engañosos	Medios engañosos
Objetivo de ataque	Fichero local	Anfitrión de la red o la propia red	Sistema	Sistema	Sistema	Sistema	Sistema
Mayores riesgos	Dañar sistema, borrado de ficheros, pérdida de información	Paralización de la red, pérdida de información	Filtración de información	Filtración de información	Filtración de información	Filtración de información	Filtración de información, dañar el sistema
Velocidad de propagación	Rápida	Muy rápida	Lenta	No	Lenta	Rápida	Muy rápida
Método de detección	Simple	Muy complejo	Complejo	Complejo	Complejo	Complejo	Complejo

2. HISTORIA DE LAS AMENAZAS

Los diferentes objetivos de ataques han ido variando según iban apareciendo nuevos vectores de ataque. Desde los inicios en los que se intentaba atacar un fichero local, hasta intentar infectar todo el sistema para obtener un control remoto del mismo sin conocimiento por parte del usuario.

Algo que prácticamente no ha cambiado, es que en la mayoría de los casos hace falta la intervención de un humano para ejecutar el *payload*. Únicamente, tenemos el caso de los gusanos con los fallos del sistema para replicarse.

Si bien es cierto que los primeros virus mas destructivo..... Por ejemplo un virus informático si puede estar pensado para la destrucción exclusiva de una serie de ficheros específicos del sistema, por contra, una *botnet*, puede estar más orientada a ejecutar peticiones web a un servidor para perpetrar un ataque *DDoS*² a dicho servicio mediante el cual no está destruyendo específicamente ficheros en nuestro sistema.

Como podemos observar en la tabla 2.1, cuando nos vemos infectados por alguno de los diferentes tipos de *malware*, los riesgos a los que nos enfrentamos en cada caso son diversos:

- En el caso de los virus, la finalidad principal es la de dañar sistemas provocando así una pérdida de ficheros dentro del sistema. Un ejemplo representativo de este caso es el virus informático *CIH* también llamado «*Chernobyl*», que estaba diseñado para modificar el sistema *BIOS*¹ consiguiendo que así, que el propio ordenador quedase inservible.
- En cuanto a los gusanos el daño más habitual es la paralización de los servicios de la red local. Este tipo *malware* se caracteriza por consumir muchos recursos de nuestra red consiguiendo colapsarla. A su vez, suelen causar una pérdida de la información contenida en los terminales que están conectados a esa red.
- Entre troyanos, backdoors, spywares, rootkits y botnets podemos encontrar-nos todos los sistemas de selección y robo de la información por parte de los

²También conocido como ataques de denegación de servicio distribuidos, son ataques a un sistema de computadoras o red que causa al servicio o recurso estar inaccesible por los usuarios legítimos.

¹Sistema Básico de Entrada/Salida

atacantes. Dicha información pasa desde direcciones de correo electrónico almacenadas en el equipo, hasta números de tarjetas de crédito.

- Por último, tenemos el caso de las botnets que es algo más especial, ya que son un conjunto de herramientas de *malware*. Por lo que, podemos encontrar con actividades maliciosas de destrucción de equipos, así como la posibilidad de robo o colapso de las redes de una empresa.

2.2 La tecnología cambia

Uno de los cambios tecnológicos más relevantes en la última década, ha sido la incursión de las llamadas tecnologías móviles. Estas, han traído consigo cambios considerables no sólo a nivel de comunicaciones, sino que también han contribuido a un cambio a la hora de trabajar.

En este sentido, Becher et al. [BFH⁺11] definieron los teléfonos móviles como dispositivos que pueden realizar o recibir llamadas usando una tarjeta controlada por un operador móvil.

El uso de dispositivos móviles se encuentra en aumento desde hace años y, con la aparición de los *smartphones*, las capacidades de estos terminales han crecido de manera exponencial. Este cambio permite a los usuarios contar con una conexión a Internet las 24 horas del día, como así también, transportar con ellos una gran cantidad de información en sus bolsillos, que años atrás no solían hacer.

La Figura 2.1 muestra la evolución en la venta de dispositivos móviles a lo largo de los últimos años. Como se observa, la evolución generada durante los últimos años ha hecho que haya habido un incremento en el número de ventas de estos dispositivos móviles.

Todas las virtudes que ofrecen estas plataformas también traen consigo ciertos riesgos, no solo relacionados con los códigos maliciosos, sino también al robo de información o la pérdida del dispositivo en sí.

Las amenazas existentes para las distintas plataformas móviles incluyen distintos tipos de *malware* así como, ataques de phishing, scams y fuga de información. Aunque por lo general, siempre se hace uso de técnicas de ingeniería social para engañar a los usuarios y consumir el ataque.

2. HISTORIA DE LAS AMENAZAS

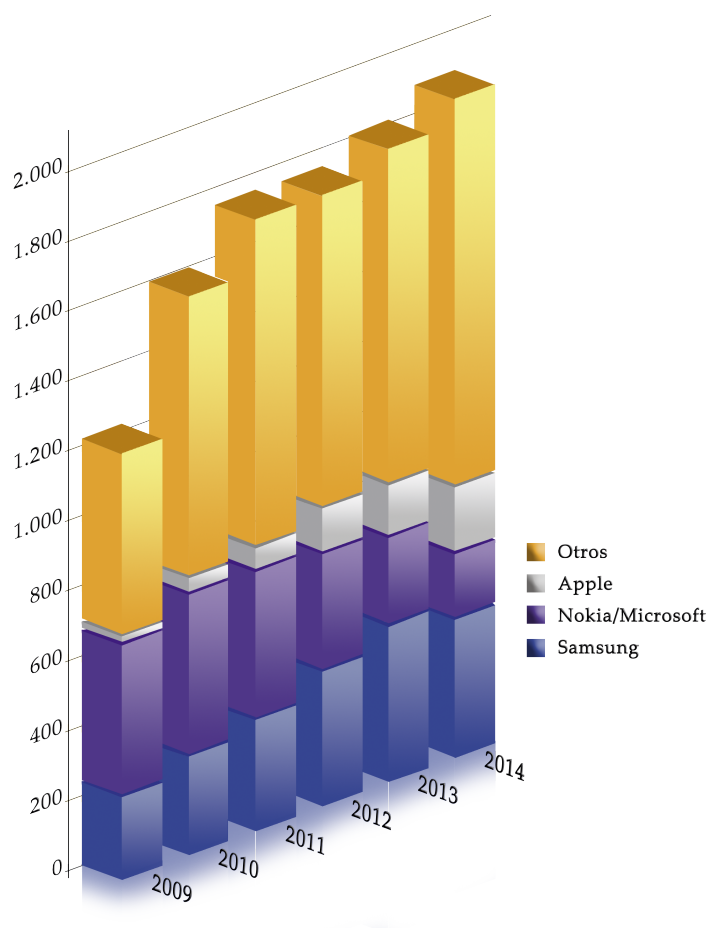


Figura 2.1: Evolución de las ventas de dispositivos móviles a lo largo de los años.

En concreto, en los *smartphones* nos podemos encontrar una gran variedad de sistemas operativos. Estos, son los siguientes: i) *Android* [Dev11], ii) *Symbian* [Har03], iii) *iOS* [HS10], iv) *RIM* [Pas10], v) *Microsoft* [Ded04].

En la Figura 2.2 podemos ver la evolución que han tenido los diferentes sistemas operativos móviles desde enero del 2012 hasta marzo del 2014. Una de las observaciones más llamativas a primera vista de toda la gráfica, es el incremento escalonado que sigue siempre el sistema operativo de *Apple*. Asimismo podemos

ver un incremento en la venta de dispositivos móviles en los meses de septiembre, algo que no se observa en el resto de meses del año. Por otro lado nos encontramos con el sistema operativo *Android*, que por norma general tiene una tendencia creciente, pasando desde casi un 50 % de los usuarios a casi un 70 % de los mismos. En este caso, se produce justamente lo contrario que con *iOS*, ya que durante los meses de septiembre de cada año podemos ver una disminución. Esto, se puede deber en gran medida a que gracias a las *Keynotes*¹ se venden más dispositivos, ya que implican la presentación de novedades.

En el caso de Blackberry podemos ver un decremento continuo hasta casi desaparecer habiendo llegado a tener una cuota de mercado cercana al 10 %.

Otro de los mayores decrementos proviene de sistemas operativos móviles *Symbian* y *Palm OS* (en la gráfica como *Otros SO*), que con una cuota de mercado del 20 % llegan casi al punto de la desaparición.

Por último, el caso más curioso de todos es el de *Windows Phone*, un sistema operativo que tiene un incremento muy lento pero progresivo. Con una cuota de mercado menor al 5 % poco a poco va llegando casi al 10 % de la cuota de mercado.

2.2.1 El *malware* en las nuevas tecnologías

La primera aparición de un software malicioso para dispositivos móviles fue en el año 2004. En concreto, se denominó *Cabir*¹ y estaba escrito por el grupo conocido como 29A y desarrollado para el sistema operativo *Symbian*.

Este *malware*, se propagaba usando el *Bluetooth*, y uno de los daños ocasionados era la pérdida rápida de la batería del sistema, con una consecuencia directa en la disminución de la vida útil de la misma.

Posteriormente, se empezaron a crear diferentes códigos maliciosos para otros sistemas. En la gran mayoría de los casos, la propagación del *malware* se hacía

¹Una Keynote es una charla que establece el principal tema. En el caso de los entornos corporativos o comerciales, es un discurso o conferencia con una mayor importancia. La tónica de la Keynote establece el marco para siguientes eventos, etc. En el caso de *Apple* las *Keynote* son el evento más esperado del año, puesto que en el mismo se mostrarán los nuevos dispositivos y avances de la compañía para el siguiente año.

¹<https://www.f-secure.com/v-descs/cabir.shtml>

2. HISTORIA DE LAS AMENAZAS

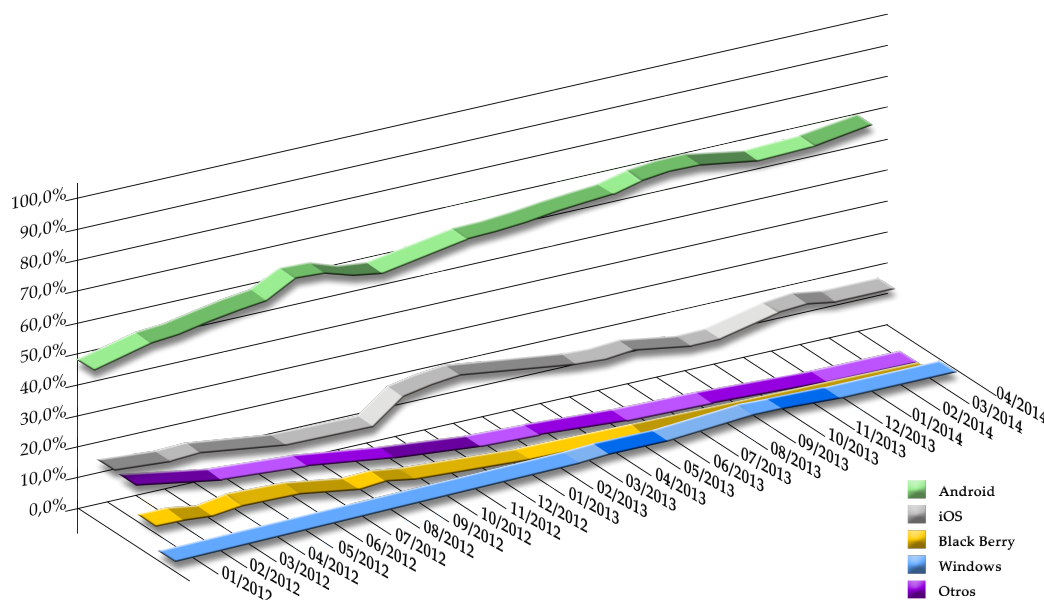


Figura 2.2: Evolución de S.O. móvil

a través del *Bluetooth*, los mensajes de texto o *SMS* o los mensajes multimedia o *MMS*. (Citas)

En el año 2004, Guo et ál.[GWZ04] trabajaron en una aproximación acerca del daño causado por estos códigos maliciosos en los sistemas y sus posibles defensas. El artículo ilustra los ataques específicos de telefonía móvil, tales como: i) la violación de la privacidad, ii) el robo de identidad, ii) el ataque de denegación de servicio al sistema de emergencias, así como, iii) las crisis nacionales.

Más tarde, concretamente en el año 2007, Milligan et ál.[MH08] presentaron un estudio sobre los riesgos que nos encontramos al usar dispositivos móviles en un negocio. Los riesgos más relevantes extraídos a partir de esta investigación son: i) la fuga intencionada o desintencionada de información, ii) el robo de información, iii) los ataques perpetrados por *malware* financiero, iv) los ataques de infección de la red, v) los ataques de congestión de la red a través del envío masivo de spam.

Asimismo, tanto los dispositivos móviles, como los ordenadores convencionales, comparten una gran cantidad de características debido principalmente, a su

hardware y a su software interno. En el ámbito de la seguridad ocurre algo similar, aunque existen aspectos propios de los dispositivos móviles.

En 2011, Becher et al. [BFH⁺11] establecieron algunas propiedades específicas de los sistemas móviles que podemos ver en la Figura 2.3.



Figura 2.3: Características específicas de la seguridad móvil.

A continuación, explicamos brevemente las diferencias encontradas entre dispositivos móviles y los ordenadores convencionales[]: i) los recursos limitados, ii) los costes asociados al usuario, iii) los diferentes vectores de ataque, iv) la interfaz de usuario, v) el entorno de red, vi) la reputación del operador móvil, y vii) otras capacidades de los dispositivos móviles.

- *Recursos limitados:* similares a los PC, los teléfonos móviles tienen recursos como CPU, memoria RAM, la memoria, los algoritmos, la batería, por nombrar algunos. Aún disponiendo de unos recursos menores a los de un ordenador convencional, las capacidades de estos dispositivos han aumentado rápidamente en los últimos años. Como consecuencia de estas limitaciones,

2. HISTORIA DE LAS AMENAZAS

los atacantes pueden hacer uso de ellas para dejar sin servicio el dispositivo. Además, estas limitaciones, implican que la detección y la defensa de las mismas sea más compleja que en un ordenador convencional.

- *Los costes asociados*: Una de las motivaciones principales de los atacantes es el lucro económico. Más concretamente, en los dispositivos móviles, un atacante puede generar costes para el usuario, transformándolos en ingresos para él mismo. Esta técnica se puede llevar a cabo mediante: i) llamadas de tarificación adicional para el usuario, ii) suscribirle a plataformas de mensajería *premium* o iii) sistemas de pago adicionales.

En 2005, Jamaluddin et ál. [JZEC04], realizaron una comparación entre los efectos de los troyanos entre ordenadores y dispositivos móviles. Una de las principales conclusiones obtenidas fue que en el PC los troyanos afectan la velocidad y el rendimiento de la red, mientras que, por contra, en el ámbito de los dispositivos móviles afectan más a los costes económicos que tendría que asumir el usuario. Particularmente, los autores plantean su argumento mediante el desarrollo de un troyano que se encuentra dentro de una aplicación, el cual envía mensajes SMS o MMS, con un costo adicional para el usuario.

- *Vectores de ataque*: A diferencia de los vectores de ataque tradicionales, los dispositivos móviles tienen algunos vectores no tradicionales que pueden hacer que un código malicioso se propague rápidamente a través de ellos como por ejemplo, los SMS, los MMS, el *Bluetooth* y las aplicaciones basadas en IPv4¹.
- *Interfaz de usuario*: Otra de las distinciones más significativas es la diferencia de tamaños entre dispositivos; por lo que, los mecanismos de seguridad aplicados para PCs como indicadores visuales en los navegadores, barras de direcciones, o *CAPTCHAS*² pueden no ser directamente aplicables al dispo-

¹IPv4 es la cuarta versión del protocolo Internet Protocol. Dicha versión del protocolo usa direcciones de 32 bits.

²Siglas de Completely Automated Public Turing test to tell Computers and Humans Apart (prueba de Turing completamente automática y pública para diferenciar computadoras de humanos)

sitivo móvil. Esto, también implica la necesidad de rediseñar para pantallas más pequeñas. Además, esto exige una mayor atención a la seguridad usada. Felt et al.[FW11] muestran que las limitaciones en la interfaz de usuario móvil hacen que sea más fácil para un atacante realizar ataques de phishing que en los navegadores de escritorio.

- *Entorno de red*: Es el entorno que nos encontramos entre el dispositivo y el operador de red móvil (*MNO* de sus siglas en inglés *Mobile Network Operator*). Este, juega un papel importante en los teléfonos móviles, ya que por ejemplo, los procesos de actualizaciones de firmware y administración remota de dispositivos son controlados por el MNO. Esta fuerte influencia de MNO sobre el dispositivo trae una nueva dimensión en cuanto a ataques se reciben.

En este sentido el MNO, previniendo ataques relacionados con vulnerabilidades en el firmware, realiza actualizaciones periódicas del mismo de forma remota en los dispositivos, liberando así al usuario de esta tarea.

- *Reputación*: En el caso de los teléfonos inteligentes, la reputación de MNO juega un papel clave. Cuando un teléfono móvil está infectado por *malware*, este podría ser utilizado para actividades maliciosas sin conocimiento del usuario. Teniendo en cuenta que el MNO cobrará por cada evento que se produzca, independientemente de si es una acción del usuario legítimo o una actividad llevada a cabo por un software malicioso que se está ejecutando de manera oculta en el dispositivo.

Si en vez de ponernos en el lado del operador móvil, nos ponemos en el lado del usuario (la acción más normal), veríamos que es el propio operador el que nos está cobrando un suplemento por una serie de actividades que a priori, no hemos realizado, pero que en realidad está llevando a cabo nuestro dispositivo infectado. Estos hechos son los que podrían hacer que la reputación de nuestro MNO se vea afectada.

- *Otras capacidades*: Los dispositivos móviles también son más vulnerables debido a los sensores que dispone. En el caso de un ordenador convencional, estos sensores serían periféricos externos que conectaríamos al mismo, por

2. HISTORIA DE LAS AMENAZAS

contra en los dispositivos móviles, estos periféricos son una parte esencial en las capacidades del mismo.

2.2.2 Vectores de amenaza

Las amenazas de seguridad móviles podrían ser (i) físicas, (ii) basadas en la conectividad de red o (iii) mediante el uso de *malware*. El vector de ataque es un medio por el cual un atacante puede obtener acceso a un sistema. Becher et al. [BFH⁺11] dividen los ataques a un dispositivo móvil en las categorías siguientes:

2.2.2.1 Basados en el *hardware*

Estos ataques están más relacionados con el acceso físico del dispositivo, como la interceptación de la tarjeta SIM¹ del dispositivo. La extracción del código de desbloqueo de la *SIM* en los dispositivos *iPhone* o los ataques de «*Man in the Middle*» son algunos de los ejemplos que encontramos en este tipo de ataques.

2.2.2.2 Independientes del dispositivo

Los ataques que son independientes del dispositivo como por ejemplo en la infraestructura, protocolos etc. entran en esta categoría.

El Sistema Global para las Comunicaciones Móviles (GSM de sus siglas en inglés, *Global System for Mobile communications*) fue desarrollado hace 25 años y tiene una cantidad elevada de vulnerabilidades, que se han ido detectando en diversos puntos de la infraestructura, como por ejemplo, el sistema de criptografía asimétrica, el cual está anticuado, o también, la no autenticación de un dispositivo en la red.

Del mismo modo, la infraestructura de envío de mensajes SMS se puede usar como un canal de búsqueda que puede sobrecargar la red de una forma rápida. En el caso de la infraestructura MMS, podemos ver que dichos fallos nos pueden llevar a

¹Subscriber Identity Module, en español módulo de identificación de abonado es una tarjeta inteligente desmontable usada en teléfonos móviles. Las tarjetas SIM almacenan de forma segura la clave de servicio del suscriptor usada para identificarse ante la red, de forma que sea posible cambiar la línea de un terminal a otro simplemente cambiando la tarjeta.

una descarga rápida de la batería del dispositivo, generando con ello una reducción en la vida útil de la misma.

2.2.2.3 Basadas en el software

Estos ataques se basan en la explotación de las vulnerabilidades del software que se ejecuta en los dispositivos móviles. Como se explicó anteriormente, el Software malicioso *Cabir* se propagaba automáticamente por los sistemas operativos Symbian en el año 2004. Algunos de los canales usados para esta propagación son los siguientes: i) canales de comunicación *SMS*, ii) canales de comunicación *MMS* iii) ataques vía navegador web y iv) ataques mediante *Rootkits*.

2.2.2.4 Basada en el usuario

En esta categoría englobamos a todos los ataques que usan la ingeniería social para engañar al usuario y no explotar de esta forma ninguna vulnerabilidad [MS11].

Cuando se analiza la seguridad del sistema, también es importante focalizarse en la modelización de los ataques; que podemos definir como el análisis de las capacidades de un atacante y cuales son los límites del mismo.

Dicho atacante puede ser i) pasivo, que no altera el contenido, o ii) activo, el cual altera o elimina parte del contenido del dispositivo.

2.3 Sumario

En este capítulo hemos revisado la literatura acerca de la evolución del *malware*. En concreto, hemos comenzado revisando conceptos básicos relativos al *malware*: sus tipos, sus ataques, y su evolución a lo largo de los años. En este último punto hemos hecho una descripción de los vectores de ataque que encontrados en los dispositivos móviles.

Reescribir esto. Así, hemos estudiando los diferentes tipos encontrados a lo largo de los años. En particular, hemos hecho una definición de estas muestras, comparándolas a su vez con el resto para ver en qué difieren cada una de ellas. Dentro de ellas hemos visto los daños ocasionados por estos tipos de *malware* a nuestros sistemas.

2. HISTORIA DE LAS AMENAZAS

Seguidamente, hemos resumido la evolución de la tecnología para centrarnos en los dispositivos móviles. En dicha evolución hemos comprobado como la venta de los dispositivos ha ido aumentando de forma exponencial durante los últimos años. Seguido, hemos hecho un estudio de las diferentes muestras de *malware* aparecidas en los dispositivos móviles. Igualmente, hemos listado las diferencias encontradas entre estos dispositivos y los ordenadores convencionales.

Finalmente, hemos realizado un resumen de los vectores de amenazas más importantes dentro de estos dispositivos. Hemos comprobado las cuatro partes en las que un dispositivo móvil está amenazado: a través del *hardware*, de entorno, del software o del usuario.

«Si no conozco una cosa, la investigaré.»

Louis Pasteur (1822 – 1895)

CAPÍTULO

3

Estado de la técnica en la plataforma *Android*

DESDE que saliera al mercado el sistema *Android*, han sido muchas las investigaciones que se han llevado a cabo sobre el. Habitualmente, los desarrolladores de malware suelen centrar sus esfuerzos en la plataforma que más es usada en ese momento [Emm14]. Como consecuencia a esto, la comunidad científica ha orientado sus investigaciones en dicho sistema, viendo nuevas técnicas de detección de *malware* en el mismo. En un primer momento, en la sección 3.1 haremos una evolución del *malware* dentro del sistema *Android* como primera aproximación a los enfoques llevados a cabo en la detección de *malware*.

Acto seguido, en la sección 3.3 describiremos los cuatro vectores de infección expuestos por Becher et ál. [Bec10] en la seguridad móvil. En la sección 3.4 explicaremos las diferentes investigaciones sacadas de la literatura, divididas en los dos enfoques más extendidos. Para terminar, en la sección 3.5 resumiremos los principales aspectos de este capítulo.

3.1 Evolución del *malware* en *smartphones*

Desde que apareció, el considerado primer *malware* para un dispositivo móvil, *Libertad crack*, en el año 2000, el *malware* evolucionó rápidamente durante siguientes años[LPMS13]. A partir de ese momento, una cantidad de *malware* fue evolucionando para focalizarse en los dispositivos móviles.

Por ende, en estos nuevos dispositivos, el *malware* sigue catalogándose como en los ordenadores de sobremesa. Entre los años 2004 y 2013, los laboratorios de Kaspersky habían añadido un total de 200.000 muestras a su base de datos, mientras que después en un sólo año añadieron un total de 295.539 muestras a su base de datos [Emm14]. Esto nos demuestra que estamos siendo partícipes de un aumento drástico de *malware* para dispositivos móviles.

La llegada de *Cabir* [Gos06] en el año 2004 para el sistema operativo Symbian, confirmó la afirmación de que iba a haber una evolución del *malware* para ordenador. Con estas premisas, para que un programa malicioso infecte un sistema se deben cumplir tres condiciones[Moh14]: i) la plataforma debe ser popular, ii) la herramientas de desarrollo deben estar bien documentadas y iii) la existencia de vulnerabilidades o errores en la codificación.

- *La plataforma debe ser popular.* Cuando apareció *Cabir* era *Symbian* el sistema operativo predilecto en los dispositivos móviles. Hoy en día en cambio, es la plataforma *Android* la que dispone de una cuota de mercado superior [Sta15], convirtiéndose así en la plataforma objetivo de los desarrolladores de *malware*.
- *Herramientas de desarrollo bien documentadas.* Una de las principales características para que una plataforma móvil triunfe es la facilidad de desarrollo en la misma. Por tanto, es necesario la existencia de herramientas de desarrollo útiles y con una documentación sencilla para poder desarrollar en la misma. Además, otra de las necesidades es la creación de comunidades para proporcionar documentaciones sobre la plataforma.
- *La existencia de vulnerabilidades o errores en la codificación.* Teniendo en cuenta que *Cabir* se aprovechaba de las vulnerabilidades que nos podíamos

3.1 Evolución del *malware* en *smartphones*

encontrar en el sistema Symbian, no es raro pensar que esto pueda suceder en el sistema *Android* [FG13].

Desde el año 2004, los dispositivos móviles han sido el objetivo principal de los desarrolladores de *malware*. En junio de dicho año, apareció el llamado primer virus para dispositivos móviles en la plataforma SymbianOS apodado con el nombre de «*Cabir*»[Gos06].

El *malware* en dispositivos móviles ha ocasionado diferentes investigaciones a lo largo del tiempo. Securelist[Gos06] y Shih[SLCS08] describieron la evolución del *malware* entre 2004 y 2006. Entre los periodos de 2004 al 2010, Hyp-ponen categorizó en total 517 familias de virus, gusanos y troyanos para estos dispositivos[Hyp10]. Schmidt y Albayrak [SA08] presentaron una lista completa de *malware*. Felt et al.[FFC⁺11] hizo una descripción detallada de lo que sería el *malware* móvil que se extendía desde enero de 2009 hasta junio de 2011. Polla et al.[LPMS13] creó un estado del arte sobre las amenazas, las vulnerabilidades y las soluciones de seguridad para dispositivos móviles durante el período de 2004 a 2011.

Como consecuencia a dichos estudios se concluyó que la cuota de mercado que ostenta un sistema operativo es crucial en la cantidad de aplicaciones maliciosas que nos podemos encontrar en el mismo. A más cuota de mercado, dicho sistema operativo tiene más posibilidades de tener *software* malicioso entre sus aplicaciones. La Figura 3.1 nos muestra las cuotas de mercado obtenidas por los sistemas operativos a comienzos del año 2015¹.

Una de las principales consecuencias del aumento de los terminales móviles, ha sido el incremento en los ataques a los mismos. Maslennikov [Mas11] demostró que los ataques orientados a los *smartphones* aumentaron un 65,12 % entre los años 2009 a 2010, casi duplicándose en los siguientes 17 meses.

Fue en Agosto del 2010, cuando el laboratorio Kaspersky identificó el primer troyano para la plataforma *Android* [Lab10]. Este *malware* se hacía pasar por una aplicación de reproducción de vídeos para poder infectar el terminal. A partir de ese

¹Datos obtenidos de la web <http://www.statista.com/statistics/266970/market-share-forecast-of-smartphone-operating-systems-from-2010-to-2015/>

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

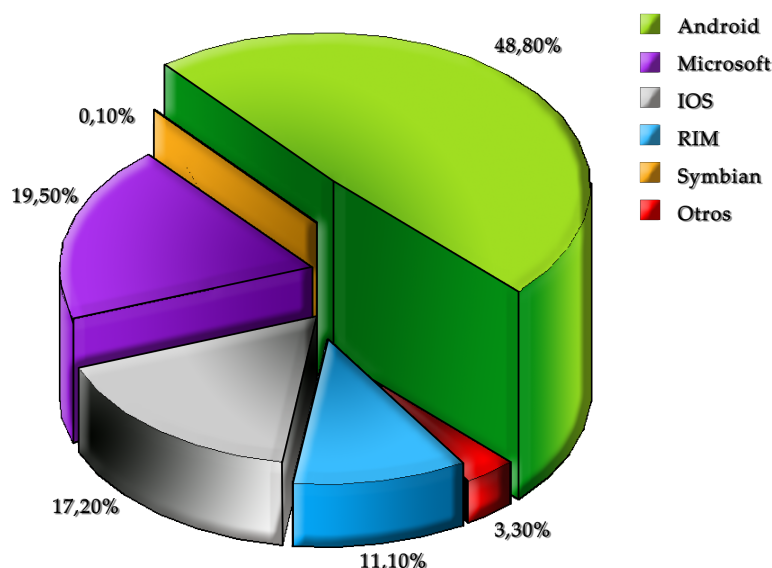


Figura 3.1: Cuota de mercado por dispositivo móvil.

momento, el *malware* para *Android* se convirtió en el predominante en los sistemas móviles.

Más tarde, el año 2013 el incremento de *malware* bancario en los dispositivos móviles hizo que las compañías antivirus incrementaran sus investigaciones en este ámbito [CU13]. En la Figura 3.2 podemos ver el incremento de las muestras de *malware* bancario durante el año 2013 únicamente en la plataforma *Android*.

En el año 2014, el 53 % de las muestras de virus detectadas por Kaspersky Labs se correspondían con muestras pensadas para el robo de dinero al usuario [Emm14]. A lo largo de ese año, se empezó a usar técnicas hasta entonces usadas en los ordenadores comunes.

Una vez definidas en el capítulo 2 las amenazas existentes en los dispositivos móviles, y habiendo seleccionado el sistema *Android* como el sistema sobre el que basar nuestras investigaciones.

Este capítulo desarrolla el funcionamiento del mismo, explicándolo desde 2 puntos de vista: i) desde su propia arquitectura y ii) desde la seguridad implemen-

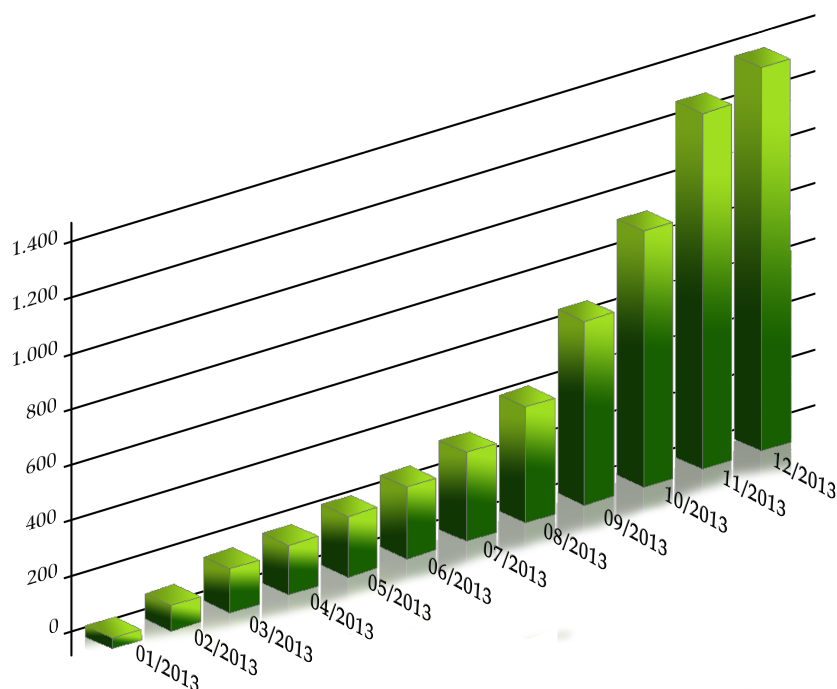


Figura 3.2: Incremento de muestras en *malware* bancario para dispositivos móviles en el año 2013.

tada en el sistema.

La primera parte explica las capas que componen el sistema. Por otro lado en la segunda parte, se hace un estudio de la seguridad dentro del sistema haciendo hincapié en las partes más importantes de la protección del mismo.

3.2 La arquitectura de *Android*

La plataforma móvil de *Google* hizo su aparición en el mercado en septiembre de 2008 [Mor08], teniendo un crecimiento importante desde el primer momento. En su creación el sistema *Android* tuvo como referencia al sistema operativo libre Linux.

Basándose en dicho sistema, *Android* contiene las siguientes capas [Dev11]: i) capa de aplicaciones (escritas en java y ejecutadas en *Dalvik*), ii) capa de servicios y librerías (escritos mayormente en *Java*), iii) capa de librerías nativas, demonios y servicios (escritos en *C* y *C++*) y iv) *kernel* de *Linux*.

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*



Figura 3.3: Arquitectura del sistema operativo *Android*

La Figura 3.3¹ muestra un esquema de la arquitectura del sistema *Android* con ejemplos de los diferentes módulos que existen en cada capa del sistema operativo [GS11]: i) la capa de aplicaciones, ii) La capa de servicios y las librerías, iii) las bibliotecas, iv) el *runtime* de *Android* y v) el *kernel* de *Linux*.

La capa de aplicaciones está formada por todas las aplicaciones del sistema. Programadas en *Java*, son las herramientas con las que el usuario ejecuta directamente. Entre estas aplicaciones nos podemos encontrar las básicas de teléfono, así como los juegos de última generación como por ejemplo *Angry birds*².

Dentro del sistema *Android*, los desarrolladores pueden disponer de una serie de *APIs*³ usadas a su vez por las aplicaciones base. Cuando se creó *Android*, se diseñó de tal forma que la reutilización de componentes fuera sencilla. Esta carac-

¹Obtenida de la propia documentación oficial de *Android*.

²<https://play.google.com/store/apps/details?id=com.rovio.angrybirds>

³Interfaz de programación de aplicaciones que ofrece cierta biblioteca para ser utilizado por otro *software* como una capa de abstracción.

terística hace que una aplicación pueda acceder a servicios de otras aplicaciones o viceversa [EOM09b].

Dentro de esta capacidad extra que se les da a las aplicaciones hay una serie de restricciones del propio framework que todas las aplicaciones tienen que cumplir [FBL⁺14]: i) una aplicación se ejecuta con un único *user-id* para mitigar fallos en la programación, ii) el acceso a un componente está restringido hasta la asignación del permiso correspondiente, iii) las aplicaciones se comunican mediante el sistema de comunicaciones inter-procesos o *IPC* (*inter-process communication* de sus siglas en inglés).

Android incluye un conjunto de bibliotecas, desarrolladas en *C/C++*, usadas por varios componentes del sistema. Los desarrolladores hacen uso de este tipo de funcionalidades a través del propio *framework* de *Android*. Las librerías más importantes son las siguientes: i) *System C library* (implementación de la biblioteca C estándar), ii) bibliotecas de medios, iii) bibliotecas de gráficos, iv) *SQLite* (sistema de gestión de bases de datos relacional) y v) bibliotecas para el uso de *3D*.

El sistema operativo de *Android* ha añadido una serie de bibliotecas principales, las cuales nos proporcionan una serie de funcionalidades disponibles a través del lenguaje de programación *Java*.

Cuando se ejecuta una aplicación dentro del sistema *Android*, dicha aplicación creará un proceso con una instancia propia de la máquina virtual *Dalvik*. Este sistema operativo está creado de tal forma que un dispositivo pueda correr múltiples máquinas virtuales a la vez de una forma totalmente eficiente.

Para ello, la máquina virtual *Dalvik* ejecuta los ficheros *Dalvik executable* (*.dex*), los cuales, están optimizados para consumir el mínimo de memoria *RAM* en el sistema. Esta máquina está basada en registros, ejecutando clases compiladas en *Java* y transformadas al formato *dex*.

Como hemos expuesto en el capítulo 2, este sistema operativo es una herencia de *Linux*. En sí, el kernel usado es una modificación de la versión 2.6 del núcleo de *Linux*. Algunas de las modificaciones llevadas a cabo para cumplir las necesidades especiales de la plataforma han sido [Bra10]: i) correcciones de errores, ii) mejora en el espacio de usuario, iii) nuevas infraestructuras y iv) el soporte para nuevos sistemas integrados.

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

Al igual que el resto de *Android*, el kernel está libremente disponible y el proceso de desarrollo es visible a través del repositorio público fuente de *Android*¹. El sistema hace uso de diferentes módulos de *Linux* y servicios del mismo. Los desarrolladores no pueden programar en esta capa directamente. Además, el núcleo también actúa como una capa de abstracción entre el *hardware* y el resto de la pila de *software*.

3.2.1 La seguridad en *Android*

Tener un sistema operativo seguro, requiere una arquitectura de seguridad robusta y ciertos programas para su correcto funcionamiento. *Android* fue planteado con la seguridad multicapa, proporcionando así la flexibilidad necesaria para las plataformas abiertas y además, proporcionando la protección a todos los usuarios de la plataforma [EOM09b].

Android si implementó pensando en los desarrolladores; y por ello, se crearon los controles de seguridad que reducen significativamente la carga a esta clase de usuarios del sistema *Android*. De esta forma, si el usuario no está familiarizado con la seguridad en los dispositivos móviles, los controles de seguridad ya existentes en el sistema aportan ese grado de confianza necesario.

Además de en los desarrolladores, *Android* fue desarrollado para los usuarios del dispositivo. Estos disponen de visibilidad en el funcionamiento de las aplicaciones, y control sobre las mismas. A su vez, este desarrollo fue realizado teniendo en cuenta la posibilidad de que un atacante usara la ingeniería social para engañar a un usuario y así instalarse un *malware*.

Desde el principio, el equipo principal de desarrollo de *Android* reconoció que un modelo de seguridad robusto fue necesario para permitir un ecosistema vigoroso de aplicaciones y dispositivos construidos sobre y alrededor de la plataforma *Android* y el apoyo de servicios en la nube.

El equipo encargado de la seguridad en *Android* ha tenido la oportunidad de observar como otros sistemas, ya sean móviles, de escritorio o servidores, han ido previniéndose de las posibles fallas en seguridad que se iban encontrando. Por ello, este equipo formó un protocolo de seguridad específico para hacer frente a los

¹<https://android.googlesource.com/>

puntos débiles encontrados en otras plataformas. Las partes más importantes que incluye el protocolo son [SFK⁺09]: i) análisis del diseño, ii) test de penetración y revisión del código, iii) ser *open source* y revisiones por la comunidad y iv) una respuesta a incidentes.

- **Análisis del diseño.** El proceso de seguridad en la plataforma *Android* empieza con un diseño del modelo de seguridad robusto y configurable. Cada módulo, característica importante de la plataforma, es revisado y diseñado basándose en los controles de seguridad apropiados e integrados por la arquitectura del sistema.
- **Test de penetración y revisión del código.** Las revisiones son llevadas a cabo por el equipo de seguridad de *Google* y consultores independientes.

El principal motivo de estas revisiones, es la identificación de debilidades y vulnerabilidades mucho antes de que la plataforma sea comercializada al resto del mundo.

- ***Open source* y revisión por la comunidad.** La plataforma *Android* es un proyecto de código abierto, permitiendo por parte de la comunidad, hacer una revisión de la seguridad del sistema. El empleo de tecnologías de código abierto externas, como por ejemplo el *kernel* de *Linux*, permite que también haya habido auditorías de seguridad externas. A su vez, la propia plataforma de *Google Play*, ofrece un foro en el cual, los usuarios y las empresas pueden proporcionar información específica acerca de las aplicaciones.
- **Respuesta a incidentes.** Aún con estos protocolos y precauciones llevadas a cabo por parte del equipo de *Google*, pueden surgir problemas de seguridad a la hora de enviar una nueva actualización de una aplicación. Para ello, el sistema está respaldado por un equipo que supervisa constantemente los nuevos foros sobre seguridad general para encontrar posibles vulnerabilidades en el sistema. En cuanto se descubre una de estas vulnerabilidades, dicho equipo tiene un proceso de respuesta para la mitigación de la vulnerabilidad, con el fin de minimizar el riesgo potencial de todos los usuarios de *Android*.

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

Este tipo de respuestas se realizan mediante el sistema soportado en la nube *OTA*¹ (*Over The Air* de sus siglas en inglés)[ABC⁺03], lo que significa que permite de forma automática la eliminación, actualización del sistema *Android* y de aplicaciones de la plataforma *Google Play* de forma remota.

La plataforma *Android* se aprovecha de la protección basada en el usuario que *Linux* dispone en sus sistemas, como medio para identificar y aislar los recursos de las aplicaciones. El sistema *Android* asigna un identificador único de usuario (*UID*) por cada aplicación de *Android* y lo ejecuta como ese usuario en un proceso separado [Poc11]. Este enfoque, difiere de otros sistemas operativos (incluyendo la configuración de *Linux* tradicional), donde diferentes aplicaciones se pueden ejecutar con los mismos permisos de usuario.

Esto establece un nivel de *kernel* para la *sandbox* de aplicaciones. El *kernel* del sistema es el encargado de aplicar la seguridad entre las aplicaciones y el sistema, basándose en los *ID* de usuario y el grupo asignado a dichas aplicaciones. En una primera instancia, las aplicaciones instaladas no pueden interactuar entre ellas mismas, teniendo a su vez un acceso limitado a los recursos del sistema operativo. Cuando una aplicación A intenta acceder a los datos de una aplicación B, por ejemplo hacer uso del teléfono, el sistema operativo, como encargado de la gestión de la seguridad, protege el sistema comprobando si la aplicación A tiene los privilegios necesarios para poder llevar a cabo esas acciones.

Como la aplicación de la *sandbox* se encuentra dentro del núcleo, el modelo de seguridad proporcionado se extiende al código nativo y a las aplicaciones del propio sistema operativo. Todo el *software* que haya por encima del propio *kernel* se ejecutará dentro de esta *sandbox*. En otras plataformas, como por ejemplo iOS, los desarrolladores se ven obligados a crear un marco específico, un conjunto de *APIs* o incluso un lenguaje propio con el fin de reforzar la seguridad del sistema. En *Android* no hay restricciones sobre la escritura de una aplicación para reforzar la seguridad, tanto el código nativo como el interpretado son igual de seguros [FBKF12].

¹El sistema *Over The Air* se conforma de varios métodos de distribución inalámbrica de nuevas actualizaciones de *software*, la configuración, y la actualización de claves de cifrado. Chan a dispositivos como teléfonos móviles

Normalmente, en muchos sistemas operativos, suelen aparecer errores en la memoria del sistema que ponen en compromiso la seguridad del dispositivo. *Android* por contra, debido a la gestión que realiza sobre la ejecución de las aplicaciones, ejecutándolas en *sandbox* diferentes. Este paradigma implica que si hubiese un fallo de corrupción en la memoria y dejase ejecutar código arbitrario, únicamente permitiría ejecutarlo en el contexto de dicha aplicación, con los permisos asociados por el sistema operativo.

Al igual que todos los elementos de seguridad, una *sandbox* no está exenta de vulnerabilidades. Aún así, para poder sobrepasar la seguridad que nos ofrece este sistema, siempre y cuando este bien configurado, es necesario poner en peligro la seguridad del propio *kernel* de *Linux*.

Por defecto en *Android*, solamente el *kernel* y una serie de aplicaciones centrales se ejecutan con permisos de administrador o *root*. El propio sistema no puede impedir que un usuario o una aplicación con permisos de *root* pueda modificar el sistema operativo, el *kernel* o cualquier parte de otras aplicaciones. Por lo tanto, el acceso *root*, implica el control total de todas las aplicaciones, los datos, etc. siendo un peligro de seguridad el hecho de que cualquier usuario pueda permitir a las aplicaciones disponer de estos permisos. Ya no solo en la posibilidad de que una aplicación maliciosa haga uso de estos permisos, sino por un posible fallo en cualquiera de las aplicaciones.

Una de las necesidades principales de los desarrolladores es la posibilidad de poder modificar un sistema operativo *Android*. Muchos dispositivos ofrecen la posibilidad de desbloquear el sistema de arranque con el fin de poder instalar otro sistema operativo diferente al existente. Estos nuevos sistemas operativos suelen permitir al usuario tener el acceso *root* cuando depuran una aplicación o cuando necesitan acceso a componentes del sistema o funciones de la *API* que no suelen ser mostradas.

A su vez, cuando existe un usuario *root* en el sistema, éste puede tener acceso a toda la información contenida en el dispositivo. Por lo tanto, cuando una aplicación maliciosa accede a un dispositivo con acceso *root*, puede hacer uso de esos privilegios para obtener la información sensible del terminal.

Una de las técnicas llevada a cabo para la no obtención de dicha información es el cifrado de los datos almacenados en el dispositivo mediante el uso de una

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

clave[FBKF12].

Otro enfoque empleado para la protección de los datos del usuario es haciendo uso de soluciones *hardware* [Rol12]. Por ejemplo, el uso del sistema *NFC*¹, se puede hacer un almacenamiento de confianza con *Google Wallet*².

Si por un casual el dispositivo ha sido extraviado o robado, los datos que hayan sido cifrados en el sistema mediante la contraseña, serán inaccesibles aún cuando se modifique el sistema de arranque del terminal.

3.2.2 Las características de la seguridad del usuario

- *El cifrado del sistema de ficheros*

A partir de la versión 3 de *Android*, se proporcionó un sistema de cifrado de ficheros. Todos los datos del usuario se pueden cifrar desde el núcleo con *AES128*³ [DR02].

Para utilizar este cifrado, el dispositivo tiene que estar haciendo uso de la contraseña de usuario, no es válido el bloqueo por patrones.

- *La protección por contraseña*

Android puede ser configurado para verificar una contraseña suministrada por el usuario antes de proporcionar el acceso a un dispositivo. Además de prevenir el uso no autorizado del dispositivo, esta contraseña protege la clave de cifrado usada en el cifrado del sistema de archivos completo [ACS09].

- *Administración del dispositivo*

Desde la versión de *Android 2.2*, el sistema ha ido proporcionando una *API* de administración de dispositivos, que proporciona una gestión del terminal a nivel del sistema [Dev11].

¹*Near Field Communication* es una tecnología de comunicación inalámbrica, de corto alcance y alta frecuencia que permite el intercambio de datos entre dispositivos.

²Sistema de pago móvil creado por Google que permite a sus usuarios almacenar tarjetas de débitos, tarjetas de crédito, tarjetas de fidelidad, y tarjetas regalo entre otras cosas, así como una redentora promoción de ventas en su teléfono móvil.

³El sistema *AES* es un esquema de cifrado por bloques.

3.2.3 La seguridad *Android* a nivel de aplicación

Para poder ejecutar una aplicación dentro del sistema, normalmente son desarrolladas mediante el lenguaje de programación *Java* y ejecutadas dentro de la máquina virtual *Dalvik*. Otra forma de creación de aplicaciones es mediante el uso de código nativo, aunque bien es cierto que no suele ser lo habitual. Una vez implementadas, se generará el fichero con extensión *APK* necesario para la instalación de la aplicación en el terminal [Mur08].

A continuación vamos a presentar los puntos más relevantes de la seguridad en *Android*, basándonos en: i) el modelo de permisos de *Android*, ii) la información personal y iii) los metadatos de *Android*.

3.2.3.1 El modelo de permisos de *Android*

Todas las aplicaciones que se ejecutan en *Android* lo hacen en un entorno de ejecución seguro. De forma predeterminada, una aplicación de *Android* sólo puede acceder a una gama limitada de los recursos del sistema. Éste gestiona el acceso de las aplicaciones de *Android* a los recursos que, si se utilizan de forma incorrecta o maliciosamente, podrían afectar negativamente a la experiencia del usuario, la conexión a *Internet*, o los datos en el dispositivo.

Estas restricciones se aplican de diferentes maneras posibles. Algunas capacidades están restringidas de forma intencionada no haciendo mención en las *APIs* de funcionalidad (por ejemplo, no hay ninguna *API Android* para manipular directamente la tarjeta *SIM*).

En algunos casos, la separación de funciones proporciona una medida de seguridad, como ocurre con el aislamiento de cada aplicación de almacenamiento. En otros casos, las *APIs* están destinadas al uso de aplicaciones de confianza y están protegidas a través de un mecanismo de seguridad conocido como permisos.

Estas *APIs* protegidas incluyen: i) funciones de la cámara, ii) los datos de localización (*GPS*), iii) funciones *Bluetooth*, iv) funciones de telefonía, v) funciones de mensajería y mensajes multimedia y vi) Conexiones de red/datos.

Estos recursos sólo son accesibles a través del sistema operativo. Para hacer uso de las *APIs* protegidas en el dispositivo, una aplicación debe definir las capacida-

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

des que necesita en su *Manifest.xml*¹. A la hora de la instalación de una aplicación, el sistema muestra al usuario un cuadro de diálogo que indica los permisos solicitados, y pregunta si desea continuar la instalación. Si el usuario continúa con la instalación, el sistema procesa que el usuario ha aceptado todos los permisos solicitados. El usuario no puede conceder o denegar permisos individuales, debiendo conceder o denegar todos los permisos solicitados conjuntamente.

Dentro del propio dispositivo *Android* se encuentran los sensores de entrada que interactúan con el medio. Entre estos sistemas están la cámara, el micrófono o el *GPS*.

Para que una aplicación de terceros pueda acceder a estos sensores, tiene que haber solicitado específicamente el uso de los mismos a través del sistema de permisos. En el caso de que una aplicación de terceros quiera acceder a la ubicación de un usuario, habrá tenido que solicitar previamente el acceso a dicha ubicación mediante el permiso correspondiente. Para ello y siempre, antes de la instalación, el sistema le pedirá al usuario si la aplicación puede acceder a su ubicación. En cualquier momento, si el usuario no quiere que las aplicaciones puedan acceder a su ubicación, puede bloquear la ubicación mediante el apagado del sensor *GPS* del terminal. Esta acción desactivará los servicios basados en la localización para todas las aplicaciones en el dispositivo del usuario.

Una vez concedidos, los permisos se atribuyen a la aplicación mientras se instala. Estos permisos pedidos por la aplicación se eliminan si se desinstala esta, por lo que en una nueva instalación se volverá a ver la pantalla de permisos solicitados por la misma.

Dentro de los ajustes del terminal, un usuario puede volver a comprobar los permisos llevados a cabo por una aplicación instalada previamente. A su vez, los usuarios pueden desactivar de forma permanente funciones usadas por aplicaciones como pueden ser el *GPS*, el *Wi-Fi*, etc.

En el caso de que una aplicación intente utilizar una función protegida que no ha sido declarada en el *Manifest.xml* de la aplicación, se ejecuta una excepción de seguridad. En la Fig 3.4 se puede observar un ejemplo de los mensajes llevados a cabo por el sistema para pedir los permisos de aplicación antes de la instalación.

¹El archivo de *Manifest.xml* contiene información relativa a la aplicación, la cual debe comprobar el sistema para poder ejecutar el código de dicha aplicación

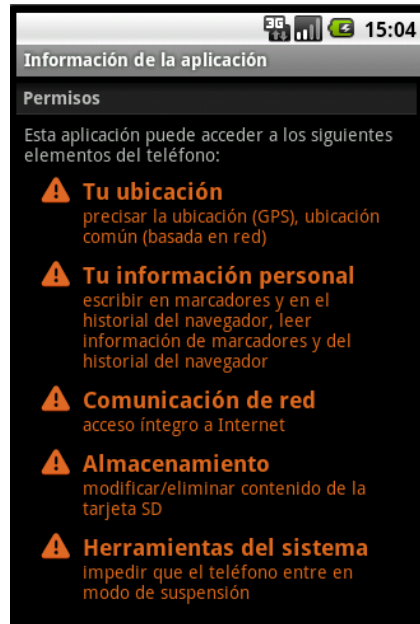


Figura 3.4: Permisos de aplicación pedidos por una aplicación en tiempo de instalación.

El sistema *Android* se esfuerza en mostrar cuando se está interactuando con aplicaciones de terceros y qué capacidades tienen dichas aplicaciones de cara al usuario. Antes de la instalación de cualquier aplicación, se muestra al usuario un mensaje claro acerca de los diferentes permisos que la aplicación está solicitando. Después de la instalación, no se vuelven a solicitar los permisos al usuario, ni se le vuelven a mostrar a menos que el usuario lo haga desde los ajustes del sistema.

Hay muchas razones para mostrar los permisos inmediatamente antes al momento de la instalación. Este momento es cuando el usuario está revisando la información perteneciente a la aplicación, al desarrollador, y a su funcionalidad para determinar si coincide con sus necesidades y expectativas. Por otra parte, hay que tener en cuenta que en ese momento, el usuario todavía no ha establecido un compromiso con la aplicación.

Otras plataformas, como por ejemplo *iOS*, hacen uso de un enfoque diferente para notificar al usuario, como por ejemplo la solicitud de los permisos al inicio de sesión o mientras se ejecuta una aplicación [Ser10]. Por contra, *Android*, únicamente muestra los permisos al usuario en tiempo de instalación. El hecho de

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

proporcionar confirmaciones cada vez, incomodaría al usuario y provocaría que *Android* no ofreciera una gran experiencia de usuario. En cambio, disponer de los permisos para revisarlos durante la instalación le da al usuario la opción de no instalar la aplicación si no se siente cómodo.

Otra de las teorías llevadas a cabo para hacer uso de este sistema, es la relación entre el exceso de peticiones al usuario por parte del sistema y su predisposición a decir *OK* en la mayoría de los diálogos que aparezcan. Uno de los objetivos de la seguridad de *Android* es transmitir eficazmente dicha información segura al usuario, puesto que si a un usuario se acostumbra a hacer una acción, termina ignorando estos cuadros de diálogo y lo que de verdad interesa de ellos. Con la presentación de la información importante de una vez, y sólo cuando es necesario, el usuario tiene más probabilidades de pensar en lo que está a punto de aceptar.

3.2.3.2 La información personal

El equipo de desarrollo de *Android* ha creado una serie de funciones y métodos contenidos en las *APIs* protegidas para poder acceder a los datos de usuario [HHJ⁺11, EOM09b]. Las aplicaciones de terceros también suelen almacenar este tipo de información. Si una aplicación ha decidido compartir esta clase de información, puede hacerlo usando el sistema de permisos, consiguiendo a su vez que haya una mayor comprobación en las aplicaciones de terceros.

Durante la instalación, una aplicación de terceros puede solicitar permiso para acceder a los recursos personales, si se otorga el permiso, la aplicación se instalará y tendrá acceso a los datos que se soliciten.

Cualquier aplicación que recoja esta información personal tiene por defecto restringido su uso: solamente podrá emplearla en una única ocasión. A su vez, una aplicación puede elegir compartir información con las aplicaciones de terceros, siendo únicamente posible mediante el uso de la comunicación entre procesos o *IPC* (*Inter-Process Communication*).

3.2.3.3 Los metadatos *Android*

El sistema *Android* está pensado para restringir el acceso a los datos que sean sensibles, pero mostrando a su vez ciertas características del usuario como sus prefe-

rencias o la manera en la que está usando dicho dispositivo.

En un primer momento, toda la información que puede estar proporcionando el usuario, como su historial de navegación, su número de móvil o la dirección *MAC* del sistema, no es accesible por otras aplicaciones. En el caso de que una aplicación quisiera acceder a esta información, debería pedir acceso mediante el permiso correspondiente y posteriormente ser aceptado por parte del usuario [CFGW11].

3.3 Vectores de infección

En un primer momento, vamos a realizar una clasificación de los vectores de ataque que pueden ser perpetrados contra un dispositivo móvil. Estos vectores de ataque son los que se han tenido en cuenta para futuros análisis en el marco de este documento.

Las amenazas a los dispositivos móviles los podemos dividir en las siguientes cuatro clases: i) centrados en el *hardware*, ii) independientes del dispositivo, iii) centrados en el *software*, o iv) ataques a la capa de usuario [Bec10].

- *Ataques centrado en el hardware*: pertenecen a la seguridad del dispositivo móvil sólo desde un punto de vista más amplio. A pesar de que son adecuados para saltarse las propiedades de seguridad (por ejemplo, la confidencialidad de los datos personales analizados por el análisis forense), no son adecuados para ser usados por un atacante, ya que este tipo de vulnerabilidades por norma general son imposibles de explotar de forma remota, únicamente disponiendo del terminal físico en las manos.
- *Ataques independientes del dispositivo*: cuando hablamos de ataques independientes al dispositivo, normalmente hablamos de los ataques perpetrados contra el medio de comunicación del dispositivo o contra las medidas de seguridad del usuario. Por ejemplo, lo normal en estos casos es el espionaje directo a la conexión inalámbrica del dispositivo para intentar captar información que no vaya cifrada por el medio.
- *Ataques centrados en el software*: en este contexto pertenecen todos los ataques que hacen uso de una vulnerabilidad de una aplicación para poder llevar a cabo su ataque.

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

- *Ataques centrados en el usuario*: cuando nos referimos a este tipo de ataques, normalmente son ataques perpetrados contra el usuario. Esto quiere decir que hacen normalmente uso de técnicas de engaño para llevar al usuario a realizar el mismo los ataques.

Desde el punto de vista de la seguridad, cada uno de los vectores esta separado de los demás y tiene sus propios mecanismos de seguridad y de ataque. En las siguientes subsecciones haremos un resumen de las partes más importantes de cada uno de ellos.

3.3.1 Ataques centrado en el *hardware*

Dentro de este tipo de ataques, nos podemos encontrar dos vectores típicos en cuanto a la seguridad del *hardware* [Bec10]: i) el ataque contra la tarjeta *SIM* o ii) directamente contra el propio dispositivo.

Ataque orientado contra la tarjeta *SIM* del dispositivo

Cuando se creó la interfaz que gestionaba la comunicación entre un dispositivo móvil y las tarjetas *SIM* del operador móvil, se dedujo que los ataques de «hombre en medio» o *MITM* (de sus siglas en inglés *Man In The Middle*) eran totalmente improbables por no decir imposibles. Esto hizo que la comunicación entre ambas partes no fuera cifrada.

Teniendo en cuenta este paradigma, no es extraño que surgieran herramientas como *TurboSIM*[BLA04], la cual implementaba un ataque de *MITM* contra la tarjeta *SIM*. Haciendo uso de esta tecnología, fue posible eliminar correctamente el bloqueo de las tarjetas *SIM* en el iPhone[Ber07].

Ataques orientados contra el dispositivo

En los ataques orientados al dispositivo nos podemos encontrar con dos momentos, que el propio dispositivo este encendido, o que el propio dispositivo esté apagado. Para los dos casos disponemos de diferentes ataques contra los mismos.

- *Ataques contra el sistema de pruebas del dispositivo.* Normalmente, la industria tecnológica suele hacer uso de las técnicas de prueba para comprobar si dicho dispositivo pasa todos los requisitos del mercado o no. Una vez pasadas estas pruebas, el dispositivo se pondrá a la venta, habiendo cerrado el acceso a este sistema. Pero no siempre es así y dicho sistema queda abierto, pudiendo un atacante hacer uso de dicho sistema para acceder al dispositivo. Normalmente este vector se ha venido usando por la industria[Lim09].
- *Análisis forense.* Cuando hablamos de un ataque de análisis forense, estamos haciendo uso de las técnicas llevadas a cabo para poder analizar el estado de un terminal sin la necesidad de tenerlo encendido. Esta técnica normalmente, tiene dos posibilidades, una que el atacante disponga del dispositivo, o dos, siendo la más común, que un terminal cambie de propietario. Si tenemos en cuenta la segunda opción, lo más normal es que nos encontremos con que el dispositivo tenga almacenados información personal o incluso corporativa[FHM08].

3.3.2 Ataques independientes al dispositivo

Dentro de esta parte se incluyen los ataques orientados al medio, tanto a la conectividad como a los sistemas de transmisión de información. En este lado podemos ver los ataques dirigidos a [Bec10]: i) ataques al sistema de mensajería y ii) orientados a las mensajería multimedia.

Ataques mediante la mensajería SMS

Debido a las características heredadas de un dispositivo móvil, los *smartphones* siguen disponiendo del servicio de mensajería tan popular en los últimos años. Este sistema de mensajería o *SMS* proporciona una fuente económica adicional para las compañías de telefonía.

Enck et ál. evaluaron la seguridad de los *SMS* dentro de las disponibilidades de las redes de telefonía móvil [ETMLP05], demostrando la posibilidad de denegar el servicio de voz en una ciudad mediante el colapso del sistema a través del envío masivo de *SMS*.

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

Traynor et ál. hicieron una nueva aproximación a lo investigado por Enck, haciendo uso de unas condiciones de red más realistas [TEMLP09]. Mediante este enfoque pudieron demostrar que los atacantes eran capaces de lograr tasas de saturación de la red superiores al 70 %.

Ataques mediante la mensajería multimedia *MMS*

El servicio *MMS* es un servicio capaz de enviar mensajes de texto de mayor tamaño, incluyendo formatos archivos multimedia desde un teléfono a otro. Para ello hace uso de la infraestructura *GPRS* y los protocolos *WAP*, *SMTP* y *HTTP* como protocolos de transmisión.

Racic et ál. desarrollaron una prueba de concepto mediante la cual explotaban una serie de vulnerabilidades *MMS* para conseguir agotar la batería del teléfono móvil [RMC06]. Dentro de este escenario, lo primero que tuvieron que hacer fue construir un listado de objetivos mediante el envío de mensajes *MMS* desde un falso servidor. Dichos *MMS* dirigía a las víctimas a un servidor web malicioso, el cual al conectarse con la víctima obtenía la dirección IP de la misma. Una vez que conoce la dirección IP de la víctima, el atacante podría enviar periódicamente paquetes *UDP* hasta colapsarlo.

3.3.3 Ataques centrados en el *software*

Las vulnerabilidades centradas en el *software* son más importantes de todas las vulnerabilidades en los dispositivos móviles. El *software*, el cual está compuesto por miles o incluso millones de líneas de código, tiene errores. Como consecuencia de esto también los sistemas operativos contienen fallos de programación. Antigualmente, dichos sistemas operativos eran de código fuente propietario, lo que implicaba que no se podía hacer un análisis del mismo código. En el año 2004 apareció Cabir[FS04], uno de los primeros gusanos para dispositivos móviles.

Desde el principio, los *smartphones* han sido comprometidos por el *malware*, causando la paralización o la interrupción de los servicios de usuario, como por ejemplo, i) dañar el sistema, ii) pérdida monetaria o iii) fuga de información [PYY14].

Dañar el sistema

Una de las posibles consecuencias de la infección de un *smartphone* es el posible daño ocasionado en el sistema, ya sea tanto *hardware* como *software*. Por un lado, los ataques contra el *hardware* nos puede suponer un deterioro de la batería. Por contra, los ataques contra el *software* nos puede implicar un cambio de funciones del sistema, como por ejemplo, una modificación de las configuraciones.

- *Descarga de la batería.* Algunas muestras de *malware* llegan a comprometer su objetivo mediante técnicas de búsqueda continua de información, infectando otros sistemas (como por ejemplo Cabir [FS04], Lasco [Emm06], y Mabir [Chi05]), o enviando continuamente mensajes SMS o MMS (como por ejemplo RedBrowser [McM06]). Como resultado de esto, los dispositivos anfitriones pierden rápidamente la batería.
- *Desactivación de funciones del sistema.* También, el *malware* puede hacer que el sistema no funcione con normalidad. Algún *malware* puede incluso bloquear la funcionalidad de las llamadas, como por ejemplo *Locknut* [Lab05].
- *Cambio de la configuración del sistema.* Por ejemplo, cambiar el fondo de pantalla por otro como hacía *Ikee* [PSY10].

Pérdida monetaria

Otro de los factores que más puede afectar al usuario es la pérdida monetaria que puede suponer el estar infectado por un *malware*.

- *El envío de mensajes SMS o MMS a números de tarificación adicional.* Un ataque ejecutado con éxito puede forzar el teléfono inteligente comprometido a enviar mensajes *SMS* o *MMS* a números de tarificación adicional, como Mquito, que pueden causar una pérdida financiera para el propietario del *smartphone* [Lea05].
- *Marcación de números de tarificación adicional.* Un ataque ejecutado con éxito puede forzar el teléfono inteligente a marcar números premium, como

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

BaseBridge, que pueden causar una pérdida financiera para el usuario del *smartphone* [Lea05].

- *Eliminación de datos importantes.* Los datos almacenados en la memoria del dispositivo o en una tarjeta SD, como por ejemplo, documentos, fotos o videos, pueden verse comprometidos y luego ser eliminados por los atacantes. Además, dicha información puede ser cifrada por el atacante, el cual enviará la clave para descifrarla previo pago de una cantidad de dinero [Abr13].

Fuga de información

Los *smartphones* contienen mucha información importante para el usuario. White [Whi10] hizo un listado de los usos más cotidianos para estos dispositivos y se podía apreciar que en ellos se almacena información confidencial incluso de empresas. Por esta premisa es importante analizar la posible fuga de información de los mismos.

- *Ataque a la privacidad del usuario.* Un ataque ejecutado con éxito, puede hacer que se nos robe los propios SMS y MMS que tenemos en el terminal. Además de estos, también nos pueden extraer los correos electrónicos, registros de llamadas y datos de contacto almacenados en el mismo [BOB⁺10].
- *Vigilancia remota.* Dadas las capacidades que tienen este tipo de dispositivos hoy en día, un atacante puede convertir un teléfono inteligente infectado en un dispositivo de escucha mediante la utilización de la grabación de voz. Además se puede acceder a la geoposición del terminal para tener una localización exacta del usuario [BOB⁺10].
- *Robo de credenciales bancarias.* La banca *online* es uno de los objetivos principales de los atacantes. Para atacar este tipo de sistemas mediante los dispositivos móviles se hace uso de troyanos para el robo de contraseñas como puede ser el caso de *Zeus/MitMo* [Mit10].

3.3.4 Ataques centrados en el usuario

A lo largo de los años ha habido diversos estudios para evaluar los conocimientos en materia de seguridad del usuario promedio. Una de las conclusiones que se llega es la que sacaron Whitten y Tygar en su estudio [WT99]: «los usuarios promedios no son capaces de usar correctamente los mecanismos de seguridad».

Con el objetivo de conseguir que el usuario final entienda la seguridad de los sistemas, ha habido varios intentos de exponerla de una forma sencilla. Desde que Whitten y Tygar [WT99] mostraran sus conclusiones, diferentes estudios se hicieron eco de ello y siguieron analizando dichos factores [CG05, Gar05]. Además, los estudios seguían demostrando que los mecanismos de seguridad nunca han sido entendidos ni bien usados por parte del usuario [FJK06, Fur05, Fur07, WT99].

Por otro lado, hay investigaciones en las que se han propuesto la incursión de la seguridad directamente en el producto, sin la necesidad de ser gestionada por el propio usuario [Kup05]. Por consiguiente, una mala decisión de seguridad en un sistema, puede acarrear una serie de consecuencias bastante drásticas.

3.4 Técnicas de detección de *malware*

Cuando hablamos de técnicas de detección de *malware*, nos estamos refiriendo a las investigaciones llevadas para una correcta clasificación de aplicaciones maliciosas dentro de un sistema. A lo largo de los años ha habido numerosas investigaciones en este ámbito, las cuales se pueden dividir en dos grandes grupos: (i) técnicas de análisis estático y (ii) técnicas de análisis dinámico.

3.4.1 Análisis estático

Muchos investigadores hacen uso de las técnicas de análisis estático para analizar las muestras sin necesidad de ejecutarlas. Esto puede ser llevado a cabo extrayendo los permisos del fichero *Manifest* o bien analizando la información obtenida del paso entre Intents, en la comunicación entre componentes, o las llamadas a la API.

Hay enfoques que utilizan funciones vectoriales y técnicas basadas en la firma para la detección de *malware* [VH08, CWYL07, SM97, MHH06, BHSP08, SPL⁺09]. Algunas soluciones propuestas se detallan a continuación.

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

- Moser et ál. [MKK07] presentaron un esquema binario de ofuscación que se basaba en la idea de las *constantes opacas* que son primitivas para cargar una constante en un registro. A partir de ese registro, una herramienta de análisis no puede determinar su valor de modo que se consigue i) ocultar el flujo de control del programa, ii) ocultar el acceso a las variables locales y globales y iii) interrumpir el seguimiento de los valores almacenados en los registros del procesador. Usando su propuesta de método de ofuscación, Moser et ál. demostraron que los detectores avanzados de *malware* basados en la semántica podían ser evadidos y que las técnicas de análisis estático no son suficientes por sí mismas para identificar el *software* malicioso en general, necesitando ser complementadas con análisis dinámico.
- Venugopal et ál. [VH08] describieron algunas de las principales consideraciones para la detección de *malware* en dispositivos móviles. Se propone un método de detección basada en firmas que se adapta bien para su uso en el dispositivo móvil debido a unos bajos requisitos de memoria y a una alta velocidad de escaneo lo que facilita su portabilidad a toda clase de dispositivos.
- Egele et ál. [ESKK06] describen un análisis estático para aplicaciones web en PHP. Se basan en la identificación de los mecanismos de control de las solicitudes teniendo en cuenta los parámetros de la llamada para crear modelos de detección lo más precisos posibles.
- Christodorescu et ál. [CJ06] utilizaron el análisis estático para la creación de autómatas basados en el propio lenguaje ensamblador para intentar la detección de actividad maliciosa. En particular, el objetivo era abordar el problema de la ofuscación que aún entonces no había sido suficientemente atajado por las aplicaciones comerciales *anti-malware*, las cuales no funcionaban correctamente para hacer frente a estas amenazas. La herramienta desarrollada por Christodorescu et ál. pasó a llamarse analizador estático de ficheros ejecutables (o SAFE, por sus siglas en inglés) y se trata de un enfoque muy apropiado para el manejo de programas maliciosos a través de un sistema estacionario.

- Bergeron et ál. [BDD⁺01] realizaron un análisis semántico del código binario. Su enfoque se divide en tres etapas: i) la creación de una representación intermedia, ii) el análisis del flujo basado en el comportamiento y iii) la verificación estática de los comportamientos críticos contra las propias políticas de seguridad.

El análisis basado en el flujo de información es una técnica valiosa para la investigación de *malware*, pero también tiene que hacer frente a las limitaciones técnicas de los dispositivos móviles.

- Krugel et ál. [KRV04] utilizaron el análisis estático de los binarios a fin de detectar *rootkits* a nivel de *kernel* a través de secuencias de instrucciones antes de los módulos correspondientes se cargan en el núcleo. Afirman que su prototipo no produjo ningún falso positivo, mientras que la detección de todos los *rootkits* probados. Adicionalmente, los autores se refieren a un problema que necesita ser seguido causado por lo que denominan «explosión exponencial de trayectorias posibles» lo que dispara la complejidad del método propuesto. Estos caminos son creados mediante la creación de estados de la máquina observada para el análisis de flujos de control.
- Provos [Pro03] escribió una herramienta capaz de generar y aplicar políticas relacionadas con las llamadas al sistema. Así, la herramienta *Systrace* estaba diseñada para ser eficiente y no imponer sanciones significativas de rendimiento, lo que la ha llevado a ser usada en la propia papelera de reciclaje de ficheros en los sistemas *Linux/Unix*.
- Warrender et ál. [WFP99] compararon las secuencias de llamadas al sistema con el fin de distinguir el comportamiento normal del anormal, poniendo a prueba cuatro métodos con una complejidad cada vez mayor como por ejemplo, los Modelos Ocultos de Markov (*Hidden Markov Models* por sus siglas en inglés). Demostraron que aunque *HMM* obtenía mejor precisión, otros métodos menos complejos eran suficientes. Por contra, encontraron la problemática de que el análisis de las secuencias de llamadas sigue siendo una tarea compleja para implementarlo en los teléfonos inteligentes.

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

- Mutz et ál. [MVVK06] afirman que el sistema común de los enfoques de llamadas basados en no considerar los argumentos o parámetros de la misma permite a los atacantes crear métodos para evadir la detección. Ellos proponen dos mejoras principales sobre el sistema existente:
 - La mejora *RST*¹ aplica varios modelos de detección a los argumentos de llamadas del sistema. Esto les permite analizar los argumentos de las llamadas al sistema desde varias perspectivas.
 - La segunda mejora se describe como un sofisticado método para la agregación de los resultados de todos los modelos de detección aplicados. Este método, se basa en las «redes bayesianas» para la clasificación, mejorando la precisión de la detección y la resistencia contra los intentos de evasión.
- Liu et ál. [LBV05] propone un autómata con un número finito de estados relativos a la información de las llamadas almacenada en la pila para, de forma efectiva, capturar el flujo de control de los programas. Los autores usan el análisis estático para la creación de un modelo base y agregar secuencias de llamadas para un aprendizaje dinámico. Los resultados de la detección muestran que este enfoque tiene mayor capacidad de detección que los enfoques estáticos tradicionales. Este enfoque hace que un dispositivo ligero sea capaz de implementar un método de análisis de llamadas con relativa fluidez.
- Zhang y Reeves [ZR07] proponían un análisis estático para establecer una medida de similitud entre dos ejecutables con el fin de identificar el *malware* metamórfico. Este tipo de *malware* es aquél capaz de evolucionar y de adaptarse a nuevos entornos. En esta línea, Kruegel et ál. describieron el ensamblado estático [KRVV04] mientras que Wang, Wuy Hsieh [WWH08] describieron los métodos de minería de datos para discriminar entre los ejecutables benignos y los maliciosos. En este enfoque extraían las bibliotecas de enlace dinámico (DLL o *dynamic-link libraries* en inglés) y referencias a las interfaces de programación de forma estática y hacían uso de las «máquinas de soporte vectorial» o *SVM* (*Support Vector Machines* de sus siglas en

¹El código RST es un código usado para describir la calidad de las transmisiones de radio

inglés) para la extracción de características y clasificación. En esta línea, Eskin et ál. [SEZS01] también aplicaron los métodos de aprendizaje automático en un conjunto de datos de ejecutables maliciosos.

- Zhang et ál. [ZLH03] mencionaron que la detección de intrusiones en el entorno de los dispositivos móvil puede tener enfoques distribuidos y cooperativos. En este sentido, propusieron el uso de modelos de detección de anomalías contruidos utilizando la información disponible a través de los protocolos de enrutamiento.
- Zhou et ál. [ZWZJ12] propusieron un sistema mediante el cual podían detectar código malicioso. Para ello, comprobaban la carga dinámica de código no fiable, por ejemplo, mediante el uso de la clase de carga de ficheros *Dex* proporcionada por la *API* de *Android*.
- Fuchs et ál. [FCF09] llevaron a cabo una investigación detectando los flujos de información entre componentes del sistema. Para ello, se basaron en las *URI*¹ (*Uniform Resource Identifier* de sus siglas en inglés) encontradas en el sistema. También para ello hicieron uso de los nombre de los *Intents* que se definían en el *Manifest.xml* para poder conseguir el flujo de datos entre diferentes *Intents*.
- Otro de los análisis llevado por Chin et ál. [CFGW11] hacía uso de técnicas de análisis estático para analizar los ficheros *Dalvik*, realizando un estudio de los flujos sensibles entre los procesos, los permisos pedidos por las aplicaciones, los *Intents* usados por la aplicación, así como los componentes recibidos por dichos *Intents*.

Todos estos enfoques tienen el objetivo de intentar hacer frente a algunas de las preocupaciones especiales de seguridad que surgen en la tecnología móvil. Además uno de los ámbitos que más investigaciones ha generado en la detección estática de *malware* en dispositivos móviles ha sido mediante el uso de permisos de aplicación.

¹Un identificador de recursos uniforme es una cadena de caracteres que identifica los recursos de una red de forma unívoca

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

3.4.1.1 Análisis de los permisos de aplicación

Como se ha comentado en el capítulo anterior, los permisos en *Android* son, en gran parte, los encargados de controlar la seguridad dentro del sistema. Un usuario debe aceptar los permisos que pide una aplicación para poder instalarla en el dispositivo. A lo largo de los años ha habido varios enfoques orientados al uso de los permisos para la detección de *malware*.

- Wook et ál [SKFT09] propusieron un análisis formal del modelo de seguridad basado en permisos basándose en el estado del terminal y verificando que el sistema operativo satisface los protocolos de seguridad.
- Por otro lado Rassameeroj et ál. [RT11] crearon una matriz de adyacencia de permisos. En ella calculaban los pesos de los permisos dependiendo de la concurrencia de dicho permiso en las aplicaciones. Por otro lado, crearon una matriz de adyacencia de aplicaciones basándose en la similitud de las aplicaciones en base a los permisos pedidos por las aplicaciones.
- Tang, Wei et ál. [TJHJ11] definió el concepto de distancia de seguridad basado en los permisos utilizados por la aplicación. Dentro de este enfoque, asignaban a cada grupo de permisos un grado de peligrosidad dentro del sistema.
- Enck et ál. presentaron *Kirin* [EOM09a], una certificación de aplicaciones que se valía de la comparación de los permisos pedidos y la configuración de seguridad de las aplicaciones con las políticas de seguridad a la hora de la instalación.
- Sanz et ál. [SSL⁺13] hicieron una aproximación a la detección de *malware* en *Android* mediante el uso de los permisos de las aplicaciones. En dicho estudio extrajeron los permisos de muestras tanto de *goodware* como de *malware*, para mediante el uso de técnicas de aprendizaje supervisado clasificar dichas muestras.
- Otro artículo presentado por Felt et ál. [FCH⁺11], muestra una aplicación llamada *Stowaway* y compara los permisos pedidos por una aplicación en el

fichero *Manifest.xml*, con los permisos que podemos encontrar en esa misma aplicación en las llamadas a la API. Haciendo uso de esta técnica eran capaces de clasificar aplicaciones maliciosas.

3.4.2 Análisis dinámico

El análisis dinámico se basa en la observación del comportamiento del *malware* y analiza sus propiedades ejecutando las muestras sospechosas en un entorno simulado. Así, en esta sección, recogeremos algunas de las técnicas de análisis más conocidas. Dentro de este enfoque nos podemos encontrar las siguientes investigaciones más relevantes:

- *TaintDroid* fue presentado por Enck et ál. [EGH⁺14]. Dentro de este sistema podían señalar la información obtenida de diferentes fuentes como el *GPS*, la cámara, el micrófono, o la monitorización de las interfaces de red, comprobando así posibles fugas de información en el sistema.
- El equipo formado por Yan et ál. [YY12] crearon *DroidScope*, una máquina virtual de *Android* para la inspección profunda de la información en las tres capas de un dispositivo *Android* (el *hardware*, el *SO* y las instrucciones *Dalvik*).
- Por otro lado, en la línea del equipo de *DroidScope* nos encontramos con *AASandbox*, herramienta creada por Thomas et ál. [BBS⁺10]. Este sistema ejecuta las aplicaciones en un entorno aislado de *sandbox* para analizar a bajo nivel las interacciones de la aplicación con el sistema.
- Zhao et ál. [ZGZY11] propusieron *AntiMalDroid*, un sistema que haciendo uso del comportamiento registrado como una característica, construyen los modelos para detectar *malware* y sus variantes, en tiempo de ejecución.

3.4.2.1 Deteccion de anomalías

Varios enfoques prometedores se han presentado para la detección de anomalías en los sistemas fijos y móviles:

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

- Forrest et ál. [FHSL96] propusieron un método para la detección de anomalías. Sus experimentos mostraban que las secuencias cortas de llamadas del sistema en el funcionamiento de los procesos generaban una firma estable para un comportamiento normal.
- Davis et ál. [JDI04] propusieron un método basado en el motor de detección de intrusos (*HIDE*). Dicho enfoque tenía el objetivo de alertar al usuario antes de que el daño sea fuese irreparable. El enfoque estaba pensado para actuar cuando se sospechaba que un ataque está en marcha. Por lo tanto, *HIDE RST* monitorizaba el comportamiento anómalo de la batería cuando ésta ya no lograba entrar en el modo de suspensión. A continuación, y de forma oculta, enviaba un mensaje de alarma para el usuario y el servidor *proxy* más cercano y comenzaba a escribir los registros que contenían las causas del mayor consumo de energía. En el siguiente paso, *HIDE* pasaba a sugerir medidas de mitigación del problema.
- Cheng et ál. [CWYL07] presentan un sistema que utiliza el estado del sistema y un registro de seguimiento, para detectar las infecciones de *malware*.

Así, un cliente de control se instalaba en un dispositivo Windows Mobile 5 de modo que fuera capaz de determinar su propio número de teléfono, la fecha, el identificador del teléfono, los SMS y los registros de llamadas para que, utilizando el análisis estadístico y la anormalidad, se pudieran procesar convenientemente los datos de seguimiento. Adicionalmente, por cuestiones de privacidad y autenticación, se introdujeron procesos de cifrado y de *ticketing*. De esta manera, cada vez que se detectaba una infección, el sistema alertaba al dispositivo correspondiente, así como a todos los dispositivos enlazados con el infectado. La evaluación de las capacidades de este método se produjo a través de una simulación basada en los rastros de SMS procedentes de un proveedor de red telefónica móvil de la India.

- Buennenmeyer et ál. [BNC⁺08] presentaron un enfoque similar al de Davis et ál. [JDI04], supervisando los cambios en un teléfono inteligente con el fin de detectar anomalías. Los cambios pueden ser causados tanto por *malware* como por ataques externos: por ejemplo mediante el escaneo de la red. Los

datos de seguimiento se envían a un servidor remoto que crea perfiles para cada dispositivo de seguimiento de modo que se hace capaz de detectar anomalías en sus representaciones. Los autores evalúan el consumo de energía consumida por los dispositivos a sabiendas de que el cliente utiliza, aproximadamente, menos del 2 % de los recursos de la batería en comparación con la vida útil total de la misma.

- Schmidt et ál. [SPL⁺09] mostraron cómo controlar un *smartphone* con sistema operativo Symbian con el fin de extraer las características que describen el estado del dispositivo para ser utilizadas para la detección de anomalías. Estas características son enviadas a un servidor remoto, porque la ejecución de sistemas de detección de intrusiones es compleja en este tipo de dispositivo móvil y todavía no es factible por culpa de las limitaciones de capacidad y del propio *hardware*. En los artículos citados, se dan ejemplos de cómo calcular algunas de las características e introducen diez aplicaciones utilizadas por los usuarios de teléfonos móviles basados en un estudio realizado en 2005. El uso de estas aplicaciones se registra y visualizan para favorecer una posterior comparación en la que se facilitan los resultados de datos relativos a la vigilancia de las muestras de supuesto *malware* a analizar.

3.4.2.2 Detección de comportamientos maliciosos

Cuando se habla de detección de comportamiento malicioso, se suele basar en base a comportamientos ya conocidos. Normalmente se modelan estos comportamientos para que si una aplicación ejecuta un comportamiento parecido, entonces se clasifica como comportamiento malicioso.

Dentro de este ámbito nos encontramos las siguientes investigaciones más relevantes en el ámbito de la detección para móviles.

- Bose et ál. [BHSP08] propusieron un marco de análisis del comportamiento para detectar *malware* para móviles, en lugar del común ya basado en firmas. Para ello, representaron los comportamientos del *malware* basándose en una observación cuya clave era el seguimiento del orden lógico de las acciones de la aplicación en el tiempo, ya que partían de la base de que a menudo éste

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

revela la intención maliciosa del *software* incluso cuando cada acción aislada por sí sola pueda parecer inofensiva. Además, propusieron una técnica de mapeo de dos etapas que construía la firma de los comportamientos susceptibles de ser maliciosos en tiempo de ejecución, a través de la observación de los eventos del sistema monitorizados y las llamadas a API, mientras que estudiaban 25 familias diferentes de *malware* móvil en el sistema operativo Symbian. Entonces, se pasaba a discriminar el comportamiento malicioso del *malware* del comportamiento normal de las aplicaciones mediante la formación de una clasificación basada en máquinas de soporte vectorial (*Support Vector Machine*). Las tasas de detección obtenidas de muestras de *malware* reales y simulados fueron superiores al 96 %.

- Kang et ál. [KFH05] presentaron un enfoque teniendo en cuenta las secuencias de llamadas al sistema como un problema de clasificación en una bolsa de llamadas al sistema. En esta bolsa, la frecuencia de las llamadas al sistema se almacenan teniendo en consideración la llamada que la ordenó y la llamada que la despidió. Los resultados experimentales en los conjuntos de datos públicos muestran que, frecuentemente, la cantidad de información generada es lo suficientemente eficaz para discriminar entre secuencias normales y secuencias anormales de llamadas al sistema.
- Samfat y Molva [SM97] presenta un sistema de detección de intrusiones distribuidas en redes de teléfonos móviles y *smartphones* que trata de detectar algunos comportamientos abusivos típicos, como el enmascaramiento o el espionaje de red. Usan algoritmos de aprendizaje automático para obtener los perfiles de usuario, que a su vez utilizan las firmas para detectar comportamientos anómalos como por ejemplo la duración exagerada de una llamada o el flujo de información del propio terminal. Cabe destacar en este punto que se centran en la detección de intrusiones y no en la detección del *malware* en sí dentro del dispositivo.
- Lee et ál. [Lea05] presentaron en 2005 un método para utilizar las técnicas de minería de datos con el objetivo de descubrir patrones en las características del sistema que describen la aplicación y el comportamiento del usuario. De

esta manera utilizaron las huellas que dejaban las llamadas al sistema a partir del comando «sendmail», así como los datos de «tcpdump» con el fin de detectar comportamientos sospechosos.

- Hay autores que han presentado métodos para la captura de los comportamientos de flujo de datos en la parte superior de las trazas de llamadas del sistema. En esta línea, Chaturvedi et ál. [CBS05] proporcionaron una definición formal de los comportamientos de los flujo de datos y de los algoritmos actuales para la construcción de dichos modelos. Estos les permiten detectar incluso ataques sofisticados que, utilizando enfoques basados en la detección de los rastros que deja el *malware* en las llamadas al sistema, les permiten detectar variantes que otros modelos no son capaces de detectar. El inconveniente de su enfoque reside en que los datos de entrenamiento necesarios eran de un tamaño de 1,4 a 4 millones de llamadas al sistema para representar cada una de las muestras lo que obligaba forzosamente a que las muestras analizadas tuvieran un tamaño de 100 a 300 MB con los problemas adicionales de sobrecarga que la recogida de estos restos de gran tamaño presenta.
- Miettinen et ál. [MHH06] diseñaron un marco de detección de intrusos que utilizaba la detección de intrusiones basado en *host* y en redes, de tal modo que, si se detectaba una anomalía en un dispositivo móvil, era el propio dispositivo el que enviaba una alerta de intrusión a un servidor *back-end*. Así, este servidor era capaz de recopilar una gran cantidad de información de la red con el fin de crear alarmas relacionadas con la red de intrusos cuando fuera necesario tal y como estaba concebido el motor de correlación diseñado por sus autores.
- Kirda et ál. [KKB⁺06] presentaban en 2006 una técnica de detección de *software* espía que ya superaba entonces algunas de las limitaciones de los actuales enfoques basados en firmas. Su técnica 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 el seguimiento de las acciones de los usuarios y el

3. ESTADO DE LA TÉCNICA EN LA PLATAFORMA *ANDROID*

envío de informes a una entidad externa podía ser identificado como detector de estos comportamientos. La principal ventaja es que sus características lo hacen resistente a la ofuscación, independiente de la imagen binaria que presente y de la firma o firmas obtenidas del fichero, por lo que puede ser utilizado para identificar programas de *software* espía indetectables por otros métodos. Debido a que un componente de *software* espía necesita interactuar con el sistema operativo, se analizan las llamadas a la API de Windows para tratar de determinar qué componentes podrían estar utilizándolas para filtrar información del proceso actual, especialmente aquellas tareas que se realizan en respuesta a determinados eventos.

Por esa razón, Kirda et ál. utilizan el análisis 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 los eventos. A continuación, utilizan el análisis estático para examinar estas regiones en busca de llamadas al sistema relacionadas con la creación de hilos o temporizadores para tratar de identificar aquellas regiones susceptibles de ser invocadas en respuesta a un evento. Por último, se generaba de forma automática una lista negra o *blacklist* de llamadas a la API a partir de datos extraídos mediante el análisis de frecuencia. La aplicación de un sistema similar en dispositivos móviles podría ser potencialmente interesante a sabiendas de que se cuentan con más indicadores relativos a la gestión de los procesos por la propia arquitectura de los sistemas *Android* e iOS.

3.4.2.3 Detección de consumos energéticos

Varias publicaciones se han hecho en el campo de la detección de *malware* en teléfonos inteligentes y los sistemas de detección de intrusos asociados de los que se puede extraer que algunos de los enfoques más prometedores se refieren a datos de consumo de energía con el fin de detectar el hecho de que se están cometiendo una serie de ataques [JDI04, KSS08, MVVK06, BNC⁺08]. A continuación se citan algunos casos de éxito.

- Nash et ál. [NMHH05] proponen el desarrollo de un IDS capaz de enfrentarse a una forma única de ataque DoS conocida como *ataque de agotamiento de la batería*. Este modelo tiene en cuenta el rendimiento, la energía y las limitaciones de memoria de los dispositivos móviles para detectar cuándo el dispositivo está siendo objeto de un ataque de estas características. Así, sus IDS utilizan varios parámetros, tales como la carga de la CPU y los accesos al disco para estimar el consumo de energía por proceso utilizando un modelo de regresión lineal y así poder identificar los procesos que han desencadenado un consumo desproporcionado de la batería.
- Kim et ál. [KSS08] proponen un *framework* de conocimiento para la detección de *malware* que supervisa, detecta y analiza amenazas energéticas previamente desconocidas agotamiento. El modelo incluye, por un lado, una serie de procesos de monitorización de la potencia que construyen una historia de consumo de energía de las muestras recogidas y, por otro, un analizador de datos que genera una firma histórica del gasto de alimentación del dispositivo. Con el fin de reducir las tasas tanto de falsos positivos como de falsos negativos en la detección, las similitudes entre las firmas de energía se evalúan teniendo como referencia las 2 más similares.
- Martin et ál. [MV06] describen tres métodos principales para llevar a cabo la prevención de ataques a la CPU de los dispositivos móviles, los cuales impiden a los teléfonos entrar en modo reposo al reducir la vida útil de la batería. Para contrarrestarlos, se propone una arquitectura de energía segura mediante la autenticación y las firmas de consumo de energía.
- Otro IDS basado en el consumo de las baterías fue presentado por Jacoby et ál. [JMD06]. Básicamente, media el consumo de alimentación del dispositivo para posteriormente correlacionarlo con la actividad de las aplicaciones en el dispositivo por medio de la ejecución de una serie de reglas orientadas a la detección de intrusiones en el propio *host*.

3.5 Sumario

A lo largo del capítulo hemos hecho una presentación de como ha ido evolucionando el *malware* a lo largo de los años. La primera parte se ha tenido en cuenta la presentación de los diferentes vectores de infección que nos encontramos en este sistema.

Seguido hemos explicado la arquitectura del sistema *Android*, dividida en sus diferentes capas. En cada una de las capas hemos hecho un análisis de los cometidos de cada una de ellas.

Continuando, nos hemos centrado en la seguridad de *Android*. Hemos explicado el sistema operativo y como ha sido pensado basándose desde un primer momento en la seguridad del usuario. Más adelante, hemos resumido todos los aspectos a tener en cuenta en el diseño del sistema operativo, tanto desde el punto de vista *software*, como desde el punto de vista *hardware*, demostrando así que para el sistema *Android* es igual de importante los sensores de los que dispone como lo que pueda ocurrir dentro de él.

Dentro de esta explicación hemos visto la importancia de las máquinas virtuales *Dalvik* y su sistema de *sandboxing* para evitar que una aplicación pueda obtener más privilegios de los que debería, o también, un ataque malintencionado entre diferentes aplicaciones.

A continuación, hemos desarrollado toda la parte de la seguridad relacionada con el usuario y sus tres aspectos más importantes en cuanto a seguridad: i) el cifrado de ficheros, ii) el bloqueo del sistema por contraseña y iii) la posibilidad de administración remota del terminal.

Por último, hemos revisado los sistemas de seguridad que incorpora la plataforma a nivel de aplicación. Dentro de estos enfoques, hemos estudiado el sistema de permisos que implementa *Android*. Si bien es cierto que otras plataformas ya lo habían implementado, *Android* ha hecho de dicho sistema uno de sus puntos más fuertes en cuanto a la seguridad se refiere.

Además de explicar todos los aspectos más importantes de los permisos, hemos expuesto todo lo que estos permisos implican en cuanto a la seguridad de la información personal, los posibles sensores de entrada de información (como el *GPS*) o los metadatos de las aplicaciones. Por otro lado, también hemos comprobado que

Google tiene bien elaborado el proceso de identificación de los desarrolladores en caso de que una aplicación esté realizando alguna actividad maliciosa.

También hemos visto los riesgos de los ataques dentro de nuestro sistema, ofreciendo una visión de los daños que puede causar un *malware* para el usuario, tanto en a nivel físico del dispositivo, como a nivel económico para el propio usuario. La capacidad que estas muestras están teniendo para monetizar el daño causado esta siendo una de las máximas a la hora de analizarlas.

Una vez explicadas las partes más generales del *malware*, tanto como actúa y los efectos que puede generar en nuestro sistema, hemos hecho una división de los dos enfoques llevados a cabo en la detección de *malware* a lo largo de los años: el análisis estático y el análisis dinámico de aplicaciones.

Dentro del apartado de análisis estático hemos podido ver todas las investigaciones que se han llevado a cabo durante los últimos años para la correcta clasificación de estas aplicaciones.

En la parte de análisis dinámico de aplicaciones, en un primer momento hemos definido los tres enfoques llevados por la comunidad científica para la detección dinámica. Aunque los dos primeros enfoques se parecen mucho, hay que tener en cuenta que el basado en anomalías y el basado en comportamiento son muy diferentes. En el primero de ellos hay que modelizar antes de nada cual va a ser considerado como bueno, y una vez que se sale de lo modelo quiere decir que es comportamiento malo. En el segundo enfoque lo que hay que modelizar son los comportamientos maliciosos, y clasificando como malicioso todo comportamiento que se asemeje a lo modelado.

Para terminar se ha procedido también a hacer una comprobación de las investigaciones llevadas por la comunidad científica en cuanto al gasto de energía en el dispositivo cuando está infectado por una aplicación maliciosa.

«La mayoría de las ideas fundamentales la ciencia son esencialmente sencillas y, por regla general pueden ser expresadas en un lenguaje comprensible para todos.»

Albert Einstein (1879–1955)

CAPÍTULO

4

Utilización de opcodes como predictor de la detección

LAS capacidades de los *smartphones* han hecho que el incremento de aplicaciones haya sido exponencial. Hoy en día muchos *smartphones* tienen capacidades de cómputo y de memoria mayores que muchos ordenadores de unos pocos años.

Como se hemos comentado en el capítulo 2, estos dispositivos hacen uso de un sistema operativo bastante simple visualmente para que el usuario no tenga problemas a la hora de utilizarlo. En una entrevista a Andy Rubin, exdirector de *Google*, dijo: *There should be nothing that users can access on their desktop that they can't access on their cell phone*. [Wat08]. Esta frase no hace más que demostrar el avance sufrido estos pequeños ordenadores.

En la sección 4.1 haremos un pequeño estado de la técnica de las investigaciones relacionadas a los sistemas *Android* que se han llevado a cabo por la comunidad científica. En la sección 4.2 veremos el alcance de esta experimentación, y como se ha obtenido la información necesaria para realizarla. En la sección 5.4 haremos un resumen de los diferentes clasificadores usados para esta experimentación. En la sección 4.4 expondremos los resultados, y de los parámetros a tener en cuenta con dicha validación experimental. Por último en la sección 4.5 mostraremos las

4. UTILIZACIÓN DE OPCODES COMO PREDICTOR DE LA DETECCIÓN

conclusiones obtenidas por el experimento así como las posibles líneas de trabajo futuro que se llevarán a cabo.

4.1 Trabajo relacionado

El sistema operativo *Android* está pensado para que cada aplicación se ejecute dentro de su propia máquina virtual *Dalvik*. Este tipo de implementación hace que el sistema sea más robusto y limite los daños ocasionados por una mala programación [EOMC11].

Los sistemas *Android*, además, vienen provistos de un sistema de permisos que no permite que aplicaciones de terceros accedan a recursos que no deberían poder acceder. Estos permisos se le asignan a la aplicación en tiempo de instalación, siendo la decisión final del usuario instalar dicha aplicación o no, dependiendo de los permisos que ésta le esté pidiendo. Los desarrolladores de aplicaciones maliciosas pueden pedir más permisos de los que realmente necesita para, por ejemplo, obtener información privada del usuario [FBJS12]. Por esta línea, una de las investigaciones ha sido llevada a cabo por Sanz et al. [SSL⁺13], en la cual mediante los permisos eran capaces de clasificar si una aplicación era maliciosa o benigna según los permisos solicitados por ella. Otros autores han utilizado también esta característica en sus investigaciones [SKFT09, SKFT10]. Sin embargo, *Android* es un sistema que evoluciona muy rápido y, actualmente realiza agrupaciones de permisos. Con las agrupaciones, estas aproximaciones pierden efectividad.

Teniendo en cuenta que según la documentación de *Android*, hoy en día los permisos han cambiado a ser un grupo de permisos, estas aproximaciones ya no tendrían tan buenos resultados puesto que ya no se puede comprobar permisos específicos.

Una aproximación clásica en dispositivos móviles es el analizar el comportamiento de distintos elementos *hardware*, como la *CPU* o la batería, para detectar anomalías en el comportamiento [JDI04, BNC⁺08]. Estas aproximaciones, aunque si es cierto que encuentra comportamientos extraños del sistema, tienen el problema de que hoy en día debido a la complejidad en la programación y el ritmo de actualizaciones por parte de una aplicación, pueden tener fallos de programación

que afecten a estos elementos y no por ello se pueden considerar aplicaciones maliciosas.

A su vez, podemos ver que las primeras y más importantes aproximaciones que ha habido dentro de la detección de *malware* en *Android*, se han focalizado en los permisos de las aplicaciones. Así podemos ver como Sanz et al. [SSL⁺13] hicieron una primera investigación con la cual mediante los permisos pedidos por una aplicación son capaces de clasificar mediante técnicas de *machine learning* si dicha aplicación es benigna o maligna. A parte de ellos, han sido bastantes los investigadores que han centrado sus avances en esta característica .

Mientras, Schmidt et al. [SBS⁺09] desarrollaron un *framework* que usaba las llamadas al sistema como característica para la clasificación de aplicaciones maliciosas o benignas dentro del sistema *Android*. Este tipo de clasificación tiene la problemática de que es muy costoso conseguir dicha información. Además, el rendimiento de esta aproximación en función del número de llamadas, puede ser muy bajo.

Shabtai et al. [SFE10] hicieron una investigación mediante el uso de los ficheros decompilados de una aplicación *Android*. En dicho estudio, usando características extraídas como por ejemplo las clases usadas por la aplicación y haciendo uso de técnicas de aprendizaje automático podían clasificar las aplicaciones maliciosas. Para esta aproximación hace falta una cantidad muy grande de características extraídas de las aplicaciones para tener una buena clasificación.

Ya existe una aproximación mediante el uso de los ficheros decompilados de una aplicación *Android* llevada a cabo por Shabtai et al. [SFE10]. En dicho estudio, usando características extraídas como por ejemplo las clases usadas por la aplicación y haciendo uso de técnicas de *machine learning* podían clasificar las aplicaciones maliciosas.

Una de las técnicas usadas en la creación de *malware* para *Android* es usar una aplicación legítima e incluirle un código malicioso dentro de ella. Siguiendo esta línea, Zhou et al. [ZZJN12] hicieron una investigación en tiendas de aplicaciones de terceros. Una de las mayores limitaciones de esta aproximación es la necesidad de tener la aplicación legítima para comprobar si se ha introducido código malicioso en ella.

4. UTILIZACIÓN DE OPCODES COMO PREDICTOR DE LA DETECCIÓN

La seguridad en los sistemas *Android* ha estado en el punto de mira desde que saliera a la venta el primer *smartphone* con este sistema operativo. Este sistema ha ido creciendo hasta conseguir una cuota de mercado del 75 % en el primer cuarto del 2013 [IDC13].

Los sistemas *Android* vienen protegidos por un sistema de permisos que no permite que aplicaciones de terceros accedan a recursos que no deberían poder acceder. Estos permisos se le asignan a la aplicación en tiempo de instalación, siendo la decisión final del usuario instalar dicha aplicación o no dependiendo de los permisos que ésta le esté pidiendo. Los desarrolladores de aplicaciones maliciosas pueden pedir más permisos de los que realmente necesita para, por ejemplo, obtener información privada del usuario [FBJS12].

El sistema operativo *Android* está pensado para que cada aplicación se ejecute dentro de su propia máquina virtual *Dalvik*. Este tipo de implementación hace que el sistema sea más robusto y limite los daños ocasionados por una mala programación [EOMC11].

Desde un primer momento han ido haciendo diferentes estudios basados en el uso de la batería para encontrar posibles anomalías [JDI04, BNC⁺08]. A su vez, podemos ver que las primeras y más importantes aproximaciones que ha habido dentro de la detección de *malware* en *Android*, se han focalizado en los permisos de las aplicaciones. Así podemos ver como Sanz et al. [SSL⁺13] hacen una primera investigación con la cual mediante los permisos pedidos por una aplicación son capaces de clasificar mediante técnicas de *machine learning* si dicha aplicación es benigna o maligna. A parte de ellos, han sido bastantes los investigadores que han centrado sus avances en esta característica [SKFT09, SKFT10].

Schmidt et al. [SBS⁺09], también usando técnicas de aprendizaje supervisado, idearon un *framework* que usaba las llamadas al sistema como característica para la clasificación de aplicaciones maliciosas o benignas dentro del sistema *Android*.

Ya existe una aproximación mediante el uso de los ficheros decompilados de una aplicación *Android* llevada a cabo por Shabtai et al. [SFE10]. En dicho estudio, usando características extraídas como por ejemplo las clases usadas por la aplicación y haciendo uso de técnicas de *machine learning* podían clasificar las aplicaciones maliciosas. En el caso de Zhou et al. [ZZJN12] consideraron las ins-

trucciones para detectar aplicaciones recompiladas en tiendas de aplicaciones de terceros.

4.2 Alcance de la experimentación

En este contexto, siguiendo las pautas usadas para la detección de código malicioso en ordenadores de sobremesa llevada por Santos et al.[SBS⁺11], los autores proponen un estudio que consista en: i) la captura de información relativa a los códigos operacionales usados por una aplicación, ii) la clasificación maliciosa o benigna de una aplicación, mediante el uso de técnicas de *machine learning*.

Una vez obtenidas todas las aplicaciones, el objetivo principal de la investigación es la capacidad, mediante técnicas de aprendizaje supervisado, de detectar con el menor número de falsos positivos y falsos negativos, las aplicaciones generadas para un uso malicioso dentro de la plataforma *Android*.

En las subsecciones posteriores se define la metodología empleada para la modelización de las aplicaciones que se van a usar así como sus características que van a ser utilizadas para exponer esta prueba de concepto, que explota las posibilidades de la herramienta de la Universidad de Waikato, *Weka*¹ para emplear diversos algoritmos de clasificación.

4.2.1 Recolección de las muestras

Como se ha comentado en el Capítulo 5 vamos a proceder a la explicación de la obtención de las muestras obtenidas para el experimento.

Puesto que para los enfoques de aprendizaje supervisado hace falta muestras de todas las clases en las que se va a clasificar, hemos procedido a hacer una recolección de muestras de código benigno y código maligno. Para la obtención de las muestras de *goodware*, hemos creado una herramienta de automatización. Mediante la aplicación descargaremos las aplicaciones de la propia tienda de *Android*, la *Google Play Store*.

¹*Weka: Data Mining Software* es una colección de algoritmos de aprendizaje automático para la automatización de tareas de minería de datos: <http://www.cs.waikato.ac.nz/ml/weka/>

4. UTILIZACIÓN DE OPCODES COMO PREDICTOR DE LA DETECCIÓN

Para ello, hemos recuperado todas las categorías de las que se conforma la tienda de aplicaciones, obteniendo un total de cuarenta y tres categorías diferentes. Una vez obtenidas las categorías, hemos procedido a obtener el listado de paquetes pertenecientes a las aplicaciones gratuitas de esas categorías, consiguiendo una media de 164 aplicaciones por cada categoría, con un total de 7.062 aplicaciones. La Tabla 4.1 muestra el número de aplicaciones descargadas por categoría.

Tabla 4.1: Cantidad de aplicaciones por categoría descargadas de la *Play Store* de Google

<i>Categoría</i>	<i>#</i>	<i>Categoría</i>	<i>#</i>
Negocios	179	Juegos Acción	156
Estilo de Vida	168	Juegos Puzzle	170
Compras	163	Juegos Simulación	165
Libros y Referencias	173	Entretenimiento	157
Medicina	176	Juegos Trivial	161
Juegos Estrategia	163	Juegos Rol	159
Juegos Aventuras	161	Noticias y Revistas	170
Juegos Casuales	160	Finanzas	173
Media y Videos	156	Juegos Palabras	170
Juegos Cartas	176	Fondos de Escritorio	163
Bibliotecas y Demos	185	Productividad	178
Tiempo	172	Música y Audio	156
Juegos Educativos	170	Comunicaciones	177
Juegos Mesa	175	Salud y Fitness	172
Educación	179	Comics	61
Juegos Familiares	157	Juegos Arcade	169
Juegos Deportes	168	Juegos Carreras	168
Sociales	163	Viajes y Locales	174
Juegos Casino	175	Fotografía	159
Transporte	174	Herramientas	177
Deportes	161	Personalización	111
Juegos Música	162		
<i>Total: 7.062</i>			

Una vez obtenidos todos los paquetes de aplicaciones, fue la descarga de las mismas. Con este objetivo, usamos la aplicación de automatización de navegadores web *Selenium*². Mediante esta aplicación y el servicio *online APIfy*³, descargamos todas las aplicaciones para su posterior análisis e inclusión en el *dataset* de código benigno.

Para analizar todas las aplicaciones hemos hecho uso del sistema propiedad de *Google: VirusTotal*⁴. Este sistema nos proporciona el resultado de analizar una aplicación con cuarenta y seis motores antivirus diferentes. Puesto que hoy en día no todas las muestras son completamente maliciosas, o completamente benignas, decidimos que en nuestro *dataset* únicamente incluiríamos las muestras que al ser enviadas a *VirusTotal*, nos devolviese como benigno por todos los motores que contiene el sistema. De este primer análisis, un 14 % de las muestras descargadas han sido clasificadas como código malicioso, quedándonos con 6.073 aplicaciones.

Debido a que el conjunto de muestras de código malicioso dispone de 5.560 aplicaciones, hemos tenido que reducir aún más las muestras de código benigno. Con el fin de igualar los dos conjuntos de datos, hemos realizado una selección aleatoria de las diferentes aplicaciones finales del dataset total de aplicaciones, eliminando así un total de 513 aplicaciones.

Para el conjunto de datos de código malicioso, hemos utilizado el proporcionado por la *Universidad de Göttingen* y el centro *Siemens CERT* y publicado con el nombre de *Drebin* [ASH⁺14]. Dicho conjunto de aplicaciones, está formado por un total de 5.560 aplicaciones maliciosas divididas en 179 familias diferentes.

4.2.2 Obtención de la información

En esta fase se ha procedido a crear una plataforma en el lenguaje de programación *C#*. En dicha plataforma se ha implementado un *plugin* para la obtención de los diferentes códigos operacionales que usa una *apk* para ejecutar sus acciones. Estos códigos operacionales se han obtenido mediante el uso de la aplicación *Dedexer* en su versión 1.26. Esta herramienta realiza un desensamblado del fichero *.dex*

²<http://docs.seleniumhq.org/>

³<http://apify.ifc0nfig.com/>

⁴<http://www.virustotal.com>

4. UTILIZACIÓN DE OPCODES COMO PREDICTOR DE LA DETECCIÓN

extraído de la aplicación a la hora de descomprimirla. Una vez desensamblada toda la aplicación se procede a leer de forma secuencial todos los ficheros y extraer los códigos operacionales de las mismas.

Por cada aplicación extraída se ha generado un fichero único con toda la información necesaria de la misma.

4.3 Aprendizaje supervisado

De igual manera a como ya hicimos en el pasado Brezo et ál. para detectar el tráfico procedente de una botnet *HTTP* [BdlPUP⁺13], se ha planteado el problema como un problema de clasificación supervisada. En este tipo de problemas, se estudia un fenómeno representado por un vector X en R^d que puede ser clasificado de K maneras diferentes de acuerdo a un vector Y de etiquetas. El aprendizaje supervisado y la utilización de algoritmos de clasificación hacen uso de un conjunto de datos previamente etiquetado [KZP07] que, en nuestro caso, se corresponde con las etiquetas de tráfico de *goodware* o *malware* en función del origen de las representaciones utilizadas para el experimento.

En las páginas que siguen, revisamos los diferentes algoritmos de aprendizaje supervisado utilizados para afrontar el problema de la detección de tráfico de red como un problema de clasificación supervisada. Se ha optado por comparar el rendimiento de los diferentes algoritmos de clasificación dadas las ocasionalmente notables diferencias de efectividad que se pueden observar en experimentos similares llevados a cabo en otros ámbitos como la detección de errores en modelos de calidad del *software* [SKM09] o la clasificación automática de comentarios en sitios web sociales [SDLPSPL⁺12] entre otros.

4.3.1 Máquinas de soporte vectorial

Los clasificadores basados en máquinas de soporte vectorial o *SVM* (de sus siglas en inglés *Support Vector Machines*) se basan en un hiperplano que divide una representación n -dimensional de los datos en dos regiones separadas. Se define el hiperplano como aquel espacio que maximiza el margen m entre dos clases, también llamadas regiones espaciales. Este margen se define a su vez como la distancia

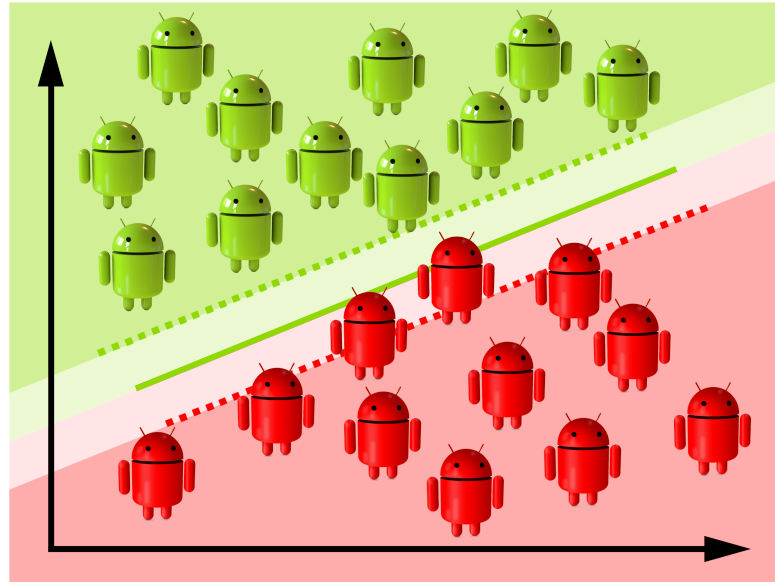


Figura 4.1: Ejemplo de máquina de soporte vectorial.

más larga entre los ejemplos de las clases, calculada a partir de la distancia de los especímenes más cercanos al hiperplano como se puede observar en la Figura 4.1. La idea es que, haciendo uso de un conjunto de datos de entrenamiento, un *SVM* genere un modelo lineal capaz de predecir si un nuevo espécimen desconocido pertenece a una u otra categoría aplicando la citada división del hiperplano. Para ello, las máquinas de soporte vectorial hacen uso de diferentes funciones de *kernel*.

4.3.1.1 Funciones de kernel

Si se utiliza la forma dual¹ de la representación cuadrática para representar al hiperplano, se revela que la tarea de clasificación es únicamente una función de los vectores de soporte [Bis06]. Para ello hemos usado varios tipos de *kernel*: i) el *kernel polinomial*, ii) el *polinomial normalizado*, iii) *Kernel Radial Basis Function (RBF)* y iv) *Pearson VII Universal Kernel (PUK)*.

¹En programación lineal, el problema primario y la forma dual son complementarios. Una solución a cualquiera de ellos determina la solución a ambos.

4. UTILIZACIÓN DE OPCODES COMO PREDICTOR DE LA DETECCIÓN

- Los *kernels polinomiales* [AW99] han sido utilizados con éxito en los últimos años con los *SVM*, especialmente para afrontar los problemas relacionados con el procesamiento del lenguaje natural tanto en su versión normal que podemos ver en la ecuación 4.1:

$$k(x, xy) = (x \cdot y)^d \quad (4.1)$$

como en su versión normalizada que observamos en la ecuación 4.2:

$$K(x, xy) = \frac{(x \cdot y)^d}{\sqrt{(x \cdot y)^d}} \quad (4.2)$$

En ambos casos, d representa el grado del polinomio. Es habitual no encontrarse con valores de d superiores a 2, ya que se ha observado que se presentan problemas de sobreajuste a los datos de muestra [CHC⁺10]. Por situaciones de sobreajuste, hay que entender aquellas situaciones que se dan cuando uno de los clasificadores ofrecen resultados óptimos para el conjunto de datos de entrenamiento utilizados pero este no es capaz de acomodarse a la variedad de datos que otros proveen. Esta característica puede dar lugar a errores en la aplicación de este algoritmo de clasificación en entornos reales [SLTW04].

4.3.2 Árboles de decisión

Los árboles de decisión son un tipo de clasificadores de aprendizaje automático que pueden ser representados gráficamente como árboles como observamos en la Figura 4.2. Los nodos interiores representan las condiciones o estados posibles de las variables del problema y los nodos finales u hojas constituyen la decisión final del clasificador [Qui86].

Un árbol de decisión es, formalmente, un grafo $G = (V, E)$ que consiste en un conjunto no vacío de nodos finitos V y un conjunto de aristas E . Si el conjunto de las aristas E está compuesto a su vez por bi-tuplas ordenadas de nodos de la forma (v, w) , entonces se puede decir que el grafo G es dirigido.

Un camino en el grafo G se define como una secuencia de aristas de la forma $(v_1, v_2), (v_2, v_3), \dots, (v_{n-1}, v_n)$. Los caminos se pueden expresar por el origen, su

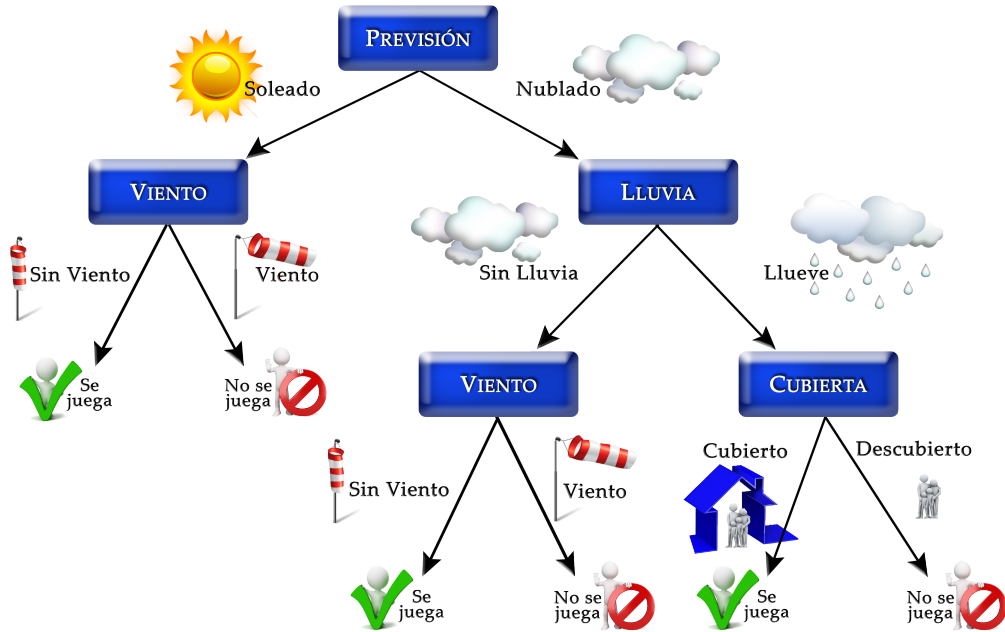


Figura 4.2: Ejemplo de árbol de decisión.

final y la distancia recorrida, entendiendo como tal el mínimo número de aristas desde el origen hasta el final. Además, si (v, w) es una arista en el árbol, entonces se considera a v nodo padre de w , mientras que w sería el nodo hijo de v . El único nodo en el árbol sin padres se denomina nodo raíz. Cualquier otro nodo en el árbol, es un nodo interno. Para construir la representación gráfica de un árbol, se contesta a un conjunto de preguntas binarias.

Hay varios algoritmos de aprendizaje supervisado, que se utilizan para aprender la estructura de un árbol mediante un conjunto de datos etiquetados. En este trabajo, hemos utilizado *Random Forest*, que es una combinación de árboles de decisión contruidos al azar. Además, hemos utilizado *J48*, que es la implementación de *Weka* [G⁺95] del algoritmo *C4.5* [Qui93].

C4.5 crea árboles de decisión dada una cantidad de información de entrenamiento usando el concepto de entropía de la información [Sal94]. En cada nodo del árbol de decisión, el algoritmo elige un atributo de los datos que más eficazmente dividen el conjunto de muestras en subconjuntos enriquecidos en una clase u otra

4. UTILIZACIÓN DE OPCODES COMO PREDICTOR DE LA DETECCIÓN

utilizando como criterio de selección la diferencia de entropía. El algoritmo construye árboles de decisión utilizando un conjunto de datos etiquetados y mediante el concepto de entropía de la información, que mide la incertidumbre asociada con una variable aleatoria. Este concepto, también conocido como entropía de Shannon [Sha51, Sha01], mide el valor esperado de información (*Information Gain*, *IG*) dentro de un mensaje.

Para generar un árbol de decisión mediante el algoritmo *C4.5*, se requiere un conjunto de datos de entrenamiento $S = s_1, s_2, \dots, s_{n-1}, s_n$ previamente etiquetado. Así, los datos de entrenamiento conforman un corpus de ejemplos $s_i = x_1, x_2, \dots, x_m$ ya clasificados en los que $x_1, x_2, \dots, x_{m-1}, x_m$ representan los m atributos o características del ejemplo. El atributo m de cada representación corresponderá con alguna de las o etiquetas $C = c_1, c_2, \dots, c_{o-1}, c_o$ y representarán la clase a la que pertenece dicho espécimen s_i .

Para cada nodo del árbol, el algoritmo *C4.5* elige el atributo dentro de los datos que, de forma más eficiente, divide el conjunto restante de especímenes en subconjuntos de una clase o de la otra. El criterio para dividir el conjunto es la ganancia de información o *Information Gain (IG)*, la cual calcula la dependencia estadística entre dos variables aleatorias. De este modo, el atributo con el mayor *IG* se selecciona para ser el nodo que divida el nuevo conjunto de datos. Este proceso se repite recursivamente hasta que no se puedan realizar más divisiones del conjunto de datos. *C4.5* es capaz de manejar tanto atributos continuos como discretos, datos sin completar, atributos con diferentes costes y, además, puede optimizar los árboles después de generarlos.

El algoritmo *Random Forest* se trata de un clasificador de agregación desarrollado por Leo Breiman [Bre01] y que consta de un abanico de subárboles de decisión elegidos de forma aleatoria. *Random Forest* mejora la precisión en la clasificación ya que en la construcción de cada clasificador individual se introduce un componente estocástico, bien en la partición del espacio, es decir, introduciendo elementos aleatorios en la propia construcción de los árboles haciendo del sistema uno no determinista, o bien, en la muestra de entrenamiento [Bre01].

Para la clasificación de una instancia, se genera entonces un vector de entrada para cada uno de los árboles de la combinación. De esta manera cada árbol genera

una clasificación siendo el clasificador global el que elija aquella que haya obtenido una mayor cantidad de votos.

Como ya se ha comentado, a la hora de generar los árboles de decisión aleatorios que componen a un *Random Forest*, se elige un subconjunto aleatorio del conjunto total de datos. Para ello, el método utiliza una discriminación estocástica [K⁺96].

Seguidamente, se calcula la mejor división posible para ese subconjunto de características. Se repite el proceso para cada nodo del árbol de decisión sin llegar a efectuar una optimización del mismo una vez que el árbol ha sido creado.

Es de esta manera como el algoritmo *Random Forest* produce un clasificador preciso para muchos conjuntos de datos, manteniendo su precisión incluso cuando existe un gran número de variables de entrada. Además, estima la importancia de las mismas a la hora de determinar la clasificación final y es capaz de enfrentarse a datos incompletos. Sin embargo, suelen ser bastante susceptibles a sobreajustarse al conjunto de datos, especialmente en tareas con una gran cantidad de datos ruidosos en donde el proceso de recogida no ha sido realizado de forma correcta.

4.3.3 Redes bayesianas

El teorema de *Bayes* [BP63] es la base para la inferencia bayesiana, un método estadístico de razonamiento que determina, basándose en un número de observaciones, la probabilidad de que una hipótesis dada sea cierta, como por ejemplo en la Figura 4.3. Una de sus principales características es que el teorema de *Bayes* ajusta la probabilidades tan pronto como se dé con nuevas observaciones.

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)} \quad (4.3)$$

De acuerdo con su formulación clásica, dados dos eventos A y B , la probabilidad condicionada $P(A|B)$ de que ocurra A si ha ocurrido B , se puede obtener si sabemos la probabilidad de que ocurra A , $P(A)$, la probabilidad de que ocurra B , $P(B)$, y la probabilidad condicionada de B dado A , $P(B|A)$.

Podemos decir por tanto que las redes bayesianas conforman un modelo probabilístico que representa una colección de variables aleatorias mediante un grafo

4. UTILIZACIÓN DE OPCODES COMO PREDICTOR DE LA DETECCIÓN

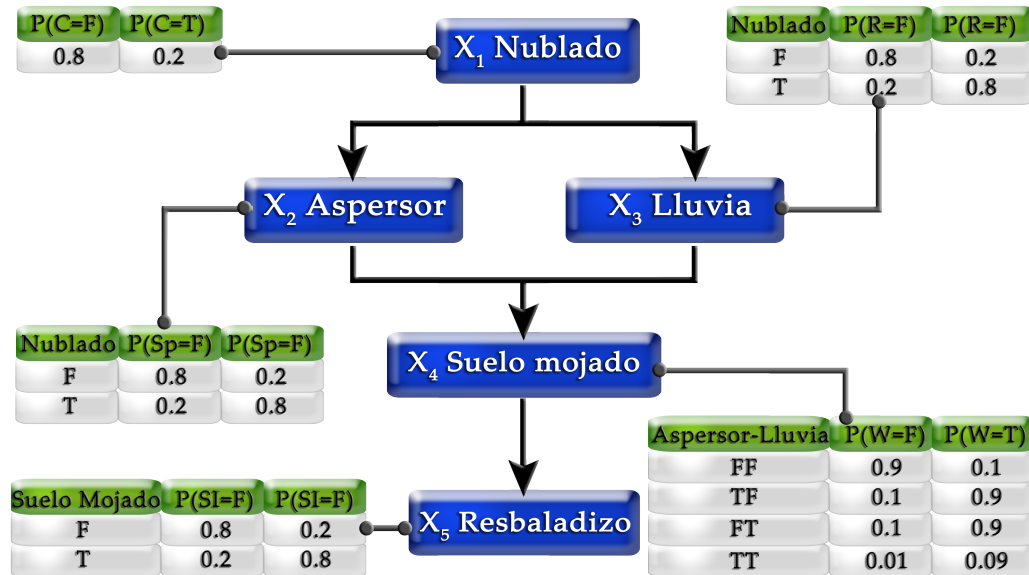


Figura 4.3: Ejemplo de Red Bayesiana.

dirigido (*DAG*) que indica explícitamente la influencia causal entre ellas. Formalmente, son grafos dirigidos acíclicos con una función de distribución probabilística asociada que representa la fuerza de las relaciones mantenidas [Cas97]. Los nodos de dicho grafo representan variables mientras que las aristas representan dependencias condicionales entre las variables. Para las necesidades aquí expuestas, la capacidad más importante de las redes bayesianas es su habilidad para inferir la probabilidad de que cierta hipótesis sea cierta:

Existen varios algoritmos para el aprendizaje de la estructura de una red bayesiana se utiliza *K2* [CH91] y *Tree Augmented Naïve (TAN)* [FGG97], además del clasificador *Naïve Bayes* (clasificador bayesiano ingenuo) [Lew98].

Un clasificador *Naïve Bayes* es un clasificador probabilístico basado en el teorema de *Bayes* que aplica una serie de operaciones de simplificación. La idea es que si el número de variables independientes que se manejan es demasiado grande, carece de sentido aplicar tablas de probabilidad [JWCY07], de ahí las reducciones que simplifican la muestra y que le dan el apelativo de *Bayes* ingenuo. Asimismo,

asume que no existe ningún tipo de dependencia entre los nodos de la red bayesiana [Cas97]. Por tanto, cada nodo en la red está únicamente vinculado con el nodo conclusión de la misma. El clasificador combina el modelo bayesiano con una regla de decisión (ecuación 4.4) donde cada f_i corresponde a cada una de las características conocidas del espécimen a clasificar.

$$\arg \max_{c \in C} P(c) = \prod_{i=1}^n p(F_i = f_i | C = c) \quad (4.4)$$

El algoritmo *K2* busca la estructura de red más probable dado un conjunto de datos. Para esto, el algoritmo adapta un algoritmo *hill climbing* restringiendo el orden en las variables [FGG97]. La diferencia de *K2* con el *hill climbing* sencillo es que este algoritmo no considera las aristas de la primera red ingenua para borrarlas posteriormente.

El algoritmo *K2* comienza asumiendo que un nodo no tiene padre para, seguidamente, añadir de forma incremental los padres cuya adición incremente la probabilidad de la estructura resultante donde π_i es el conjunto de padres del nodo x_i . q_i es el cardinal del conjunto $p_h i_i$, que contiene todas las posibles instancias de los nodos padre de x_i en el conjunto de datos D . Por otro lado, r_i es el cardinal del conjunto V_i que contiene todos los posibles valores para el nodo x_i . $\alpha_{i,j,k}$ es el número de especímenes dentro de D donde un atributo x_i se instancia con su valor k -ésimo, y los nodos padre de x_i en π_i se instancian con la j -ésima instancia en ϕ_i .

$$f(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(r-1)!}{(N_{i,j} + r_i - 1)!} \cdot \prod_{k=1}^{r_i} \alpha_{i,j,k}! \quad (4.5)$$

Ocasionalmente, el algoritmo termina de añadir padres a los nodos. Esto sucede cuando al añadir un nodo padre no se incrementa la probabilidad de la estructura de la red [LRS05] y, por tanto, el algoritmo finaliza su ejecución, dando como resultado la estructura de la red bayesiana final.

Por otro lado, el método *TAN* es un método que determina el peso máximo que abarca un árbol para luego devolver una red bayesiana ingenua aumentada con dicho árbol. Este método mejoraba los resultados obtenidos por *Naïve Bayes* en términos de precisión [FGG97] mediante la imposición del árbol de decisión sobre

4. UTILIZACIÓN DE OPCODES COMO PREDICTOR DE LA DETECCIÓN

la estructura bayesiana, con la ventaja de que mantiene la simplicidad computacional y la robustez característica del clasificador *Naïve Bayes*.

4.3.4 **K** vecinos más cercanos

El algoritmo *K vecinos más cercanos* o *KNN* [FHJ52] es uno de los algoritmos de clasificación más simples de los relativos a las técnicas de aprendizaje automático. Se trata de un método de clasificación basado en el análisis de los k vecinos más próximos en el espacio analizado ($\forall k \in N$) como se observa en la Figura 4.4. En este caso y dada la simplicidad del algoritmo, los valores de k empleados para comprobar la efectividad del mismo han sido $k = 1, 3, 5$ a fin de determinar si la toma en consideración de más vecinos mejora significativamente los resultados.

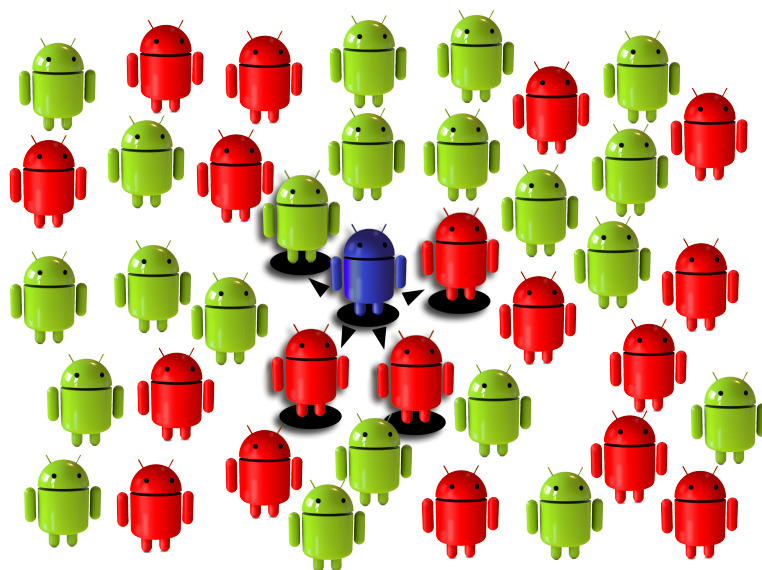


Figura 4.4: Ejemplo del funcionamiento de *KNN* en un espacio bidimensional para $k = 4$

En la fase de entrenamiento de este algoritmo, se utiliza un conjunto de instancias de entrenamiento $S = s_1, s_2, \dots, s_{n-1}, s_n$ donde n es el número de variables de cada instancia, como por ejemplo, los atributos que aparecen procedentes de cada

paquete o secuencia de paquetes. Así, estos datos son representados posteriormente en un espacio de entrenamiento.

En la fase de clasificación se mide la distancia entre los especímenes del conjunto de datos de entrenamiento y la instancia a clasificar. La distancia entre dos puntos $P = (p_1, p_2, \dots, p_n)$ y $Q = (q_1, q_2, \dots, q_n)$ representados en n dimensiones ($P, Q \in R^n$) se puede medir utilizando cualquier métrica de distancia. Para ello, la herramienta *Weka* implementa diferentes métricas como la distancia *Manhattan* $d_{Man}(P, Q)$, definida como:

$$d_{Man}(P, Q) = |p_1 - q_1| + |p_2 - q_2| + \dots + |p_n - q_n| \quad (4.6)$$

Sin embargo, en el caso de esta experimentación se emplea la distancia euclídea $d_E(P, Q)$, entendida como la raíz cuadrada del sumatorio de los cuadrados de las diferencias entre los n coeficientes de dos puntos dados:

$$d_E(P, Q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \dots + (p_n - q_n)^2} = \sqrt{\sum_{i=1}^n (p_i - q_i)^2} \quad (4.7)$$

Aunque existen varios métodos para determinar la clase de una instancia, la técnica más utilizada consiste en elegir la clase de la instancia como aquella más común entre los k puntos del espacio de entrenamiento más cercanos al espécimen a clasificar.

4.3.5 Simple Logistic

Este algoritmo se usa para predecir el resultado de una variable en función de las variables independientes o predictoras. La fórmula de regresión logística es:

$$Y_i = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_{1,i} + \dots + \beta_d X_{d,i})}} \quad (4.8)$$

Siendo Y_i la clasificación a predecir por el modelo, en nuestro caso sería *goodware* o *malware*. La variable X es el vector generado con los *opcodes* para una aplicación específica, encontrándonos que $X_{d,i}$ es el valor asignado a un n-grama de *opcodes* en la posición d de la aplicación que se encuentra en la fila i . Los parámetros β_* son determinados por el algoritmo en la fase de entrenamiento.

4. UTILIZACIÓN DE OPCODES COMO PREDICTOR DE LA DETECCIÓN

4.3.6 Validación cruzada

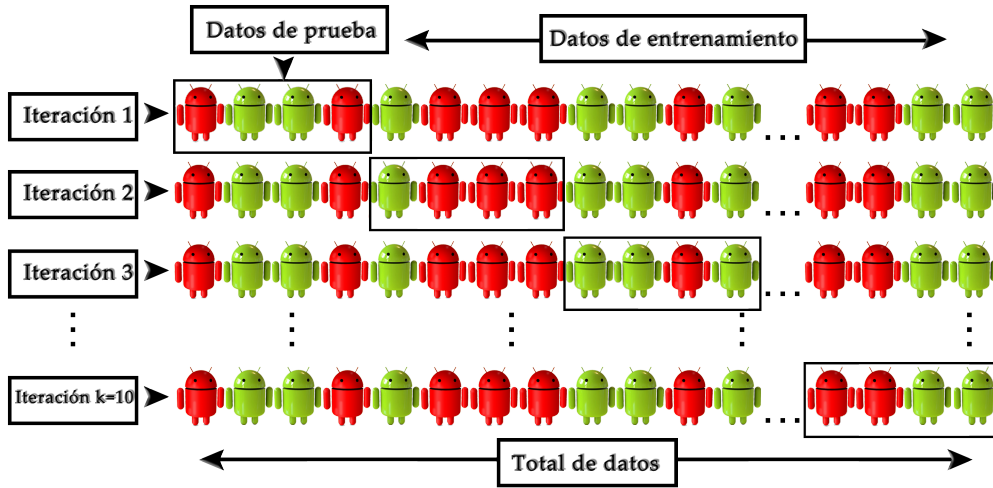


Figura 4.5: Ejemplo de los conjuntos de entrenamiento en la validación cruzada para una división de $K = 10$ conjuntos

Una de las formas de evaluar el comportamiento de los clasificadores es utilizar técnicas de *cross validation* o validación cruzada. Estas técnicas dividen un conjunto de datos en k conjuntos de entrenamiento diferentes formados por el $[100 - (k - 1) \cdot \frac{100}{k}] \%$ de las muestras como se muestra en la Figura 4.5. Tras el entrenamiento, el modelo será validado a su vez con el $(\frac{100}{k}) \%$ restante de las muestras. El error del sistema E se puede definir como sigue (ecuación 4.9):

$$E = \frac{1}{k} \cdot \sum_{i=1}^k E_i \quad (4.9)$$

Siendo E_i el error propio de cada una de las iteraciones. El valor de k utilizado en *Weka* para esta experimentación es de $k = 10$, por lo que cada conjunto de datos utilizado se divide en diez conjuntos de entrenamiento y de prueba: los primeros estarán formados por el 90 % de todas las representaciones mientras que los segundos, los utilizados para analizar las capacidades del modelo, por el 10 % restante.

4.4 Validación experimental

Para obtener la información de las muestras se hizo uso de la herramienta *Dedexer*, un desensamblador para los ficheros *.DEX* que nos encontramos en la plataforma *Android*. Mediante esta herramienta obtuvimos los códigos operacionales de las diferentes muestras a analizar. Dichos códigos operacionales son las instrucciones mínimas que entiende la máquina virtual *Dalvik* para poder ejecutar aplicaciones sobre ella. En la Figura 4.6 podemos ver una secuencia de códigos operacionales extraídos de una aplicación de *Android*.

Figura 4.6: Example of operational codes with variables used in an *Android* application.

const-string	v0, aCursor
invoke-interface	v3, v0, ;ref Map.get(ref) imp. @ _def_Map_get@LL;
move-result-object	v0
check-cast	v0, ;t: String;
const-string	v1, aHasmore
invoke-interface	v3, v1, ;ref Map.get(ref) imp. @ _def_Map_get@LL;
move-result-object	v1
check-cast	v1, ;t: String;
const-string	v3, aTrue_0
invoke-virtual	v3, v1, ;boolean String.equalsIgnoreCase(ref) imp. @ _def_String_equalsIgnoreCase@ZL;
move-result	v7
invoke-virtual	this, v0, ;boolean KiwiPurchaseUpdatesCommandTask.isNullOrEmpty(ref) imp. @ _def_KiwiPurchaseUpdatesCommandTask.isNullOrEmpty@ZL;
move-result	v1
if-eqz	v1, loc_C1AD2
sget-object	v6, Offset-BEGINNING

Con todas las muestras desensambladas hemos generado ficheros de *opcodes* que van desde 10 *Kb* hasta los 32 *Mb* en el caso del *goodware* y desde los 5 *Kb* hasta los 20 *Mb* en el caso del *malware*.

En dichos ficheros podemos encontrarnos los diferentes códigos operacionales usados por las aplicaciones tanto benignas como maliciosas. A partir de aquí se ha generado un fichero *Arff* para su uso con la herramienta *Weka*.

Para cada uno de ellos, se realizará un experimento con el fin de demostrar su validez como predictores utilizando los diferentes clasificadores detallados en la

4. UTILIZACIÓN DE OPCODES COMO PREDICTOR DE LA DETECCIÓN

sección 5.4.

Así, en esta sección detallaremos los resultados obtenidos a la hora de evaluar los distintos códigos operacionales.

Tabla 4.2: Capacidad de los clasificadores usando los códigos operacionales como característica de la clasificación.

Clasificador	<i>TPR</i>	<i>FPR</i>	<i>Accuracy</i> (%)	<i>AUC</i>
<i>IBk k=1</i>	0,94408	0,04217	95,761	0,97198
<i>IBk k=3</i>	0,93558	0,04202	95,727	0,97924
<i>IBk k=5</i>	0,93344	0,04432	95,495	0,98102
<i>Simple Logistic</i>	0,92177	0,05535	94,379	0,97839
<i>NaïveBayes</i>	0,83789	0,20294	80,561	0,88461
<i>BayesNet K2</i>	0,82057	0,19325	81,006	0,87881
<i>BayesNet TAN</i>	0,89365	0,09793	90,181	0,94768
<i>SMO PolyKernel</i>	0,92995	0,04916	95,022	0,94039
<i>SMO Norm. PolyKernel</i>	0,90985	0,04828	95,002	0,93078
<i>J48</i>	0,92581	0,05901	94,071	0,94434
<i>RandomTree</i>	0,91454	0,06147	93,749	0,92654
<i>RandomForest I=10</i>	0,95147	0,05091	94,953	0,98879
<i>RandomForest I=50</i>	0,94853	0,03645	96,322	0,99208
<i>RandomForest I=100</i>	0,94829	0,032	96,758	0,99255

Esta evaluación se realizará en base a cuatro parámetros:

- *True Positive Ratio (TPR)*, el cual se calcula dividiendo el número de paquetes de muestra legítimos correctamente clasificados (TP) entre el total de muestras extraídas ($TP + FN$).

$$TPR = \frac{TP}{(TP + FN)} \quad (4.10)$$

- *False Positive Ratio (FPR)*, el cual se calcula dividiendo el número de muestras de paquetes correspondientes a tráfico de *botnet* cuya clasificación se erró (FP) entre el número total de muestras ($FP + TN$).

$$FPR = \frac{FP}{(FP + TN)} \quad (4.11)$$

- *Accuracy (P)*, que se calcula dividiendo el total de aciertos entre el número total de instancias del conjunto de datos.

$$Acc = \frac{TP + TN}{TP + FP + TN + FN} \quad (4.12)$$

- *Area Under ROC Curve (AUC)* [SKM09], la cual establece la relación entre los falsos negativos y los falsos positivos. La curva *ROC* se suele usar para generar estadísticas que representan el rendimiento del clasificador.

Siendo así, los resultados obtenidos se visualizan en la Tabla 4.2. Los mejores resultados han sido obtenidos para el clasificador *Random Forest* con número de árboles igual a 100, con una precisión de clasificación del 96,758 % y un valor del área por debajo de la curva *ROC* de 0,99255.

Por contra, se ha observado que los clasificadores que peor han funcionado han sido *Naive Bayes* (con una precisión total del 80,561 % y un valor de *AURC* de 0,88461) y *BayesNet* con el algoritmo *K2* (con una precisión total del 81,006 % y un valor de *AURC* de 0,87881).

En la Tabla 4.3 podemos ver los resultados de clasificación de aplicaciones maliciosas usando los permisos de aplicación con nuestro *dataset*, llevado a cabo por Sanz et al. [SSL⁺13]. En dicha tabla podemos ver como uno de los mejores clasificadores, usando los permisos de aplicaciones, es *Random Forest* con número de árboles igual a 100 teniendo el área bajo la curva *ROC* de 0,99382 y una precisión del 95,423 %.

También podemos comprobar que uno de los peores clasificadores es *SMO* usando *Normalized PolyKernel* con un área bajo la curva *ROC* de 0,95329 y una precisión del 94,751 %.

Si comparamos los dos métodos se puede observar que las dos aproximaciones siguen más o menos la misma tendencia en cuanto a valores obtenidos. Por un lado podemos ver que el área bajo la curva *ROC* tiende a estar en valores del 95 % o superior en la mayoría de los clasificadores. Por otro lado si hay que tener en cuenta que el mejor clasificador para ambas aproximaciones es el *Random Forest* con número de árboles igual a 100.

4. UTILIZACIÓN DE OPCODES COMO PREDICTOR DE LA DETECCIÓN

Tabla 4.3: Capacidad de los clasificadores usando los permisos como característica de la clasificación.

Clasificador	<i>TPR</i>	<i>FPR</i>	<i>Accuracy</i> (%)	<i>AUC</i>
<i>IBk k=1</i>	0,94615	0,04924	95,103	0,98597
<i>IBk k=3</i>	0,93701	0,05281	94,717	0,98737
<i>IBk k=5</i>	0,93312	0,05893	94,127	0,98709
<i>Simple Logistic</i>	0,96235	0,0587	94,288	0,9894
<i>NaïveBayes</i>	0,95392	0,21255	81,854	0,95679
<i>BayesNet K2</i>	0,96981	0,22208	81,463	0,96868
<i>BayesNet TAN</i>	0,97689	0,09284	91,378	0,98233
<i>SMO PolyKernel</i>	0,96616	0,05957	94,221	0,95329
<i>SMO Norm. PolyKernel</i>	0,9745	0,05433	94,751	0,96009
<i>J48</i>	0,95195	0,06457	93,686	0,96116
<i>RandomTree</i>	0,93178	0,06179	93,824	0,95412
<i>RandomForest I=10</i>	0,961	0,04892	95,189	0,99148
<i>RandomForest I=50</i>	0,95806	0,04551	95,511	0,99361
<i>RandomForest I=100</i>	0,94829	0,04654	95,423	0,99382

4.5 Sumario

En este capítulo hemos hecho un enfoque orientado a la detección estática de la detección de *malware*. Este método se basa en recuperación de información y utiliza un modelo de espacio vectorial. El ejecutable se representa con secuencias de *opcodes*, mediante la frecuencia de aparición y la relevancia de los *opcodes* que componen la secuencia, formando un vector para representarlo en el espacio vectorial.

Para llevar a cabo este enfoque, hemos obtenido dos conjuntos de datos que representen tanto aplicaciones de código malicioso, cómo aplicaciones de código benigno. Para ello hemos confeccionado dos conjuntos de datos con aplicaciones benignas (*goodware*) y maliciosas (*malware*). La obtención de estas últimas se hizo

a través del conjunto de datos *Drebin Dataset* proporcionado por la *University of Göttingen* [ASH⁺14]. Por contra, para la obtención de las muestras de código benigno, hemos procedido a la descarga de 7.062 aplicaciones. Una vez descargadas las aplicaciones procedimos a la verificación de las mismas a través del servicio *Virus-Total*, herramienta que comprueba las muestras con/contra más de 40 motores de detección de distintas casas de antivirus.

A fin de obtener un conjunto de datos de *software* benigno lo más fidedigno posible, procedimos a eliminar de dicho conjunto todas las muestras que eran detectadas como código malicioso aunque fuera únicamente por un motor antivirus. De esta forma se igualó el conjunto de datos de código malicioso con un total de 5.559 muestras con el de conjunto de datos de código benigno.

El método realizado para la detección estática consiste en desensamblar una aplicación para su análisis haciendo uso de herramientas de inteligencia artificial. Para ello modelamos los *opcodes* del conjunto de datos seleccionado (tanto aplicaciones benignas como de aplicaciones malignas), para posteriormente representar los *opcodes* en un espacio vectorial y determinar la relevancia de cada uno de ellos respecto a su categoría, haciendo uso de la técnica de *Term Frequency - Inverse Document Frequency (TF-IDF)*.

Seguido, 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*), árboles aleatorios (denominado *Random Tree*) y bosques aleatorios (denominado *Random Forest*), todos ellos con distintas configuraciones, y con el objetivo de determinar cuál es el que ofrece un mejor resultado.

«Cada fracaso enseña al hombre algo que debía aprender.»

Charles Dickens (1812 – 1870)

CAPÍTULO

5

Análisis del tráfico de red en la detección de *malware*

LAS posibilidades que ofrece la gestión de ingentes cantidades de equipos y/o redes está atrayendo a un cada vez mayor número de desarrolladores de *malware*. En este capítulo, proponemos una metodología de trabajo para la detección de tráfico malicioso, basándose en el análisis del flujo de paquetes que circulan en la red.

Este objetivo se consigue mediante la parametrización de las características de esos paquetes para ser analizadas posteriormente con técnicas de aprendizaje supervisado enfocadas al etiquetado de tráfico, de modo que se permita hacer frente de una forma proactiva al gran volumen de información que manejan los filtros actuales.

Puesto que los dispositivos móviles son terminales continuamente conectados a la red de redes, se puede pensar que el análisis de los paquetes procedentes de las aplicaciones podría modelarse para la obtención de una clasificación válida en términos de *goodware* o *malware*.

Como se comprobó en anteriores publicaciones [BdlPB12], el uso del tráfico proveniente de aplicaciones en los ordenadores convencionales, es un buen enfoque para la detección de aplicaciones maliciosas. A lo largo de este capítulo volveremos

5. ANÁLISIS DEL TRÁFICO DE RED EN LA DETECCIÓN DE MALWARE

a hacer uso de la metodología mostrada en el pasado para comprobar si en un cambio de tecnología sigue comportándose de manera eficiente.

La estructura de este capítulo está organizada como sigue. En la sección 5.1 detallaremos los diferentes protocolos de comunicación existentes. En dicha sección se hace una explicación de los dos metodos de comunicación: i) *User Datagram Protocol* y ii) *Transmission Control Protocol*. En la sección 5.2 detallaremos el trabajo realizado por otros autores para la modelización de tráfico. En la sección 4.2 concretaremos el alcance de esta investigación describiendo los procesos de captura de datos. En la sección 5.3 definiremos las técnicas y métodos empleados para la definición de las capturas.

En la sección 5.4 y 5.6 explicaremos los métodos de aprendizaje automático utilizados y aglutinaremos el conjunto de experimentos realizados y los resultados obtenidos tras su ejecución. Por último, en la sección 5.7 recogeremos las conclusiones extraídas durante la realización de este trabajo.

5.1 Protocolos de comunicación

Cuando hablamos de una comunicación a través del medio de red, nos referimos al envío de la información desde un dispositivo a un servidor u otro dispositivo. El funcionamiento de este sistema es muy sencillo; se divide la información en unidades de tamaño reducido, o también llamados paquetes, enviándolos de forma totalmente autónoma a través del medio de red, y siendo de nuevo ensamblados en el destinatario.

La problemática surgida en la década de los años ochenta a este método, era que la gran mayoría de las empresas tenía su particular forma de ver las redes por lo que se hacia sumamente complicado la comunicación [Zim80, Sta04]. Esto, fue el detonante de que surgiera la estandarización del método de comunicación entre las diferentes máquinas. De esta forma surgió la pila de protocolos *TCP/IP*, compuesta por cinco capas con objetivos muy bien diferenciados [Coo96]: i) la capa de nivel físico, ii) la capa de nivel de enlace, iii) la capa de nivel de red, iv) la capa de nivel de transporte y v) la capa de aplicación.

1. *Capa de nivel físico*. Es la encargada de hacer las transformaciones que se hacen a la secuencia de *bits* para transmitirlos de un lugar a otro.

5.1 Protocolos de comunicación

2. *Capa de nivel de enlace.* Es responsable de la transferencia fiable de información a través de un circuito de transmisión de datos.
3. *Capa de nivel de red.* Proporciona conectividad y selección de ruta entre dos sistemas de hosts que pueden estar ubicados en redes geográficamente distintas.
4. *Capa de transporte.* Encargado de la transferencia libre de errores de los datos entre el emisor y el receptor, aunque no estén directamente conectados, así como de mantener el flujo de la red.
5. *Capa de aplicación.* Ofrece a las aplicaciones la posibilidad de acceder a los servicios de las demás capas y define los protocolos que utilizan las aplicaciones para intercambiar datos, como correo electrónico (*POP* y *SMTP*), gestores de bases de datos y protocolos de transferencia de archivos (*FTP*).

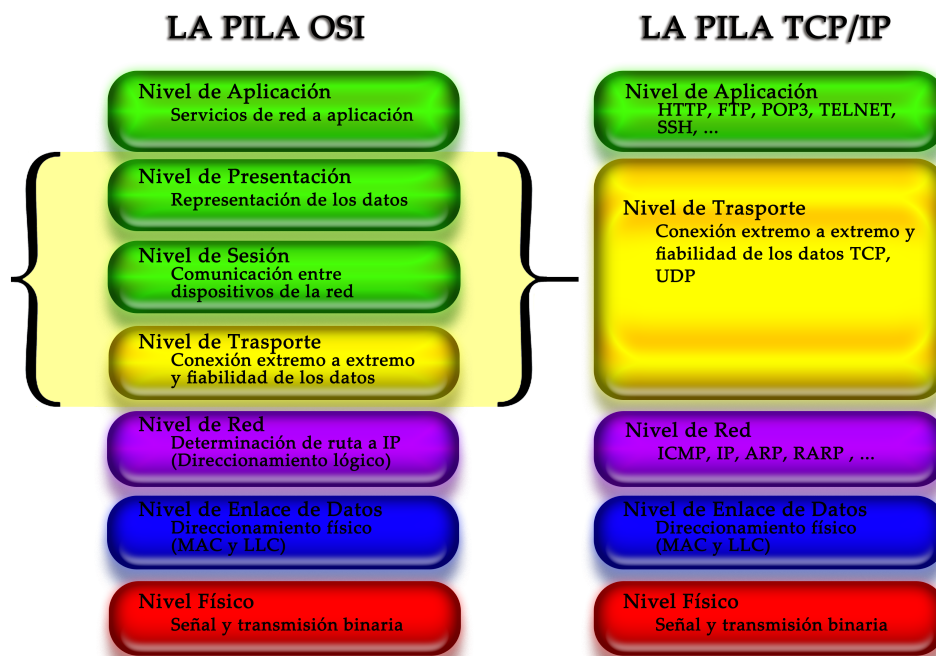


Figura 5.1: Diferencia entre la pila de protocolos OSI y pila TCP/IP

5. ANÁLISIS DEL TRÁFICO DE RED EN LA DETECCIÓN DE MALWARE

Mas adelante, la Organización Internacional para la Estandarización (*ISO*) fijaría un nuevo marco de referencia para definir arquitecturas que permitan la conexión y la interactividad entre sistemas de comunicación de diferentes fabricantes: el modelo de interconexión de sistemas abiertos o *ISO/IEC 7498-1* [fSEC94] o también conocida como la Pila *OSI* la cual añadía nuevas capas a la pila de protocolos *TCP/IP*. Estas dos capas son: i) capa de sesión y ii) capa de presentación.

1. *Capa de sesión*. Proporciona los mecanismos para controlar el diálogo entre las aplicaciones de los sistemas finales.
2. *Capa de presentación*. Se encarga de la representación de la información, de manera que aunque distintos equipos puedan tener diferentes representaciones internas de caracteres (*ASCII*¹, *Unicode*², *EBCDIC*³), números, sonido o imágenes lleguen de manera reconocible.

En esta investigación, nos centramos en los paquetes de red, como uno de los elementos primordiales dentro del protocolo de comunicación.

5.1.0.1 *User Datagram Protocol*

El *User Datagram Protocol (UDP)* se definió con la idea de crear un sistema de comunicación de datagramas de paquetes dentro de un conjunto interconectado de redes de ordenadores. Al usar este protocolo también se supone que el *Protocolo de Internet (IP)* [Pos81] se utiliza como el protocolo subyacente.

Este protocolo proporciona un procedimiento para que los programas de la capa de aplicación, puedan enviar mensajes con otros programas mediante una estandarización. Este protocolo no está orientado a la transacción, por lo tanto, no está garantizada ni la entrega, ni la protección del paquete enviado. Si por contra, una aplicación tuviese la necesidad de requerir una entrega confiable de los datos, tendría que hacer uso del *Protocolo de Control de Transmisión (TCP)*. Las características principales de este protocolo son [Pos80]: i) trabaja sin conexión, es decir

¹ Acrónimo de *American Standard Code for Information Interchange* o Código Estándar Estadounidense para el Intercambio de Información.

² El término *Unicode* proviene de los tres objetivos perseguidos: universalidad, uniformidad y unicidad.

³ Acrónimo de *Extended Binary Coded Decimal Interchange Code*.

que no emplea ninguna sincronización entre el origen y el destino, ii) trabaja con paquetes o datagramas enteros, no con bytes individuales como *TCP*. Una aplicación que emplea el protocolo *UDP* intercambia información en forma de bloques de bytes, de forma que por cada bloque de bytes enviado de la capa de aplicación a la capa de transporte, se envía un paquete *UDP*, iii) no es fiable, puesto que no emplea control del flujo ni ordena los paquetes, iv) provoca poca carga adicional en la red ya que es sencillo y emplea cabeceras muy simples, v) un paquete *UDP* puede ser fragmentado por el protocolo *IP* para ser enviado fragmentado en varios paquetes *IP* si resulta necesario, y vi) puesto que no hay conexión, un paquete *UDP* admite utilizar como dirección *IP* de destino la dirección de *broadcast* o de *multicast* de *IP*, permitiendo enviar un mismo paquete a varios destinos.

5.1.0.2 *Transmission Control Protocol*

El *Protocolo de Control de Transmisión (TCP)* se usa para una comunicación fiable de un *host* a otro *host* y en los que la comunicación entre los diferentes *hosts* tiene que estar garantizada.

TCP está diseñado para caber en una jerarquía de protocolos que soporte aplicaciones multired. El protocolo nos establece una comunicación entre diferentes pares de procesos de varios *host* conectados a las redes de comunicación distintas pero interconectadas entre sí.

El protocolo *TPC* está basado en los estudios que llevaron a cabo en un primer momento Cerf et ál. [CI05]. Este protocolo lo encajamos directamente en la parte superior del protocolo de Internet, el cual proporciona un medio para que *TCP* pueda enviar y recibir los segmentos de longitud variable con la información incluida en paquetes de datagramas.

Las características más importantes dentro del protocolo *TCP* son las siguientes [Pos83]: i) colocar los datagramas nuevamente en orden cuando vienen del protocolo *IP*, ii) permite el monitoreo del flujo de datos, evitando la saturación de la red, iii) permitir que los datos se formen en segmentos de longitud variada para entregarlos al protocolo *IP*, iv) permitir la multiplexación de los datos pudiendo circular información de diferentes fuentes de manera simultanea, y v) permitir comenzar y finalizar la comunicación amablemente.

5. ANÁLISIS DEL TRÁFICO DE RED EN LA DETECCIÓN DE MALWARE

Tabla 5.1: Tabla comparativa de ventajas y desventajas entre los protocolos de comunicación *UDP* y *TCP*(12)

Ventajas de <i>UDP</i>	Ventajas de <i>TCP</i>
• No te restringe a un modelo de comunicación basado en la conexión, la latencia para el inicio en aplicaciones distribuidas es mucho menor, al igual que la sobrecarga del sistema operativo. • Todo el control de flujo, los acuses de recibo, el registro de transacciones, etc. depende de los programas de usuario. Además, sólo es necesario implementar y utilizar las funciones que necesita. • El receptor de los paquetes <i>UDP</i> los recibe sin fragmentar, incluyendo los límites de los bloques. • <i>Broadcast</i> y transmisión <i>multicast</i> están disponibles con <i>UDP</i>.	• El sistema operativo hace todo el trabajo, el manejo de paquetes de entrada tiene menos cambios de contexto del <i>kernel</i> al espacio de usuario y de vuelta, todo la reconstrucción, acuse de recibo, control de flujo, etc. se lleva a cabo por el <i>kernel</i>. • <i>TCP</i> garantiza tres cosas: que sus datos lleguen, que lleguen en orden, y que lleguen sin duplicaciones. • Los <i>routers</i> pueden notar los paquetes <i>TCP</i> y los tratan de forma especial. los pueden almacenar en <i>buffer</i> y los retransmiten. • <i>TCP</i> tiene un buen rendimiento relativo a través de un módem o una <i>LAN</i>.

Tabla 5.2: Tabla comparativa de ventajas y desventajas entre los protocolos de comunicación UDP y TCP(2/2)

Desventajas de UDP	Desventajas de TCP
• No hay garantías con <i>UDP</i>. un paquete puede no ser entregado, o entregado dos veces o entregado fuera de orden, no se obtiene ningún indicio de esto a menos que el programa de escucha en el otro extremo decide decir algo. • <i>UDP</i> no tiene control de flujo. La implementación es el deber de los programas de usuario. • Los <i>routers</i> son muy descuidados con <i>UDP</i>. nunca se retransmiten si colisionan, y parecen ser la primera cosa descartada cuando un <i>router</i> está corto de memoria. <i>UDP</i> sufre más pérdida de paquetes que <i>TCP</i>.	• El sistema operativo puede ser defectuoso. Puede ser ineficaz, y puede que no se pueda afinar. • <i>TCP</i> es difícil de expandir, se puede establecer una pocas opciones de socket, pero tiene que tolerar el control de flujo incorporado. • <i>TCP</i> puede tener un montón de características que no son necesarias, puede desperdiciar ancho de banda, tiempo o esfuerzo en asegurar cosas que son irrelevantes para la tarea en cuestión. • <i>TCP</i> no tiene límites de bloques, debe crear el suyo. • Los <i>routers</i> de la <i>Internet</i> de hoy en día están agotando su memoria, no pueden prestar mucha atención a <i>TCP</i>, las asumpciones de diseño de <i>TCP</i> se descomponen en este entorno. • <i>TCP</i> tiene rendimiento relativamente pobre en conexiones de alta latencia, gran ancho de banda como una conexión por satélite o con sobrecarga. • <i>TCP</i> no puede ser utilizado para <i>broadcast</i> o transmisión <i>multicast</i>. • <i>TCP</i> no puede concluir una transmisión sin todos los datos en movimiento explícitamente confirmados.

5.2 Estado del arte

La clasificación en tiempo real del tráfico tiene el potencial para resolver los problemas asociados a la gestión de red que los proveedores de *Internet* y administradores de sistemas tienen que gestionar. Por esta razón, los administradores de red necesitan saber lo que está fluyendo a través de sus redes.

Los métodos de clasificación del tráfico de red pueden ser una parte fundamental de los sistemas de detección de intrusiones o *IDS* (de sus siglas en inglés *Intrusion Detection System*) [R⁺99, Pax99]. Estos sistemas se utilizan para [SABZ05]: i) la detección de patrones indicativos de ataques de denegación de servicio, ii) reasignación de los recursos de la red para los clientes más prioritarios, iii) la identificación de los clientes de la red que hagan un uso malicioso de la misma.

Muchas aplicaciones utilizan números de puerto cada vez más impredecibles [KBB⁺04]. Como consecuencia, las técnicas de clasificación más sofisticadas comprueban el tipo de aplicación mediante la búsqueda de los datos específicos de la aplicación dentro del *payload* de *TCP* o *UDP* [SSW04].

Cuando hacemos una clasificación tradicional del tráfico *IP* nos basamos en [NA08]: i) la inspección de los paquetes provenientes del protocolo *TCP* o el del puerto *UDP*, o ii) la reconstrucción de la firma del *payload*.

- *Inspección basada en el puerto.* *TCP* y *UDP* permiten la multiplexación de flujos entre diferentes terminales *IP* mediante el número de puerto. Por ello, muchas aplicaciones usan un puerto conocido en el *host* como punto de entrada para que otros terminales puedan iniciar una comunicación. Desde este punto, es posible obtener el número de puerto de la conexión para después, buscarlo dentro de una lista de puertos registrados (*IANA* o *Internet Assigned Numbers Authority* de sus siglas en inglés) [Zei02].

Este tipo de enfoque tiene sus limitaciones, por ejemplo, algunas aplicaciones pueden no tener sus puertos registrados en *IANA* como puede ser el caso de aplicaciones *P2P* [RSSD04].

Por otro lado, Moore et ál. [MP05] observaron que el uso de la lista *IANA* para este tipo de clasificación no era buen enfoque ya que únicamente obtenía un 70 % de precisión en la clasificación. A su vez, Madhukar et ál. [MW06]

demonstraron que el análisis basado en los puertos no puede identificar entre el 30 y el 70 % del tráfico que existe en *Internet*.

- *La reconstrucción de la firma del payload*. En la actualidad, muchas aplicaciones para evitar tener que hacer uso de los puertos, utilizan un sistema de reconstrucción de estado de sesión a partir de la información contenida en el paquete. Sen et ál. [SSW04] demostró que la clasificación del *payload* del tráfico en redes P2P podía reducir el número de falsos positivos y falsos negativos.

A su vez Moore, et ál. [MP05] hicieron una comprobación híbrida entre los puertos y el *payload* del paquete. En una primera aproximación comprobaban si el puerto pertenecía a la lista *IANA*, si no pertenece pasaría a la siguiente fase. En la segunda fase, examinarían los paquetes de la conexión. En ellos analizarían si la firma del *payload* es conocida o desconocida.

A pesar de que esta aproximación nos ofrece una dependencia a la necesidad de conocer los números de puerto, incorpora una complejidad en el procesamiento del *payload* en el dispositivo. Dicho dispositivo habrá que tenerlo en continua actualización, además de la necesidad de una potencia suficiente para poder analizar una cantidad grande de paquetes de red. Sin embargo, este enfoque puede ser totalmente inútil si por ejemplo el tráfico está cifrado o utiliza un protocolo privado.

Los enfoques anteriores están limitados por la semántica de la información obtenida, debido a la inspección necesaria de los paquetes obtenidos, así como del *payload* y los números de puerto. En cambio, estudios más recientes se basan en la obtención de las características estadísticas del tráfico para identificar una aplicación. En otras palabras, el tráfico de red tiene una serie de propiedades estadísticas, tales como la distribución de la duración del flujo, el flujo de tiempo de inactividad o tiempo entre llegadas de paquetes y longitudes de paquetes, que las hacen únicas para cierto tipo de aplicaciones, permitiendo de este modo diferenciarse unas de otras.

Paxson et ál. [Pax94] analizaron y construyeron modelos empíricos sobre las características de la conexión. Dentro de este enfoque, hacían uso de los bytes de

5. ANÁLISIS DEL TRÁFICO DE RED EN LA DETECCIÓN DE MALWARE

la conexión, la duración de la misma, como la periodicidad de la llegada, extraídas de un número específico de aplicaciones que usaban *TCP*. También, Dewes et ál. [DWF03] realizaron un estudio sobre los *chat* de *Internet*. En él, se centraron en las características del tráfico, más concretamente en la duración del flujo de paquetes, tiempo entre las llegadas de paquetes y el tamaño de cada uno. En estudios posteriores se harían evoluciones de este tipo de enfoques, pero añadiendo a su vez las longitudes de los paquetes, así como la diferencia entre las llegadas de los mismos [Cla94, LABC03, LBA04]. Los resultados de estos trabajos han estimulado nuevas técnicas de clasificación basados en propiedades estadísticas de flujo de tráfico.

En el pasado se han realizado una serie de estudios en los cuales analizaban el uso del espectro y las características de la aplicación en las redes de datos móviles [PPJP05, RU05, MSZ⁺07]. La mayoría de estos estudios previos, trataron de entender el uso del espectro inalámbrico y a su vez caracterizar el rendimiento de la red.

Para conseguir crear un patrón de uso de red con su comportamiento, es necesario hacer un análisis integral de todo el tráfico generado en dicha red. Aunque algunos estudios recientes [WMBW08, KNZG10] han estado analizando las muestras recogidas en *Internet*, centrándose concretamente en el tráfico generado mediante voz o el tráfico generado por la navegación del usuario, hacen falta estudios detallados del tráfico de red.

Por otro lado, McGregor et ál. [MHLB04] publicaron uno de los primeros trabajos en relación al uso del *Machine Learning* para la clasificación del tráfico de red. El enfoque agrupa el tráfico con propiedades similares para diferentes tipos de aplicaciones.

Más adelante, Moore et ál. [MZ05] propusieron el uso del clasificador *Naive Bayes* para categorizar el tráfico generado por las aplicaciones. El tráfico usado para el *dataset* es etiquetado de forma manual para permitir una evaluación precisa por parte del clasificador.

Para llevar a cabo la clasificación, hicieron uso de 248 características extraídas para entrenar el clasificador. El tráfico seleccionado de diferentes aplicaciones lo agruparon en diferentes categorías para conseguir una buena clasificación.

También Bernaille et ál. [BTA⁺06] plantearon un enfoque haciendo uso del clasificador *K-Means* o *K-vecinos* para clasificar diferentes tipos de aplicaciones

basadas en *TCP* usando únicamente los primeros paquetes de la comunicación de la misma. En contraste con otras técnicas llevadas a cabo por otros autores, Bernaille et ál. podían detectar y clasificar antes las aplicaciones gracias al uso de los primeros paquetes de red. En definitiva, el enfoque se centra en obtener los primeros paquetes que normalmente son secuencias predefinidas de mensajes entre las distintas aplicaciones.

Nguyen et ál. [NA06] describieron un método para abordar la clasificación del tráfico de red a través de un número finito N de paquetes de red. Como consecuencia directa del uso de un número pequeño de paquetes, es el decremento en el tiempo usado para la clasificación, reduciendo así mismo el espacio necesario para el almacenamiento intermedio de la información relativa a los paquetes necesarios para la clasificación. El uso de este enfoque, permite que el proceso de clasificación se inicie en cualquier momento, no únicamente al inicio de la aplicación como en enfoques anteriores.

El trabajo de Erman et ál. [EMAW07] estaba dirigido al reto de clasificar el tráfico en el núcleo de la red, donde la información disponible sobre los paquetes y sus conexiones podría ser limitada. Por ello, en el trabajo proponían el uso únicamente de la información saliente del dispositivo.

También, Crotti et ál. [CDGS07] propusieron un mecanismo de clasificación de flujo basado en tres propiedades de los paquetes *IP* capturados: i) longitud de paquete, ii) el tiempo entre llegadas y iii) el orden de llegada de paquetes. Definieron una estructura llamada huellas de protocolo que expresan las tres propiedades de tráfico en una forma compacta y utilizan un algoritmo basado en umbrales normalizados para la clasificación de flujo.

Haffner et ál. [HSSW05] plantearon un método para la generación automática de firmas de las aplicaciones haciendo uso de técnicas de aprendizaje automático. A diferencia de otros enfoques, este trabajo hace uso de los primeros *N-Bytes* de un flujo de paquetes como características para la generación de la firma.

Erman et ál. [EMA07] demostraron que la precisión en los *bytes* es crucial a la hora de evaluar los algoritmos de clasificación. En esta hipótesis, señalaron que la mayoría de los flujos en *Internet* son reducidos, representando sólo una pequeña porción de los *bytes* y paquetes en la red totales. Por otro lado, también observaron

5. ANÁLISIS DEL TRÁFICO DE RED EN LA DETECCIÓN DE *MALWARE*

que la mayoría de los *bytes* de tráfico son generados por un pequeño número de grandes flujos de paquetes de red.

También, Brezo [Bre14] demostró que el uso de los paquetes de red para la detección de canales de *Command&Control* era una buena aproximación. En dicha investigación, el autor hacía uso de *N-Packets* para la clasificación de los mismos mediante algoritmos de aprendizaje supervisado.

5.3 Metodología creada

La metodología para la obtención de muestras se centrará en recabar información sobre dos tipos de tráfico claramente diferenciados: i) por un lado, paquetes pertenecientes a conexiones legítimas llevadas a cabo por aplicaciones obtenidas del mercado de aplicaciones de *Google* y validadas como benignas y, ii) por otro, paquetes pertenecientes a aplicaciones maliciosas obtenidas de un dataset de la comunidad científica.

Seguidamente a la instalación de las aplicaciones de *goodware* y *malware*, hemos procedido a la ejecución de las mismas dentro de un entorno de pruebas controlado, evitando la inclusión de sesgos indeseados por parte de otras aplicaciones.

5.3.1 Preparación del entorno

Una vez hemos definidos la obtención de muestras necesarias para la ejecución de la experimentación, tenemos que hacer la preparación del entorno en el cual se ejecutaran dichas muestras.

La filosofía de trabajo propuesta en anteriores trabajos [BdlPB12], está basada en un IDS¹ basado en *host*. Es decir, el punto de escucha de tráfico esta situado en los nodos infectados de la red.

Puesto que las aplicaciones que se van a ejecutar están basadas en el sistema operativo *Android*, hace falta la ejecución de una máquina virtual para poder ejecutar el código de la misma. Con este fin, hemos hecho uso del sistema de máquinas

¹Del inglés, *Intrusion Detection System*.

virtuales *Genymotion*². Este, está basado en la plataforma de virtualización *Virtual-Box*³ de la compañía *Oracle*⁴. *Genymotion* provee de una cantidad de máquinas virtuales del sistema operativo *Android* modificadas para tener acceso *root*.

La máquina virtual usada para esta experimentación dispone de un procesador, con una memoria *RAM* de 1024 *MB*, una resolución de pantalla de 720x1280 y una densidad de *píxeles* de 320dpi. El sistema que hemos usado ha sido *Android* 4.1.1 *Jelly Bean*.

La monitorización del tráfico se realizará empleando una de las herramientas más conocidas: *Wireshark*⁵. *Wireshark* es una herramienta de análisis de paquetes multi-plataforma. Es habitualmente utilizada para la monitorización de problemas relacionados con la conectividad de la red, el análisis de tráfico y el desarrollo de *software* o nuevos protocolos de comunicación.

5.3.2 Proceso de automatización

La obtención de tráfico legítimo y malicioso se hizo mediante una automatización de la máquina virtual. Para crear este proceso, hemos hecho uso del lenguaje de programación C#. La Figura 5.2 muestra el proceso de automatización llevado a cabo para la obtención del tráfico.

En la primera sección del sistema, hemos hecho uso de la aplicación *Genymotion* para crear la máquina virtual inicial con las características comentadas en la subsección 5.3.1. A la máquina virtual inicial hemos tenido que instalar las *Google Apps*, una serie de aplicaciones nativas de *Android*, mediante las cuales, las aplicaciones pueden acceder a las *APIs* que *Google* ofrece.

Seguido, hacemos una copia de dicha máquina virtual, la cual será en la que nosotros instalemos las aplicaciones necesarias. Después de la creación e inicio de la máquina, asociaremos una instancia de *Wireshark* a la máquina con el propósito de obtener todos los paquetes de red de la ejecución. Posteriormente, haremos una instalación de una de las aplicaciones con su consiguiente ejecución dentro del sistema *Android*, dejándola en ejecución durante un tiempo de un minuto. Una vez

²<https://www.genymotion.com/>

³<https://www.virtualbox.org/>

⁴www.oracle.com/es/

⁵<https://www.wireshark.org/>

5. ANÁLISIS DEL TRÁFICO DE RED EN LA DETECCIÓN DE *MALWARE*

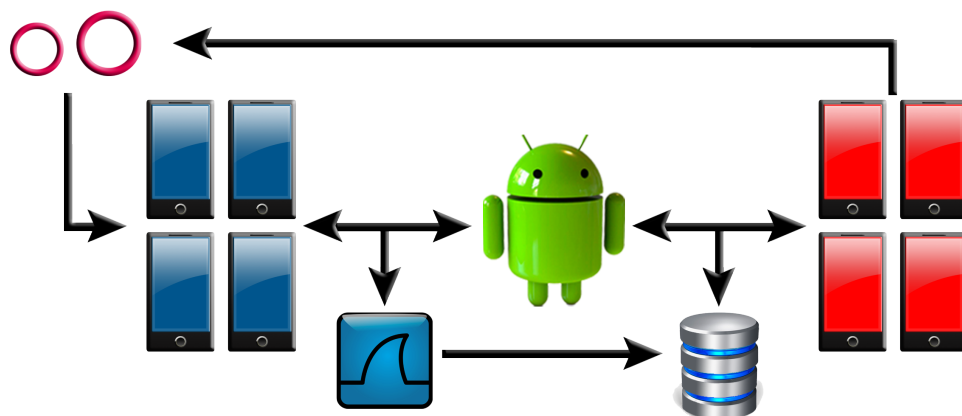


Figura 5.2: Proceso de automatización de las máquinas virtuales

transcurrido el tiempo comentado, hemos procedido a almacenar todos los ficheros de red, uno por cada aplicación ejecutada. Por último, la máquina virtual utilizada para la ejecución de una aplicación, es eliminada para volver a empezar el proceso haciendo una copia de la máquina virtual original.

5.4 Aprendizaje supervisado

De igual manera a como ya hicimos en el pasado Brezo et ál. para detectar el tráfico procedente de una botnet *HTTP* [BdlPUP⁺13], se ha planteado el problema como un problema de clasificación supervisada. En este tipo de problemas, se estudia un fenómeno representado por un vector X en R^d que puede ser clasificado de K maneras diferentes de acuerdo a un vector Y de etiquetas. El aprendizaje supervisado y la utilización de algoritmos de clasificación hacen uso de un conjunto de datos previamente etiquetado [KZP07] que, en nuestro caso, se corresponde con las etiquetas de tráfico de *goodware* o *malware* en función del origen de las representaciones utilizadas para el experimento.

Para ello hemos usado los algoritmos presentados en la sección 5.4 del Capítulo 4 más algún algoritmo añadido que procederemos a su explicación en las páginas siguientes.

5.4.1 Máquinas de soporte vectorial

En este caso hemos usado las máquinas de soporte vectorial anteriormente comentadas con los siguientes *kernels*:

- Los *kernels polinomiales* [AW99] han sido utilizados con éxito en los últimos años con los *SVM*, especialmente para afrontar los problemas relacionados con el procesamiento del lenguaje natural tanto en su versión normal que podemos ver en la ecuación 5.1:

$$k(x, xy) = (x \cdot y)^d \quad (5.1)$$

como en su versión normalizada que observamos en la ecuación 5.2:

$$K(x, xy) = \frac{(x \cdot y)^d}{\sqrt{(x \cdot y)^d}} \quad (5.2)$$

En ambos casos, d representa el grado del polinomio. Es habitual no encontrarse con valores de d superiores a 2, ya que se ha observado que se presentan problemas de sobreajuste a los datos de muestra [CHC⁺10]. Por situaciones de sobreajuste, hay que entender aquellas situaciones que se dan cuando uno de los clasificadores ofrecen resultados óptimos para el conjunto de datos de entrenamiento utilizados pero este no es capaz de acomodarse a la variedad de datos que otros proveen. Esta característica puede dar lugar a errores en la aplicación de este algoritmo de clasificación en entornos reales [SLTW04].

- **Kernel Radial Basis Function (RBF)** [AW99]. Es uno de los más populares utilizados en la clasificación empleando máquinas de soporte vectorial [CHC⁺10]. El *kernel RBF* entre dos muestras x e y representadas por vectores de atributos se define como en la ecuación 5.3:

$$K(x, y) = \exp\left(-\frac{\|x - y\|^2}{2\sigma^2}\right) \quad (5.3)$$

5. ANÁLISIS DEL TRÁFICO DE RED EN LA DETECCIÓN DE MALWARE

- *Pearson VII Universal Kernel (PUK)*. Utilizada con frecuencia en el campo de la espectroscopia¹, es una alternativa a las propuestas de *kernels* anteriores [ÜMB06]. Se define como la ecuación 5.4:

$$k(x, y) = \frac{1}{\left[1 + \left(\frac{\sqrt{\|x-y\|^2} \cdot \sqrt{2^{\frac{1}{\omega}} - 1}}{\delta} \right)^2 \right]^{\omega}} \quad (5.4)$$

5.4.2 Método *Bagging*

Bagging, abreviatura de *bootstrap aggregating* fue un método propuesto por Leo Breiman en 1994 [Bre96a] para mejorar los métodos tradicionales de clasificación mediante la combinación de clasificaciones sobre conjuntos de datos de entrenamiento generados aleatoriamente. El método *Bagging* es un meta-algoritmo diseñado para mejorar la estabilidad y precisión de otros algoritmos de aprendizaje automático en procesos de clasificación estadística y regresión. Una de las cualidades del método es que puede ser utilizado para proveer estimaciones mejoradas sobre las probabilidades reales de un nodo al utilizar valores estimados de salida y no las salidas observadas tal cual, lo que a la postre permitiría reducir la varianza y minimizar los problemas relacionados con el sobreajuste [Bre96b]. Aunque es habitualmente utilizado para los árboles de decisión también puede ser utilizado para cualquier tipo de método.

Así, dado un conjunto D de datos estándar de tamaño n , *Bagging* genera un número m de nuevos conjuntos de entrenamiento D_i , cada uno de ellos de un tamaño $n' < n$, extrayendo a continuación muestras de D uniformemente. Mediante la extracción de muestras con reemplazamiento, algunas de las observaciones pueden ser repetidas en cada uno de los conjuntos D_i , lo que ocurrirá con más frecuencia cuanto más próximo a n sea n' . En el caso particular de un conjunto de datos D_i formado por un número de muestras $n' = n$, se espera que se cuente con $1 - \frac{1}{e}$ de muestras únicas de D , siendo el resto muestras duplicadas [Bre96a]. Este

¹La espectroscopia es el estudio de la interacción entre la radiación electromagnética y la materia, con absorción o emisión de energía radiante. Tiene aplicaciones en campos como la astronomía, la física y la química.

tipo de muestreo es conocido como *bootstrap sample*. Los m modelos generados son ajustados utilizando las m muestras y son combinados promediando la salida o mediante votos.

Según Breiman, *Bagging* deriva en mejoras sustanciales de aquellos modelos inestables [Bre96a], entre los que se incluyen las redes neuronales o los árboles de clasificación y de regresión. Por este motivo, el algoritmo utilizado es *REPTree*, un árbol de decisión de aprendizaje rápido y que también ha sido utilizado en el pasado para mejorar la precisión de los procesos de *clustering* o agrupamiento [DF03]. Por otro lado, es también cierto que este método podría ver moderadamente perjudicado el rendimiento de los métodos estables, como el de *KNN*.

5.4.3 Redes bayesianas

Los algoritmos para el aprendizaje de la estructura de una red bayesiana que hemos utilizado *K2* [CH91], *Hill Climbing* [RN95] y *Tree Augmented Naïve (TAN)* [FGG97], además del clasificador *Naïve Bayes* (clasificador bayesiano ingenuo) [Lew98].

El algoritmo *K2* busca la estructura de red más probable dado un conjunto de datos. Para esto, el algoritmo adapta un algoritmo *hill climbing* restringiendo el orden en las variables [FGG97]. La diferencia de *K2* con el *hill climbing* sencillo es que este algoritmo no considera las aristas de la primera red ingenua para borrarlas posteriormente.

El algoritmo *K2* comienza asumiendo que un nodo no tiene padre para, seguidamente, añadir de forma incremental los padres cuya adición incrementa la probabilidad de la estructura resultante donde π_i es el conjunto de padres del nodo x_i . q_i es el cardinal del conjunto $p_h i_i$, que contiene todas las posibles instancias de los nodos padre de x_i en el conjunto de datos D . Por otro lado, r_i es el cardinal del conjunto V_i que contiene todos los posibles valores para el nodo x_i . $\alpha_{i,j,k}$ es el número de especímenes dentro de D donde un atributo x_i se instancia con su valor k -ésimo, y los nodos padre de x_i en π_i se instancian con la j -ésima instancia en ϕ_i .

$$f(i, \pi_i) = \prod_{j=1}^{q_i} \frac{(r-1)!}{(N_{i,j} + r_i - 1)!} \cdot \prod_{k=1}^{r_i} \alpha_{i,j,k}! \quad (5.5)$$

5. ANÁLISIS DEL TRÁFICO DE RED EN LA DETECCIÓN DE MALWARE

Ocasionalmente, el algoritmo termina de añadir padres a los nodos. Esto sucede cuando al añadir un nodo padre no se incrementa la probabilidad de la estructura de la red [LRS05] y, por tanto, el algoritmo finaliza su ejecución, dando como resultado la estructura de la red bayesiana final.

Hill Climbing [RN95] es un método de búsqueda local. El algoritmo intenta maximizar una función heurística $f(x)$. Con este fin, comienza con una solución aleatoria e iterativamente realiza pequeños cambios a esta, consiguiendo mejoras en cada iteración. Cuando el algoritmo no puede conseguir más mejoras en la función heurística, éste finaliza. A pesar de que el objetivo es encontrar una solución cercana a la óptima, no se puede garantizar que el algoritmo *Hill Climbing* encuentre la mejor solución, consiguiendo en muchas ocasiones un máximo local¹.

Como algoritmo de aprendizaje bayesiano, añade, borra y da la vuelta a las aristas basándose en una función heurística. La búsqueda no se restringe por un orden en las variables, al contrario que lo que hace *K2*. La heurística que se suele emplear para buscar las relaciones entre los nodos, es la función de puntuación bayesiana, denominada también *Bayesian scoring function* [DCP⁺08].

Por otro lado, el método *TAN* es un método que determina el peso máximo que abarca un árbol para luego devolver una red bayesiana ingenua aumentada con dicho árbol. Este método mejoraba los resultados obtenidos por *Naïve Bayes* en términos de precisión [FGG97] mediante la imposición del árbol de decisión sobre la estructura bayesiana, con la ventaja de que mantiene la simplicidad computacional y la robustez característica del clasificador *Naïve Bayes*.

5.4.4 K vecinos más cercanos

En este experimento hemos hecho uso de los K-vecinos más cercanos con los valores de $k = 1, 2, 3, 4, 5, 10$.

¹Un máximo local se corresponde con el máximo valor encontrado en un intervalo de observación pudiendo ser este inferior a otros valores obtenidos para una misma función fuera de dicho intervalo.

5.5 Evaluación general

La evaluación de cada uno de los métodos se va a realizar en base a diferentes parámetros habitualmente utilizados en la evaluación del rendimiento de los métodos de clasificación [Pow07].

- *True Positive Ratio (TPR)* Conceptualmente, en *text mining*, se identifica como la probabilidad de que un documento elegido aleatoriamente sea verdaderamente relevante en la búsqueda. En nuestro caso, se puede definir como la tasa de paquetes maliciosos detectados correctamente de entre todos los analizados por el clasificador. Se calcula dividiendo el número de paquetes maliciosos correctamente clasificados (TP) entre el total de muestras maliciosas extraídas, ya sean *true positives* o *false negatives* ($TP + FN$) (ecuación 5.6):

$$TPR = \frac{TP}{TP + FN} \quad (5.6)$$

- *Accuracy*. Se calcula dividiendo, en este caso, el total de aciertos entre el número de instancias que componen la totalidad del conjunto de datos (ecuación 5.7).

$$Acc = \frac{TP + TN}{TP + TN + FP + FN} \quad (5.7)$$

- *Precision o Exactitud*. La precisión es un valor que presenta la posibilidad de que un resultado positivo refleje la condición que ha sido testada, es decir, de que una muestra etiquetada como positiva sea efectivamente un verdadero positivo. Se define como (ecuación):

$$PPV = \frac{TP}{TP + FP} \quad (5.8)$$

- *F-measure (F)*. La *F-measure* es una métrica típica de *Information Retrieval (IR)* para evaluar el resultado de una búsqueda que determina hasta qué punto los grupos obtenidos se asemejan a los que se hubieran logrado con una categorización manual. Esta medida se puede definir como 5.9:

5. ANÁLISIS DEL TRÁFICO DE RED EN LA DETECCIÓN DE MALWARE

$$F = 2 \cdot \frac{precision \cdot recall}{precision + recall} = 2 \cdot \frac{PPV \cdot TPR}{PPV + TPR} \quad (5.9)$$

De este modo si sustituimos en F los valores de PPV y TPR podremos simplificar la fórmula como sigue:

(5.10)

- El significado del área por debajo de la curva ROC es el establecimiento de la relación entre los falsos negativos y los falsos positivos [SKM09] y se obtiene representando, para cada posible elección de valores de corte la TPR en el eje de ordenadas y la FPR en el eje de abscisas. Empleada originalmente por el ejército estadounidense en investigaciones tras el ataque de *Pearl Harbor* de 1941 para detectar correctamente las firmas de la señal de radar de las aeronaves japonesas [Gre6], se ha venido utilizando desde finales de siglo XX como métrica de comparación y evaluación de diferentes algoritmos de clasificación [Spa89]. Se suele usar para generar estadísticas que representan el rendimiento de un clasificador en un sistema binario como herramienta para seleccionar modelos posiblemente óptimos y descartar aquellos subóptimos. Aunque no proporciona información sobre el buen comportamiento del modelo, el AUC ayuda a determinar la validez de una distribución de datos a lo largo de una serie dada de condiciones predictivas [LJVR08].

5.6 Validación empírica

En esta sección vamos a proceder a la recogida de datos de los experimentos llevados a cabo para esta tesis doctoral. Como ya se ha explicado anteriormente, el proceso está compuesto por el análisis de los resultados para la representación de paquetes individuales procedentes de las aplicaciones. Para ello vamos a representar los resultados como tráfico malicioso o benigno dentro de las aplicaciones.

Como se ha comentado anteriormente en la sección 5.4, para la validación de los experimentos se ha hecho uso de algoritmos de aprendizaje supervisado con validación cruzada. En concreto hemos hecho uso de los siguientes algoritmos: i) *Random Forest*, ii) *J48*, iii) *Naïve Bayes*, iv) *BayesNet*, v) *IBK* con valores de $K = 1, 2, 3, 4, 5, 10$.

Para evaluar las capacidades de cada clasificador, hemos utilizado las métricas que hemos descrito con anterioridad para el apartado 5.5: i) *True Positive Ratio (TPR)*, ii) *Accuracy*, iii) precisión, y iv) área por debajo de la curva *ROC (AUC)*.

En primer lugar, hemos extraído las características atómicas individuales que definen cada uno de los paquetes de una determinada conexión. En total, las características totales extraídas son 23. Debido a que la cantidad de características no es numerosa no es necesario hacer uso de técnicas de *Information Gain* [Föl73] para facilitar las tareas de clasificación. En nuestro caso, el uso de esta técnica nos implicaría un coste computacional dando una mejoría marginal a los resultados.

En segundo lugar, hemos hecho una comparación de los resultados obtenidos con los trabajos realizados por Brezo et ál. [BdlPB12], y por Brezo [Bre14]. En dichos experimentos realizamos la misma metodología para la detección de tráfico procedente de una *botnet* y la clasificación del tráfico procedente de diferentes *botnets*, clasificándolo como tráfico genérico de *botnet* y a su vez como la familia específica de *malware*.

5.6.1 Capacidad del clasificador para la clasificación de tráfico

La tabla 5.3 muestra los resultados obtenidos con la metodología comentada en la sección 5.3 para el tráfico de red generado por las aplicaciones para el sistema *Android*.

El mejor resultado en términos de *Accuracy* lo obtiene el clasificador *Bagging* con un valor superior al 80 %. Pero este resultado es una excepción, puesto que la mayoría de los clasificadores rondan entre el 55 % y el 65 %. A su vez, se observa que el peor clasificador de todos es *Voted Perceptron*, el cual supera por muy poco el 50 % de las muestras bien clasificadas.

Por lo que a la tasas de verdaderos positivos se refiere, los resultados son muy elevados, con tasas que llegan al 0,99 en varios clasificadores, como las redes bayesianas o los árboles de decisión, y rondando valores cercanos y superiores al 0,90 para el resto de los clasificadores, asumiendo una fiabilidad en la clasificación superior al 80 % en todos los clasificadores. Por otro lado, el peor clasificador sigue siendo *Voted Perceptron*, el cual con una tasa de verdaderos positivos del 0,69 y

5. ANÁLISIS DEL TRÁFICO DE RED EN LA DETECCIÓN DE MALWARE

una fiabilidad del del 76 % nos deja ver que no es un buen clasificador para este tipo de metodología.

La *F-Measure* presenta resultados bastante homogéneos. Existe una variación que oscila entre el 0,81 y el 0,93 en todos los clasificadores menos, una vez más, *Voted Perceptron*, el cual obtiene un valor del 0,71, diez puntos menos que el peor clasificador del resto. Por contra, los mejores clasificadores nos los volvemos a encontrar en las redes bayesianas, usando los algoritmos de búsqueda *K2* y *Hill Climbing*, y los árboles de decisión, añadiendo además el clasificador de *KNN* con el valor de $k = 1$, todos ellos con un valor de 0,93.

El valor del área bajo la curva *ROC* más alto obtenido ha sido de un 0,71 para los clasificadores *Bagging* y *KNN* con los valores de $k = 1, 2, 3, 4$. Aún así, los valores que nos encontramos son bastante inferiores, oscilando entre el 0,53 que obtienen las máquinas de soporte vectorial y entre el 0,60 y 0,70 de las redes bayesianas y el resto de los valores de $k = 5, 6, 7, 8, 9, 10$ de *KNN*. De nuevo, el peor clasificador es *Voted Perceptron* con un valor para el área de 0,48, lo que no hace más que confirmar lo comentado anteriormente sobre la ineficacia del clasificador en este tipo de problema.

Una vez explicados los resultados obtenidos usando la metodología en nuestro enfoque, hemos comparado dichos resultados con los obtenidos en investigaciones anteriores en ordenadores de escritorio [BdlPUP⁺13, Bre14].

La Tabla 5.4 muestra la diferencia entre los resultados del mejor clasificador y peor clasificador obtenidos por la metodología en el enfoque llevado por Brezo y el enfoque llevado en esta tesis. Como podemos observar el mejor clasificador en términos de *Accuracy* ha sido *KNN* con valor de $k = 1$ con un valor de 80,54 %, por otro lado en nuestro enfoque hemos obtenido un decremento en dicho clasificador de 15 puntos, logrando un 65,79 % de correcta clasificación.

Por otro lado, el peor clasificador ha sido en los dos casos *Voted Perceptron*, con un valor similar en los dos casos alrededor del 55 %. Esto demuestra lo comentado anteriormente de que *Voted Perceptron* es en todos los casos un mal clasificador para la clasificación de paquetes de red.

La Tabla 5.5 muestra la diferencia entre los resultados del mejor clasificador y peor clasificador en términos de verdaderos positivos (*TPR*) y exactitud (*Precision*). Para este caso, los resultados han sido bastante parejos, menos en el caso de las

5.6 Validación empírica

Tabla 5.3: Resultados obtenidos por los clasificadores para el análisis de tráfico de red en aplicaciones de *Android*

Clasificador	Accuracy (%)	TRP	Exactitud	F-Measure	AUC
<i>SMO Polik.</i>	60,40	0,85	0,83	0,81	0,53
<i>SMO Polik. N.</i>	60,56	0,88	0,83	0,82	0,53
<i>SMO RBF</i>	58,85	0,88	0,80	0,81	0,52
<i>SMO Pear-VII</i>	63,63	0,80	0,90	0,82	0,56
<i>Bagging REPT</i>	81,69	0,96	0,95	0,92	0,71
<i>Naive Bayes</i>	59,58	0,93	0,87	0,90	0,60
<i>BayesNet K2</i>	65,79	0,99	0,87	0,93	0,66
<i>Bayes Net Hill</i>	65,89	0,99	0,87	0,93	0,66
<i>Bayes Net TAN</i>	66,80	0,97	0,88	0,89	0,67
<i>J48</i>	64,70	0,99	0,87	0,93	0,65
<i>Random Forest</i>	67,47	0,99	0,87	0,93	0,67
<i>Voted Per.</i>	53,36	0,69	0,76	0,71	0,48
<i>KNN k=1</i>	65,79	0,96	0,95	0,93	0,71
<i>KNN k=2</i>	65,79	0,97	0,96	0,92	0,71
<i>KNN k=3</i>	65,79	0,96	0,95	0,92	0,71
<i>KNN k=4</i>	65,79	0,96	0,94	0,92	0,71
<i>KNN k=5</i>	65,79	0,95	0,94	0,91	0,70
<i>KNN k=6</i>	65,79	0,95	0,92	0,91	0,69
<i>KNN k=7</i>	65,79	0,94	0,94	0,91	0,69
<i>KNN k=8</i>	65,79	0,94	0,92	0,90	0,69
<i>KNN k=9</i>	65,79	0,94	0,92	0,90	0,69
<i>KNN k=10</i>	65,79	0,93	0,91	0,89	0,68

Tabla 5.4: Tabla comparativa del mejor y peor clasificador en término de *Accuracy* de los resultados obtenidos por Brezo y nuestro enfoque.

Clasificador	Accuracy (%)	
	Brezo et ál.	Gaviria et ál.
<i>KNN k=1</i>	80,54	65,79
<i>Voted Perceptron</i>	55,1	53,36

5. ANÁLISIS DEL TRÁFICO DE RED EN LA DETECCIÓN DE MALWARE

redes bayesianas con algoritmos de búsqueda *K2* y *Hill Climbing* los cuales han pasado de tener un *TPR* de 0,59 a un 0,99. Por otro lado, el término de *TPR* hay que mirarlo a su vez con la exactitud o *Precision*.

En la mayoría de los casos el resultado de la exactitud ha aumentado a pesar de disminuir la tasa de verdaderos positivos, haciendo así que un resultado clasificado como positivo sea más probable que esté correctamente clasificado. Esta tendencia a la alza pasa al contrario con la redes bayesianas de algoritmos de búsqueda *K2* y *Hill Climbing*, los cuales son los únicos que han tenido un descenso en la exactitud, pasando de un 0,94 a un 0,87 en ambos casos.

Tabla 5.5: Tabla comparativa del mejor y peor clasificador en término de *TPR* y *Precision* de los resultados obtenidos por Brezo y nuestro enfoque.

Clasificador	Brezo et ál.		Gaviria et ál.	
	<i>TPR</i>	<i>Exactitud</i>	<i>TRP</i>	<i>Exactitud</i>
<i>BayesNet K2</i>	0,59	0,94	0,99	0,87
<i>Bayes Net Hill Climbing</i>	0,59	0,94	0,99	0,87

Por último, la Tabla 5.6 nos muestra la diferencia en los resultados en términos de área bajo la curva *ROC*. En concreto, se produce una disminución considerable en los valores de *AUC*. En este caso, los clasificadores que anteriormente habían tenido un valor de *AUC* muy superior como son *KNN* con valores de $k = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$ y el clasificador *Random Forest* con valores superiores al 0,90 en este caso llegan a tener valores de 0,69 o 0,71 dependiendo del clasificador, suponiendo una disminución de más de dos puntos sobre su mejor resultado.

A su vez, el clasificador que también disminuye sus resultados pero no en un grado tan alto es *Voted Perceptron*. Los resultados de dicho clasificador solamente disminuyen en 0,07 pasando de un 0,55 a un 0,48 de área bajo la curva *ROC*. Esto nos hace volver a ver la mala selección de este clasificador para este tipo de problemas.

Tabla 5.6: Tabla comparativa del mejor y peor clasificador en término de área bajo la curva *ROC* de los resultados obtenidos por Brezo y nuestro enfoque.

	<i>AUC</i>	
	Brezo et ál.	Gaviria et ál.
<i>RandForest</i>	0,92	0,67
<i>KNN k=1</i>	0,92	0,71
<i>KNN k=2</i>	0,91	0,71
<i>KNN k=3</i>	0,91	0,71
<i>KNN k=4</i>	0,91	0,71
<i>KNN k=5</i>	0,91	0,70
<i>KNN k=6</i>	0,91	0,69
<i>KNN k=7</i>	0,91	0,69
<i>KNN k=8</i>	0,91	0,69
<i>KNN k=9</i>	0,91	0,69
<i>KNN k=10</i>	0,91	0,68
<i>Voted Perceptron</i>	0,55	0,48

5.7 Sumario

En este capítulo, hemos mostrado los resultados obtenidos con los modelos de representación propuestos anteriormente. En primer lugar, hemos realizado una descripción de los protocolos de comunicación. En segundo lugar y ligado al punto anterior hemos realizado un estudio de la técnica de las investigaciones más relevantes en el uso de los protocolos de comunicación y la detección de *malware*.

Seguido, hemos explicado la metodología creada en anteriores publicaciones para la detección de *malware* en ordenadores convencionales. Puesto que este enfoque está pensado para los dispositivos móviles, se ha modificado la metodología para enfocarla a este tipo de terminales. Además en este punto se han explicado los dos conjuntos de datos usados para la experimentación. Después se ha procedido a explicar el proceso de automatización usado para la obtención de la información de los experimentos.

Acto seguido, hemos procedido a definir el problema como un problema de clasificación supervisada. Para ello, los conjuntos de entrenamiento utilizados han

5. ANÁLISIS DEL TRÁFICO DE RED EN LA DETECCIÓN DE MALWARE

sido adecuados etiquetando muestras de tráfico a sabiendas de las aplicaciones maliciosas o benignas.

Después, hemos definido los veintitrés algoritmos de aprendizaje automático utilizados como clasificadores: desde máquinas de soporte vectorial, hasta árboles de decisión, pasando por perceptrones, redes bayesianas o el análisis de los k vecinos más próximos en el espacio n -dimensional analizado. Seguidamente, hemos analizado de antemano cuáles han sido las métricas a emplear para comparar los resultados obtenidos en los diferentes procesos experimentales eligiendo para ello algunas de las más utilizadas en el campo del *machine learning*.

Por último hemos procedido a representar los resultados obtenidos mediante el uso de los clasificadores seleccionados. Dichos resultados han sido comparados con los obtenidos en investigaciones anteriores. Como se ha podido ver, el uso de dicha metodología en los dispositivos móviles ha arrojado resultados bastante inferiores a los enfoques llevados en ordenadores de escritorio.

Una de las posibles razones para dicho descenso es el uso globalizado de envío de paquetes por las aplicaciones móviles. Si tenemos en cuenta que los dispositivos móviles envían una cantidad muy superior de información a través de la red que un ordenador convencional. Debido a este hecho, es más complicada la detección de paquetes maliciosos, puesto que el tráfico generado por una aplicación maliciosa puede ser confundido con el de una aplicación benigna más fácilmente.

Asimismo, las características propias del problema de clasificación de paquetes presentan una novedad adicional con respecto a los problemas de detección tradicionales: la aparición de falsos positivos no es tan relevante como ocurre en otras áreas de conocimiento dado que la escasa carga maliciosa de un paquete en sí misma no es suficiente como para suponer una amenaza real contra el nodo infectado.

*«Si buscas resultados distintos no
hagas siempre lo mismo.»*

Albert Einstein (1879–1955)

CAPÍTULO

6

Conclusiones

A través de los anteriores capítulos hemos descrito y evaluado las contribuciones de la presente tesis doctoral. Ahora es momento de evaluar el marco de estas contribuciones y el grado de consecución de las mismas, tanto en relación a la hipótesis inicial como a los objetivos operacionales y específicos.

Lo que resta de capítulo está organizado como sigue; en la sección 6.1 describiremos de forma detallada las contribuciones principales que se han realizado a lo largo de este trabajo doctoral, así como la evaluación del cumplimiento de los objetivos fijados. En la sección 6.2 haremos una exposición de los pasos siguientes a tener en cuenta para el trabajo futuro que llevaremos a cabo. Por último, en la sección 6.3 expondremos las consideraciones personales extraídas de este trabajo doctoral.

6.1 Contribuciones principales

A lo largo de esta investigación doctoral, hemos realizado una serie de contribuciones sobre los diferentes componentes que la engloban. En esta sección realizaremos una síntesis de las conclusiones más relevantes de la tesis.

6. CONCLUSIONES

Hemos revisado la literatura acerca de la evolución del *malware*. En concreto, hemos comenzado con los conceptos básicos relativos al *malware*: (i) sus tipos, (ii) sus ataques, y por último, (iii) su evolución a lo largo de los años.

Viendo la evolución de la tecnología nos hemos ido centrando en los dispositivos móviles. En dicha evolución hemos comprobado cómo la venta de los dispositivos ha ido aumentando de forma exponencial durante los últimos años. Posteriormente, hemos hecho un estudio de las diferentes muestras de malware aparecidas en los dispositivos móviles. Igualmente, hemos listado las diferencias encontradas entre estos dispositivos y los ordenadores convencionales.

Puesto que en el ámbito de los dispositivos móviles podemos encontrar una multitud de sistemas operativos, nos hemos centrado en el que a día de hoy tiene una mayor cuota de mercado, el sistema *Android*. En un primer momento, hemos visto la arquitectura del sistema, dividida en sus diferentes capas; y por cada una de ellas, hemos analizado las funcionalidades de estas.

A continuación, nos hemos centrado en la seguridad del sistema operativo elegido. Hemos realizado un estudio del mismo y cómo ha sido diseñado desde un primer momento basándose en la seguridad del usuario. Más adelante, hemos resumido todos los aspectos a tener en cuenta en el diseño del sistema operativo, tanto desde el punto de vista *software*, como desde el punto de vista *hardware*.

A su vez, hemos visto la importancia de las máquinas virtuales *Dalvik* y su sistema de *sandboxing* para evitar que una aplicación pueda obtener más privilegios de los que debería. Hemos visto los riesgos de los ataques dentro de nuestro sistema, ofreciendo una visión de los daños que puede causar un *malware* para el usuario, tanto a nivel físico del dispositivo, como a nivel económico.

Una vez explicadas las partes más generales del *malware* en los dispositivos móviles, tanto en comportamiento como los efectos generados en los sistemas, hemos aplicado los dos enfoques principales en entornos de escritorio: (i) el análisis estático y (ii) el análisis dinámico de aplicaciones.

A fin de evaluar ambas técnicas, hemos generado dos conjuntos de datos con aplicaciones benignas (*goodware*) y maliciosas (*malware*). La obtención de estas últimas se hizo a través del conjunto de datos *Drebin Dataset* proporcionado por la *University of Göttingen* [ASH⁺14]. Por otro lado, para la obtención de las muestras de código benigno, hemos procedido a la descarga de 7.062 aplicaciones. Una vez

descargadas las aplicaciones procedimos a la verificación de las mismas a través del servicio *Virus-Total*, herramienta que comprueba las muestras con/contra más de 40 motores de detección de distintas casas de antivirus.

A fin de obtener un conjunto de datos de *software* benigno lo más fidedigno posible, procedimos a eliminar de dicho conjunto todas las muestras que eran detectadas como código malicioso aunque fuera únicamente por un motor antivirus. De esta forma se igualó el conjunto de datos de código malicioso con un total de 5.559 muestras con el de conjunto de datos de código benigno.

Una vez obtenido el conjunto de datos necesarios hicimos uso de las técnicas previamente mencionadas. La primera de las técnicas es la detección estática. En dicho método, hemos utilizado los códigos operacionales (también denominados *opcodes*) como característica principal para la detección. Este método consiste en desensamblar una aplicación para su análisis haciendo uso de herramientas de inteligencia artificial. Para ello, hemos modelado los *opcodes* del conjunto de datos seleccionado (tanto aplicaciones benignas como de aplicaciones malignas), para posteriormente representarlos en un espacio vectorial y determinar la relevancia de cada uno de ellos respecto a su categoría, haciendo uso de la técnica de *Term Frequency - Inverse Document Frequency (TF-IDF)*.

Posteriormente, hemos empleado algoritmos de aprendizaje automático supervisado para generar los modelos que clasifiquen las aplicaciones en maliciosas y benignas. 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*), árboles aleatorios (denominado *Random Tree*) y bosques aleatorios (denominado *Random Forest*), todos ellos con distintas configuraciones, y con el objetivo de determinar cuál es el que ofrece un mejor resultado.

La segunda de las técnicas a utilizar ha sido la detección dinámica. Para ello hemos adaptado una metodología que ya habíamos desarrollado con anterioridad en entornos de escritorio. Esta metodología analiza las tramas de los paquetes de red que generan las aplicaciones. Cabe resaltar, que para llevar a cabo la validación de esta metodología, hemos creado un sistema de automatización de máquinas virtuales para el sistema operativo *Android*.

6. CONCLUSIONES

A continuación, hemos procedido a definir el problema como un problema de clasificación supervisada. Para ello, las muestras de tráfico de los conjuntos de entrenamiento utilizados han sido etiquetados como aplicaciones maliciosas o benignas.

Después, hemos definido los veintitrés algoritmos de aprendizaje automático utilizados como clasificadores: desde máquinas de soporte vectorial, hasta árboles de decisión, pasando por redes neuronales, redes bayesianas o el análisis de los K vecinos más próximos en el espacio n -dimensional analizado. Seguidamente, hemos analizado de antemano cuáles han sido las métricas a emplear para comparar los resultados obtenidos en los diferentes procesos experimentales, y eligiendo para ello algunas de las más utilizadas en el campo del aprendizaje automático supervisado.

Los resultados obtenidos han sido comparados con los obtenidos en investigaciones anteriores. Como se ha podido ver, el uso de dicha metodología en los dispositivos móviles ha arrojado resultados bastante inferiores a los enfoques llevados en ordenadores de escritorio.

Una de las posibles razones para el descenso en la tasa de detección, es el uso globalizado de envío de paquetes por las aplicaciones móviles; debemos tener en cuenta que los dispositivos móviles envían una cantidad muy superior de información a través de la red que un ordenador convencional. Debido a este hecho, es más complicada la detección de paquetes maliciosos, puesto que el tráfico generado por una aplicación maliciosa puede ser confundido con el de una aplicación benigna más fácilmente.

Gracias a estas contribuciones, consideramos cumplidos todos los objetivos operacionales que planteamos en la Sección 1.4, y que recordamos a continuación:

- **Estudio de las diferentes técnicas usadas para la detección de *malware* en dispositivos móviles.** Hemos realizado un estudio de las técnicas usadas para la detección de *malware* en los dispositivos móviles, así como de la evolución que han sufrido estas técnicas hasta nuestros días.
- **Caracterizar la naturaleza de las aplicaciones en base a atributos estáticos.** Hemos analizado nuevas características que faciliten la caracterización de las aplicaciones en entornos móviles.

- **Análisis de las comunicaciones en aplicaciones maliciosas.** Se ha realizado un estudio del tráfico de red generado por distintas muestras de *malware*, a fin de detectar patrones o comportamientos comunes que faciliten su caracterización.
- **Desarrollar nuevas técnicas de representación.** Hemos desarrollado nuevas técnicas de representación de *malware* usando las características tanto estáticas como dinámicas anteriormente obtenidas.
- **Desarrollar nuevas técnicas de detección.** Hemos estudiado nuevas técnicas de detección de *malware* a partir esta representación.

Así mismo, hemos logrado alcanzar los objetivos específicos planteados en la Sección 1.4, que son:

- Analizar aquellos aspectos susceptibles de ser representativos de la intencionalidad de una aplicación móvil.
- Extraer una representación de los códigos operacionales más usados para el comportamiento malicioso en dispositivos móviles para la detección del *malware*.
- Extraer una representación del comportamiento del *malware* en la red, para así poder detectarlo no sólo de forma estática, sino también de forma dinámica.
- Sistematizar la extracción y evaluación de las representaciones a fin de determinar su viabilidad en un entorno fuera del laboratorio de experimentación.

Y como consecuencia de todo ello, consideramos validada la hipótesis fundamental planteada en el Capítulo 1:

«La representación de las aplicaciones móviles obtenida a través de la caracterización tanto de su naturaleza (mediante el análisis estático) como de su comportamiento (mediante el análisis dinámico) permite una detección efectiva de aplicaciones maliciosas.»

6. CONCLUSIONES

6.2 Trabajo futuro

En esta sección vamos a desglosar las líneas de trabajo futuro que hemos identificado en torno a la presente tesis doctoral, siendo éstas: (i) la mejora del método de detección dinámica, (ii) la combinación de características en la detección estática, y (iii) la creación de un sistema híbrido para la detección de *malware*.

6.2.1 Mejora del método de detección dinámica

La primera línea de trabajo futuro se centra en la mejora del método de detección dinámica. Para ello una primera aproximación es el análisis no únicamente de las cabeceras de los paquetes, sino también de sus contenidos.

La problemática a la que nos expondríamos al analizar el contenido del paquete sería relativa a la privacidad del usuario [Son07].

Por otro lado, la detección dinámica se puede orientar a la obtención de las distintas políticas o *policies* que proporciona *Android*. Uno de los posibles enfoques es el uso del tipo de políticas instaladas por la aplicación en el sistema.

6.2.2 Combinación de características en la detección estática

Puesto que la utilización de los códigos operacionales nos ha proporcionado resultados satisfactorios, no hay que dejar de insistir en las técnicas de detección estática como métodos preventivos.

Por un lado, en investigaciones realizadas anteriormente [dIPSGb15] demostramos que el uso de permisos para la detección de *malware* en *Android* es un enfoque con el que se han obtenido buenos resultados, por lo que se podría continuar trabajando en esta línea.

6.2.3 Creación de un sistema híbrido para la detección de *malware*

Así como ya existen investigaciones en sistemas híbridos para la detección de *malware* en sistemas convencionales [HC15], proponemos un sistema que una las dos técnicas hasta ahora estudiadas. Una forma de combinar ambos enfoques es

ponderando cada uno de los análisis realizados, y determinando el valor para cada uno de ellos. Algo destacable es que el análisis estático tiene una mejor efectividad a la hora de detectar las muestras, por lo que el resultado obtenido de este análisis debería ser el que obtuviera un mayor peso.

Para llevar a cabo esta hibridación sería necesario realizar un análisis mucho más exhaustivo de los resultados obtenidos en los experimentos realizados, comprobando qué muestras han sido incorrectamente clasificadas en cada uno de ellos y analizando cuáles han sido los motivos. Habría que estudiar además la complementariedad de los resultados obtenidos.

Un enfoque híbrido provee de una visión más completa del comportamiento de la aplicación, ya que permite observar la app desde dos perspectivas diferentes, combinando en una única clasificación ambos resultados. Esto permite, a priori, elevar la tasa de detección y obtener una mayor precisión a la hora de clasificar las muestras.

Por otra parte, estos enfoques híbridos obligan a generar una infraestructura mucho más compleja. La necesidad de ejecutar las muestras en un entorno controlado provoca, además, un incremento considerable en el tiempo de análisis por muestra.

6.3 Consideraciones finales

A día de hoy, millones de usuarios poseen un terminal dotado de componentes *hardware* muy sofisticados y con una capacidad de cómputo que hace unos pocos años no existía ni en los terminales de sobremesa. Además, alrededor de estas plataformas se están desarrollando ecosistemas de aplicaciones que permiten ampliar la funcionalidad de estos terminales hasta límites aún hoy insospechados.

Las técnicas de detección de *malware* en el sistema *Android* son una de las investigaciones más relevantes en el ámbito de la seguridad. La cantidad ingente de información personal almacenada en estos dispositivos, así como los intentos llevados a cabo por los atacantes para el robo de la misma, hacen que haya que redoblar los esfuerzos en securizar estos sistemas.

Si bien es verdad que en entornos de escritorio hay una percepción de inseguridad mayor, en los dispositivos móviles es totalmente al contrario. Es por ello, que

6. CONCLUSIONES

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 científico, y en virtud del compromiso ético adquirido con la sociedad, es necesario impulsar esta formación de los usuarios ya que como ocurre en muchos otros ámbitos, la mejor cura es la prevención.

«La seguridad es el principal enemigo de los mortales.»

William Shakespeare (1564 – 1616).

«If you are looking for different results, do not do the same thing.»

Albert Einstein (1879–1955)

CAPÍTULO

7

Conclusions

THROUGHOUT this document, the contributions of the PhD thesis have been described and evaluated. Now, it's time to evaluate the framework of these contributions as well as their degree of achievement, both in relation to the initial hypotheses as to the objectives of the PhD thesis.

The rest of the chapter is organized as follows. Section 7.1 describes in detail the main contributions made throughout this PhD thesis, as well as the assessment of the fulfilment of the objectives previously set.

7.1 Summary of results

Throughout this PhD research, we have performed a series of validations on the different components that comprise this research. In this section we will make a summary of the most important conclusions of the thesis.

We reviewed the literature on the evolution of *malware*. In particular, we began by reviewing basic concepts relating to *malware*: its types, attacks and evolution over the years. Watching the evolution of technology, we focused on mobile devices. In this evolution we noticed how the sale of devices has increased exponentially over the last few years. Then, we made a study of the different samples of *malware*

7. CONCLUSIONS

appearing on mobile devices. We also listed the differences found between these devices and conventional computers.

As on mobile devices we can find a multitude of operating systems created for them, we conducted a study to focus on the one with the largest market share, the *Android* system. First of all, we watched the system architecture, divided into different layers. In each one, their tasks were analyzed.

We focused on the security of *Android*. We explained the operating system and how it has been designed from the outset taking into account the safety of the user. Later, we summarized all the aspects relevant for the design of the operating system, both from the *software* and from the *hardware* point of view.

We noticed the importance of the *Dalvik* virtual machines and their sandboxing system to prevent an application from getting more privileges than it should. We saw the risks of attacks within our system, offering a vision of the damage that *malware* can cause to the user, both damaging the device itself, as well as representing an economic loss. The ability of these samples to monetize the damage caused is one of the keys when it comes to analyze them.

Once we explained the broader parts of *malware* on mobile devices, both how it works and the effects generated in systems, we made a division of the two approaches carried out for the detection of *malware* over the years: the static analysis and the dynamic analysis of applications.

In order to detect *malware* on the platform, we used techniques of static analysis and dynamic analysis of the applications. Since both techniques will make use of a data set in a first step, we generated two data sets with benign applications (goodware) and malicious applications (*malware*). The latter were obtained through Drebin Dataset, provided by the University of Göttingen [ASH + 14]. However, to obtain samples of benign code, we proceeded to download 7,062 applications. Once the applications were downloaded, we moved on to verify them through the Virus-Total service, a tool that checks the samples using more than 40 scanning engines from different antivirus companies.

In order to obtain the most reliable possible dataset, we proceeded to remove from this set all the samples detected as malicious code, even if only by one of the antivirus engines. Thus the data set of malicious code, with a total of 5,559 samples, equaled the data set of benign code.

After obtaining the necessary datasets, we made use of the aforementioned techniques. The first technique is the static detection. In this method, we used operational codes as a feature for the detection. Using this method, we obtain the disassembly of an application so it can be analysed by artificial intelligence tools. This requires modelling operational codes, both in benign and malicious applications.

For this purpose, we have used machine learning algorithms so that, using these attributes, models that classify applications as malicious or not can be built. Specifically, we used the implementations present in the WEKA tool from the following algorithms: logistic regression, Bayesian Networks, Support Vector Machines (called SVM), K-Nearest Neighbor (called IBK), decision trees C.45 (called J.48), random trees and random forests, all of them having different configurations, and in order to determine which one offers a better result.

The second technique was the dynamic detection. As an important point in this issue we proceeded to explain the methodology we followed in previous research. Since in this part we made use of network traffic, we created a virtual machine automation system for *Android*.

We proceeded to define the problem as a problem of supervised classification. To do this, the training sets used were adequated by labelling traffic samples knowing whether applications were malicious or benign.

Then, we defined the twenty-three machine learning algorithms used as classifiers: from support vector machines to decision trees, including perceptrons, Bayesian networks or analysis of K-nearest neighbors in the n-dimensional space analyzed. Next, we analyzed in advance which metrics to use in order to compare the results of different experimental processes, choosing for this purpose some of the most commonly used in the field of machine learning.

The results were compared with those obtained in previous research. As it can be seen, the use of this methodology on mobile devices produced inferior results than the approaches taken in desktop computers.

One of the possible reasons for the decline in the detection is the globalized use of packet sending by mobile applications. We must consider that mobile devices send a much higher amount of information through the network than a conventional computer. Due to this fact, it is more difficult to detect malicious packets, since the

7. CONCLUSIONS

traffic generated by a malicious application can be more easily confused with a benign application.

Thanks to these contributions, we believe we have fulfilled all the operational objectives set in Section 1.4, which we recall below:

- **Study of the different techniques used to detect *malware* on mobile devices.** There was a study of the techniques used for detecting *malware* on mobile devices, as well as the evolution these techniques have undergone to this day.
- **Characterizing the nature of the applications on the basis of static attributes.** New features that facilitate the characterization of applications in mobile environments were analyzed.
- **Analysis of communications in malicious applications.** A study of network traffic generated by different *malware* samples was made in order to detect patterns or common behaviors that facilitate their characterization.
- **Developing new techniques of representation.** New techniques of representation of *malware* were developed using both static and dynamic characteristics previously obtained.
- **Developing new detection techniques.** New *malware* detection techniques were studied from this representation.

We have also managed to achieve the specific objectives set in Section 1.4, which are:

- Analyzing those aspects likely to be representative of the intentionality of a mobile application.
- Extracting a representation of the most commonly used operational codes for malicious behavior on mobile devices for the detection of *malware*.
- Extracting a representation of the behavior of *malware* on the network, in order to be able to detect it not only statically but also dynamically.

- Systematizing the extraction and evaluation of representations in order to determine their viability in an environment outside the experimentation laboratory.

And as a result of the above, we consider validated the fundamental hypothesis raised in Chapter 1:

«The representation of mobile applications obtained through the characterization of both their nature and their behavior allows effective detection of malicious applications.»

7.2 Future Work

In this section we will break down the lines for future work we have identified around this doctoral thesis, which are: (i) improvement of the dynamic detection method, (ii) combination of features in static detection, (iii) creation of a hybrid system for the detection of *malware*.

7.2.1 Improvement of the dynamic detection method

The first line of future work focuses on the improvement of the dynamic detection method. Our first approach will be the analysis not only of packet headers but also of its contents.

The problems we would be likely to face when analyzing the contents of the packet would be related to user privacy [Son07]. Thus, we should consider how to use such content without assaulting their privacy.

On the other hand, dynamic detection can be directed to obtaining the different policies provided by *Android*. One possible approach is the use of the kind of policies installed on the system by the application.

7.2.2 Combination of features in static detection

Since the use of operational codes has provided us with satisfactory results, we should insist on the use of static detection techniques as preventive methods.

7. CONCLUSIONS

On the one hand, Gaviria et al. [DIPSG15] demonstrated that the use of permits for *malware* detection in *Android* is a supportive approach; therefore, it could be continued by uniting the different features used by the scientific community to do so.

7.2.3 Creating a hybrid system for the detection of *malware*

As it has already been research on hybrid systems for *malware* detection in conventional systems [HC15], a system that links the two techniques so far studied is proposed, a system that links the two techniques so far studied is proposed. One way to combine the two approaches is pondering each of the analyses, determining the value of each of the weights empirically. We have seen that the static analysis has a better effectiveness in detecting samples, so that the result of this analysis should be the one that has a higher weight.

To accomplish this hybridization would be necessary to conduct a more thorough analysis of the results obtained in the experiments, ensuring which samples have been incorrectly classified and analyzing what have been the reasons. The complementarity of the results should also be studied.

A hybrid approach provides a more complete view of the application behavior, allowing to analyse the app from two different perspectives, combining in a single classification both results. This allows, in principle, to increase the rate of detection and to obtain a better accuracy when it comes to classifying samples.

Moreover, these hybrids approaches require generating a more complex infrastructure. The need to run samples in a controlled environment also causes a considerable increase in the analysis time for each sample.

7.3 Final considerations

Nowadays, millions of users possess a terminal equipped with very sophisticated *hardware* components and with a calculation capacity which only a few years ago did not exist even in desktop terminals. Besides, around these platforms applications ecosystem are being developed that allow to extend the functionality of these

terminals to limits still unsuspected. Techniques for detecting *malware* in the *Android* system are one of the most important research in the field of security. The vast amount of personal information stored on these devices, as well as the attempts made by attackers to steal it, makes it necessary to double our efforts to securize these systems.

The perceived lack of security developed in recent years in the desktop environment is still nonexistent in the environment of the mobile terminal. That is why, from the scientific community and the specialized industry, we must join forces in the development of tools that will reduce the gap between the perception of security and the actual situation.

However, these tools will never be efficient if the people using them are not aware of the danger. From the university field, considering our ethical commitment with society, it is necessary to promote the training of users in order to be able to enjoy the earthquake of information that we face today.

«Security is the chief enemy of mortals.»

William Shakespeare (1564 – 1616).

Publicaciones

En este apartado se recogen aquellas publicaciones académicas y técnicas que han dado lugar al presente trabajo doctoral divididas en dos apartados: las relacionadas de forma directa con la temática principal de la tesis y aquellas indirectamente relacionadas con el trabajo presentado pero que lo tocan transversalmente.

Publicaciones directamente relacionadas

Este apartado incluye todas aquellas publicaciones que guardan una especial relación con los contenidos presentados en esta tesis doctoral por el contenido, la aplicación de sus metodologías o la base procedimental aplicada.

Revistas y capítulos de libro

- Brezo, Félix; Gaviria de la Puerta, José; Ugarte-Pedrero, Xabier; Santos, Igor; Barroso, David y G. Bringas, Pablo (2013). *A Supervised Classification Approach for Detecting Packets Originated in a HTTP-based Botnet*. En *CLEI Electronic Journal*, v. 17, n. 3, paper 02, diciembre de 2013.

Comunicaciones en conferencias internacionales y seminarios

- Gaviria de la Puerta, José; Sanz, Borja; Santos, Igor; García Bringas, Pablo (2015). *Using Dalvik Opcodes for Malware Detection on Android*. En *Hybrid Artificial Intelligent Systems* (pp. 416-426). Springer. Junio 22-24, 2015.

7. CONCLUSIONS

- Gaviria de la Puerta, José; Sanz, Borja; Santos, Igor; García Bringas, Pablo (2015). *The Evolution of Permission as Feature for Android Malware Detection*. En *International Joint Conference* (pp. 389-400). Springer. Junio 15-17, 2015 (ERA Rank: B).
- Brezo, Félix; Gaviria de la Puerta, José; Ugarte-Pedrero, Xabier; Santos, Igor; Barroso, David y García Bringas, Pablo (2012). *Clasificación supervisada de paquetes procedentes de una Botnet HTTP*. In *Informatica (CLEI), 2012 XXXVIII Conferencia Latinoamericana* En (pp. 1-8). IEEE. Octubre 1-5, 2012
- Brezo, Félix; Gaviria de la Puerta, José y García Bringas, Pablo (2013). *Detecting Command & Control Channels of a Botnet using a N-packet-based Approach*. En *Third International Conference on Advances in Information Mining and Management IMMM 2013*. Noviembre 17-22, 2013.
- Gaviria de la Puerta, José; Sanz, Borja; y García Bringas, Pablo (2014). *Un nuevo enfoque para la detección de malware en Android a través del análisis del comportamiento*. En *CyberCamp*. Diciembre 5-7, 2014.

Publicaciones indirectamente relacionadas

Este apartado incluye aquellas publicaciones que recorren aspectos vinculados a los tratados en la tesis doctoral o que tocan áreas temáticas no directamente relacionadas con el propósito general de la tesis pero que, bien conceptual o bien metodológicamente, lo soportan.

- Brezo, Félix; Gaviria de la Puerta, José; Santos, Igor; Barroso, David y García Bringas, Pablo (2012). *C&C Techniques in Botnet Development*. En *CISIS 2012* (ERA Rank: B).
- Galán-García, Patxi; Gaviria de la Puerta, José; Laorden Gómez, Carlos; Santos, Igor; García Bringas, Pablo (2013). *Supervised Machine Learning for the Detection of Troll Profiles in Twitter Social Network: Application to a Real Case of Cyberbullying*. En *International Joint Conference SOCO'13-CISIS'13-ICEUTE'13* (pp. 419-428). Springer. Septiembre 11-13, 2013.

- Salazar, Mike; Gaviria de la Puerta, José; Laorden Gómez, Carlos; García Bringas, Pablo (2013). Enhancing cybersecurity learning through an augmented reality-based serious game. In Global Engineering Education Conference (EDUCON), 2013 IEEE (pp. 602-607). IEEE.

Bibliografía

- [ABC⁺03] Parixit Aghera, Alan Bok, Suresh Chintada, Sudharshana Rao, and Antonella Rinaldi. Over the air mobile device software management, August 29 2003. US Patent App. 10/652,352. 35
- [Abr13] Lawrence Abrams. Cryptolocker ransomware information guide and faq. *Viitattu*, 8:2014, 2013. 46
- [ACS09] Frank Ableson, Charlie Collins, and Robi Sen. *Unlocking Android*. Manning Greenwich, 2009. 37
- [All07] Open Handset Alliance. Industry leaders announce open platform for mobile devices. *Press release*, 2007. 2
- [All11] Open Handset Alliance. Open handset alliance. *Retrieved August*, 26:2011, 2011. 2
- [ASH⁺14] Daniel Arp, Michael Spreitzenbarth, Malte Hübner, Hugo Gascon, Konrad Rieck, and CERT Siemens. Drebin: Effective and explainable detection of android malware in your pocket. In *Proc. of NDSS*, 2014. 11, 66, 81, 110
- [ATS14] Shahid Alam, Issa Traore, and Ibrahim Sogukpinar. Current trends and the future of metamorphic malware detection. In *Proceedings of the 7th International Conference on Security of Information and Networks*, page 411. ACM, 2014. 6

BIBLIOGRAFÍA

- [Aus12] Ian Austen. Blackberry 10 prototype is given to developers. *New York Times*, 2012. 4
- [AW99] Shun-ichi Amari and Si Wu. Improving support vector machine classifiers by modifying kernel functions. *Neural Networks*, 12(6):783–789, 1999. 69, 96
- [BBS⁺10] Thomas Blasing, Leonid Batyuk, A-D Schmidt, Seyit Ahmet Camtepe, and Sahin Albayrak. An android application sandbox system for suspicious software detection. In *Malicious and unwanted software (MALWARE), 2010 5th international conference on*, pages 55–62. IEEE, 2010. 53
- [BDD⁺01] J. Bergeron, M. Debbabi, J. Desharnais, M.M. Erhioui, Y. Lavoie, and N. Tawbi. Static detection of malicious code in executable programs. *Int. J. of Req. Eng*, 2001:184–189, 2001. 48
- [BdlPB12] Félix Brezo, José Gaviria de la Puerta, and David Barroso. Brianna: Botnet detection relying on an intelligent analysis of network architecture. Master’s thesis, University de Deusto, España, 2012. 83, 94, 101
- [BdlPUP⁺13] Félix Brezo, José Gaviria de la Puerta, Xabier Ugarte-Pedrero, Igor Santos, and Pablo G Bringas. A supervised classification approach for detecting packets originated in a http-based botnet. *CLEI Electronic Journal*, 16(3):2–2, 2013. 67, 95, 104
- [Bec10] Michael Becher. Security of smartphones at the dawn of their ubiquitousness. 2010. 27, 41, 42, 43
- [Ber07] Justin Berka. Turbo sim add-on allows full iphone unlocking, 2007. 43
- [BFH⁺11] Michael Becher, Felix C Freiling, Johannes Hoffmann, Thorsten Holz, Sebastian Uellenbeck, and Christopher Wolf. Mobile security catching up? revealing the nuts and bolts of the security of

- mobile devices. In *Security and Privacy (SP), 2011 IEEE Symposium on*, pages 96–111. IEEE, 2011. 16, 20, 23
- [BHSP08] A. Bose, X. Hu, K.G. Shin, and T. Park. Behavioral detection of malware on mobile handsets. In *Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 225–238. ACM, 2008. 47, 55
- [BI09] Yuval Ben-Itzhak. Organised cybercrime and payment cards. *Card Technology Today*, 21(2):10–11, 2009. 5
- [Bis06] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006. 68
- [BLA04] BLADOX. Turbo sim. *Press Release*, December 2004. 43
- [BNC⁺08] Timothy K Buennemeyer, Theresa M Nelson, Lee M Clagett, John Paul Dunning, Randy C Marchany, and Joseph G Tront. Mobile device profiling and intrusion detection using smart batteries. In *Hawaii International Conference on System Sciences, Proceedings of the 41st Annual*, pages 296–296. IEEE, 2008. 54, 57, 62, 64
- [BOB⁺10] Jeffrey Bickford, Ryan O’Hare, Arati Baliga, Vinod Ganapathy, and Liviu Iftode. Rootkits on smart phones: attacks, implications and opportunities. In *Proceedings of the eleventh workshop on mobile computing systems & applications*, pages 49–54. ACM, 2010. 46
- [BP63] Mr. Bayes and Mr 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)*, pages 370–418, 1763. 72
- [Bra10] Stefan Brahler. Analysis of the android architecture. *Karlsruhe institute for technology*, 2010. 33

BIBLIOGRAFÍA

- [Bre96a] Leo Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996. 97, 98
- [Bre96b] Leo Breiman. Out-of-bag estimation. Technical report, Citeseer, 1996. 97
- [Bre01] Leo Breiman. Random forests. *Machine Learning*, 45:5–32, 2001. 10.1023/A:1010933404324. 71
- [Bre14] Félix Brezo. *Detección de tráfico de control de botnets modelizando el flujo de los paquetes de red*. PhD thesis, University de Deusto, Febrero 2014. 93, 101, 104
- [BTA⁺06] Laurent Bernaille, Renata Teixeira, Ismael Akodkenou, Augustin Soule, and Kave Salamatian. Traffic classification on the fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006. 92
- [CAJ07] Mihai Christodorescu and Somesh Adviser-Jha. *Behavior-based malware detection*. University of Wisconsin at Madison, 2007. 12, 13
- [Cas97] Enrique Castillo. *Expert systems and probabilistic network models*. Springer Science & Business Media, 1997. 73
- [CBS05] A. Chaturvedi, S. Bhatkar, and R. Sekar. Improving attack detection in host-based ids by learning properties of system call arguments. In *In Proceedings of the IEEE Symposium on Security and Privacy*. Citeseer, 2005. 56
- [CDGS07] Manuel Crotti, Maurizio Dusi, Francesco Gringoli, and Luca Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):5–16, 2007. 93

- [CFGW11] Erika Chin, Adrienne Porter Felt, Kate Greenwood, and David Wagner. Analyzing inter-application communication in android. In *Proceedings of the 9th international conference on Mobile systems, applications, and services*, pages 239–252. ACM, 2011. 41, 51
- [CG05] Lorrie Faith Cranor and Simson Garfinkel. *Security and usability: designing secure systems that people can use*. O'Reilly Media, Inc., 2005. 47
- [CH91] Gregory F Cooper and Edward Herskovits. A bayesian method for constructing bayesian belief networks from databases. In *Proceedings of the Seventh conference on Uncertainty in Artificial Intelligence*, pages 86–94. Morgan Kaufmann Publishers Inc., 1991. 73, 98
- [CHC⁺10] Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and testing low-degree polynomial data mappings via linear svm. *The Journal of Machine Learning Research*, 11:1471–1490, 2010. 69, 96
- [Chi05] E Chien. Security response: Symbos. mabir. *Symantec Corporation*, 10:108, 2005. 45
- [CI05] Vinton G Cerf and Robert E Icahn. A protocol for packet network intercommunication. *ACM SIGCOMM Computer Communication Review*, 35(2):71–82, 2005. 87
- [CJ06] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. Technical report, DTIC Document, 2006. 48
- [Cla94] Kimberly C Claffy. *Internet traffic characterization*. PhD thesis, University of California, San Diego, 1994. 91
- [Coh84] Fred Cohen. Computer viruses: theory and experiments. *Computers & security*, 1984. 12

BIBLIOGRAFÍA

- [Coo96] Douglas E Coomer. Tcp/ip principios básicos, protocolos y arquitectura. *Ed. Prentice Hall*, 1996. 84
- [CU13] Victor Chebyshev and Roman Unuchek. Mobile malware evolution: 2013. *Kaspersky Lab ZAO*, 2013. 29
- [CWYL07] J. Cheng, S.H.Y. Wong, H. Yang, and S. Lu. Smartsiren: virus detection and alert for smartphones. In *Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 258–271. ACM, 2007. 47, 53
- [DCP⁺08] Luis M De Campos, JM Puerta, et al. Learning bayesian networks by ant colony optimisation: searching in two different spaces. *Mathware & soft computing*, 9(3):251–268, 2008. 99
- [Ded04] Douglas Dedo. Windows mobile-based devices and security: Protecting sensitive business information. *Microsoft Corporation April*, 2004. 17
- [Dev07] Apple Developer. ios dev center, 2007. 3
- [Dev11] Android Developers. What is android? *ht tp://developer. android. com/guide/basics/what-is-android. html*, 2, 2011. 17, 30, 37
- [DF03] Sandrine Dudoit and Jane Fridlyand. Bagging to improve the accuracy of a clustering procedure. *Bioinformatics*, 19(9):1090–1099, 2003. 98
- [dlPSGB15] José Gaviria de la Puerta, Borja Sanz, Igor Santos Grueiro, and Pablo García Bringas. The evolution of permission as feature for android malware detection. In *International Joint Conference*, pages 389–400. Springer, 2015. 114
- [DP07] Mila Dalla Preda. Code obfuscation and malware detection by abstract interpretation. *PhD diss.*, *http://profs. sci. univr. it/dallaPreda/MilaDallaPreda_Phd. pdf*, 2007. 13

- [DR02] Joan Daemen and Vincent Rijmen. *The design of Rijndael: AES—the advanced encryption standard*. Springer Science & Business Media, 2002. 37
- [DSK11] Goran Delac, Marin Silic, and Jakov Krolo. Emerging security threats for mobile platforms. In *MIPRO, 2011 Proceedings of the 34th International Convention*, pages 1468–1473. IEEE, 2011. 12
- [DWF03] Christian Dewes, Arne Wichmann, and Anja Feldmann. An analysis of internet chat systems. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 51–64. ACM, 2003. 91
- [EGH⁺14] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)*, 32(2):5, 2014. 52
- [EMA07] Jeffrey Erman, Anirban Mahanti, and Martin Arlitt. Byte me: a case for byte accuracy in traffic classification. In *Proceedings of the 3rd annual ACM workshop on Mining network data*, pages 35–38. ACM, 2007. 93
- [EMAW07] Jeffrey Erman, Anirban Mahanti, Martin Arlitt, and Carey Williamson. Identifying and discriminating between web and peer-to-peer traffic in the network core. In *Proceedings of the 16th international conference on World Wide Web*, pages 883–892. ACM, 2007. 92
- [Emm06] David Emm. Mobile malware—new avenues. *Network Security*, 2006(11):4–6, 2006. 45
- [Emm14] David Emm. Kaspersky security bulletin 2014. malware evolution, 2014. 6, 27, 28, 29

BIBLIOGRAFÍA

- [EOM09a] William Enck, Machigar Ongtang, and Patrick McDaniel. On lightweight mobile phone application certification. In *Proceedings of the 16th ACM conference on Computer and communications security*, pages 235–245. ACM, 2009. 52
- [EOM09b] William Enck, Machigar Ongtang, and Patrick McDaniel. Understanding android security. *IEEE security & privacy*, (1):50–57, 2009. 32, 33, 41
- [EOMC11] William Enck, Damien Ocateau, Patrick McDaniel, and Swarat Chaudhuri. A study of android application security. In *USENIX security symposium*, volume 2, page 2, 2011. 62, 64
- [Erb04] Michael Erbschloe. *Trojans, worms, and spyware: a computer security professional's guide to malicious code*. Butterworth-Heinemann, 2004. 12, 13
- [ESKK06] M. Egele, M. Szydlowski, E. Kirda, and C. Kruegel. Using static program analysis to aid intrusion detection. *Detection of Intrusions and Malware & Vulnerability Assessment*, pages 17–36, 2006. 48
- [ETMLP05] William Enck, Patrick Traynor, Patrick McDaniel, and Thomas La Porta. Exploiting open functionality in sms-capable cellular networks. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 393–404. ACM, 2005. 44
- [FBJS12] Elli Fragkaki, Lujo Bauer, Limin Jia, and David Swasey. Modeling and enhancing android's permission system. In *Computer Security—ESORICS 2012*, pages 1–18. Springer, 2012. 62, 63
- [FBKF12] Rafael Fedler, Christian Banse, Christoph Krauss, and Volker Fusenig. Android os security: Risks and limitations. *Fraunhofer Research Institution for Applied and Integrated Security*, 2012. 36

- [FBL⁺14] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, M Gaur, Mauro Conti, and R Muttukrishnan. Android security: A survey of issues, malware penetration and defenses. 2014. 32
- [FCF09] Adam P Fuchs, Avik Chaudhuri, and Jeffrey S Foster. Scandroid: Automated security certification of android applications. *Manuscript, Univ. of Maryland*, <http://www.cs.umd.edu/avik/projects/scandroidascaa>, 2(3), 2009. 51
- [FCH⁺11] Adrienne Porter Felt, Erika Chin, Steve Hanna, Dawn Song, and David Wagner. Android permissions demystified. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 627–638. ACM, 2011. 52
- [FFC⁺11] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 3–14. ACM, 2011. 29
- [FG13] Christian Funk and Maria Garnaeva. Kaspersky security bulletin 2013. overall statistics for 2013. *Securelist*, [online] Retrieved on December, 2013. 28
- [FGG97] Nir Friedman, Dan Geiger, and Moises Goldszmidt. Bayesian network classifiers. *Machine learning*, 29(2-3):131–163, 1997. 73, 74, 98, 99
- [FHJ52] Evelyn Fix and Joseph L Hodges Jr. Discriminatory analysis-nonparametric discrimination: Small sample performance. Technical report, DTIC Document, 1952. 74
- [FHM08] Felix C Freiling, Thorsten Holz, and Martin Mink. Reconstructing people’s lives: A case study in teaching forensic computing. In *IMF*, pages 125–142. Citeseer, 2008. 43

BIBLIOGRAFÍA

- [FHSL96] S. Forrest, S.A. Hofmeyr, A. Somayaji, and T.A. Longstaff. A sense of self for unix processes. In *Security and Privacy, 1996. Proceedings., 1996 IEEE Symposium on*, pages 120–128. IEEE, 1996. 53
- [FJK06] Steven M Furnell, Adila Jusoh, and Dimitris Katsabas. The challenges of understanding and using security: A survey of end-users. *Computers & Security*, 25(1):27–35, 2006. 47
- [Föl73] Hans Föllmer. On entropy and information gain in random fields. *Probability Theory and Related Fields*, 26(3):207–217, 1973. 101
- [FS04] F-Secure. Bluetooth-worm:symbos/cabir, 2004. 45
- [FS05] F-Secure. Backdoor, 2005. 13
- [fSEC94] International Organization for Standardization/International Electrotechnical Commission. Information technology—open systems interconnection—basic reference model: The basic model. *ISO/IEC*, pages 7498–1, 1994. 85
- [Fur05] Steven Furnell. Why users cannot use security. *Computers & Security*, 24(4):274–279, 2005. 47
- [Fur07] Steven Furnell. Making security usable: Are things improving? *Computers & Security*, 26(6):434–443, 2007. 47
- [FW11] Adrienne Porter Felt and David Wagner. *Phishing on mobile devices*. na, 2011. 22
- [G⁺95] Stephen R Garner et al. Weka: The waikato environment for knowledge analysis. In *Proceedings of the New Zealand computer science research students conference*, pages 57–64. Citeseer, 1995. 70
- [Gar05] Simson Garfinkel. *Design principles and patterns for computer systems that are simultaneously secure and usable*. PhD thesis, Massachusetts Institute of Technology, 2005. 47

- [GG14] Sumit Garg and Vishal Gaikar. Experience new windows 10 with its new features. 2014. 4
- [GMZ14] Olga Gadyatskaya, Fabio Massacci, and Yury Zhauniarovich. Security in the firefox os and tizen mobile platforms. *Computer*, (6):57–63, 2014. 5
- [Gos06] Alexander Gostev. Mobile malware evolution. <http://securelist.com/analysis/malware-evolution-monthly/36109/mobile-malware-evolution-an-overview-part-1/>, 2006. 28, 29
- [Gre6] DM Green. Swets, ja (1966). signal detection theory and psychophysics. *Psychological Bulletin*, 75:424–429, 6. 101
- [GS11] Nisarg Gandhewar and Rahila Sheikh. Google android: An emerging software platform for mobile devices. *International Journal on Computer Science & Engineering*, 2011. 3, 31
- [GWZ04] Chuanxiong Guo, Helen J Wang, and Wenwu Zhu. Smart-phone attacks and defenses. In *HotNets III*, 2004. 19
- [Har03] Richard Harrison. *Symbian Os C++ For Mobile Phones (W/Cd)*. John Wiley & Sons, 2003. 17
- [HB06] Greg Hoglund and James Butler. *Rootkits: subverting the Windows kernel*. Addison-Wesley Professional, 2006. 13
- [HC15] Adrian Herrera and Ben Cheney. Jmd: A hybrid approach for detecting java malware. *Information Security 2015*, 27:3, 2015. 114
- [HHJ⁺11] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. These aren't the droids you're looking for: retrofitting android to protect data from imperious applications. In *Proceedings of the 18th ACM conference on Computer and communications security*, pages 639–652. ACM, 2011. 41

BIBLIOGRAFÍA

- [HS10] Cedric Halbronn and Jean Sigwald. iphone security model & vulnerabilities. In *Proceedings of Hack in the box sec-conference. Kuala Lumpur, Malaysia*, 2010. 17
- [HSSW05] Patrick Haffner, Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Acas: automated construction of application signatures. In *Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data*, pages 197–202. ACM, 2005. 93
- [Hyp10] M Hypponen. Mobile security review september 2010. *F-Secure Labs, HelsinkiFinland, Tech. Rep*, 2010. 29
- [IDC13] IDC. Android and ios combine for 92.3 % of all smartphone operating system shipments in the first quarter while windows phone leapfrogs blackberry, according to idc, 2013. <http://www.idc.com/getdoc.jsp?containerId=prUS24108913>. 63
- [JDI04] Grant A Jacoby and Nathaniel J Davis IV. Battery-based intrusion detection. In *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, volume 4, pages 2250–2255. IEEE, 2004. 53, 54, 57, 62, 64
- [JMD06] G.A. Jacoby, R. Marchany, and N.J. DAVIS. How mobile host batteries can improve network security. *IEEE security & privacy*, 4(5):40–49, 2006. 58
- [JWCY07] Liangxiao Jiang, Dianhong Wang, Zhihua Cai, and Xuesong Yan. Survey of improving naive bayes for classification. In Reda Alhajj, Hong Gao, Xue Li, Jianzhong Li, and Osmar Zaïane, editors, *Advanced Data Mining and Applications*, volume 4632 of *Lecture Notes in Computer Science*, pages 134–145. Springer Berlin / Heidelberg, 2007. 10.1007/978-3-540-73871-8_14. 73
- [JZEC04] Jazilah Jamaluddin, Nikoletta Zotou, Reuben Edwards, and Paul Coulton. Mobile phone vulnerabilities: a new generation of

- malware. In *Consumer Electronics, 2004 IEEE International Symposium on*, pages 199–202. IEEE, 2004. 21
- [K⁺96] EM Kleinberg et al. An overtraining-resistant stochastic modeling method for pattern recognition. *The annals of statistics*, 24(6):2319–2349, 1996. 71
- [KBB⁺04] Thomas Karagiannis, Andre Broido, Nevil Brownlee, Kimberly Claffy, and Michalis Faloutsos. Is p2p dying or just hiding?[p2p traffic measurement]. In *Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE*, volume 3, pages 1532–1538. IEEE, 2004. 90
- [KFH05] D.K. Kang, D. Fuller, and V. Honavar. Learning classifiers for misuse and anomaly detection using a bag of system calls representation. In *Information Assurance Workshop, 2005. IAW'05. Proceedings from the Sixth Annual IEEE SMC*, pages 118–125. IEEE, 2005. 55
- [KKB⁺06] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. Kemmerer. Behavior-based spyware detection. In *Usenix Security Symposium*, 2006. 56
- [KNZG10] Ram Keralapura, Antonio Nucci, Zhi-Li Zhang, and Lixin Gao. Profiling users in a 3g network using hourglass co-clustering. In *Proceedings of the sixteenth annual international conference on Mobile computing and networking*, pages 341–352. ACM, 2010. 92
- [KRV04] C. Kruegel, W. Robertson, and G. Vigna. Detecting kernel-level rootkits through binary analysis. In *Computer Security Applications Conference, 2004. 20th Annual*, pages 91–100. IEEE, 2004. 49

BIBLIOGRAFÍA

- [KRVV04] C. Kruegel, W. Robertson, F. Valeur, and G. Vigna. Static disassembly of obfuscated binaries. In *Proceedings of the 13th conference on USENIX Security Symposium*, volume 13, page 18, 2004. 50
- [Ksh10] Nir Kshetri. *The global cybercrime industry: economic, institutional and strategic perspectives*. Springer Science & Business Media, 2010. 5
- [KSS08] H. Kim, J. Smith, and K.G. Shin. Detecting energy-greedy anomalies and mobile malware variants. In *Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 239–252. ACM, 2008. 57, 58
- [Kup05] Peter Kuper. The state of security. *Security & Privacy, IEEE*, 3(5):51–53, 2005. 47
- [KZP07] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques, 2007. 67, 95
- [Lab05] McAfee Labs. Locknut, 2005. 45
- [Lab10] Kapersky Lab. First sms trojan detected for smartphones running android, 2010. 29
- [LABC03] Tanja Lang, Grenville Armitage, Phillip Branch, and Hwan-Yi Choo. A synthetic traffic model for half-life. In *Australian Telecommunications Networks & Applications Conference*, volume 2003, 2003. 91
- [LBA04] Tanja Lang, Philip Branch, and Grenville Armitage. A synthetic traffic model for quake3. In *Proceedings of the 2004 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 233–238. ACM, 2004. 91

- [LBV05] Z. Liu, S.M. Bridges, and R.B. Vaughn. Combining static analysis and dynamic learning to build accurate intrusion detection models. In *Information Assurance, 2005. Proceedings. Third IEEE International Workshop on*, pages 164–177. IEEE, 2005. 50
- [Lea05] N. Leavitt. Mobile phones: the next frontier for hackers? *Computer*, 38(4):20–23, 2005. 45, 46, 56
- [Lew98] David D Lewis. Naive (bayes) at forty: The independence assumption in information retrieval. In *Machine learning: ECML-98*, pages 4–15. Springer, 1998. 73, 98
- [Lim09] OMTP Limited. Advanced trusted environment: Advanced trusted environment: Omtpt r1. Technical report, OMTP Limited, 2009. 43
- [LJVR08] Jorge M Lobo, Alberto Jiménez-Valverde, and Raimundo Real. Auc: a misleading measure of the performance of predictive distribution models. *Global ecology and Biogeography*, 17(2):145–151, 2008. 101
- [LJZ09] Chao Li, Wei Jiang, and Xin Zou. Botnet: Survey and case study. In *Innovative Computing, Information and Control (ICI-CIC), 2009 Fourth International Conference on*, pages 1184–1187. IEEE, 2009. 13
- [LPMS13] Mariantonietta La Polla, Fabio Martinelli, and Daniele Sgandurra. A survey on security for mobile devices. *Communications Surveys & Tutorials, IEEE*, 15(1):446–471, 2013. 27, 29
- [LRS05] Evelina Lamma, Fabrizio Riguzzi, and Sergio Storari. Improving the k2 algorithm using association rule parameters. *Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU04)*, pages 1667–1674, 2005. 74, 98

BIBLIOGRAFÍA

- [Mas11] Denis Maslennikov. Mobile malware evolution: An overview, part 4. *Disponibile a https://www.securelist.com/en/analysis/204792168/Mobile_Malware_Evolution_An_Overview_Part_4*, 2011. 29
- [McM06] Robert McMillan. New redbrowser trojan first to target j2me. *Computerworld*, February, 26, 2006. 45
- [McQ08] Samuel C McQuade. *Encyclopedia of cybercrime*. Greenwood Press, 2008. 6
- [MH08] Patricia Mayer Milligan and Donna Hutcheson. Business risks and security assessment for mobile devices. *Information Systems Control Journal*, 1:24, 2008. 20
- [MHH06] M. Miettinen, P. Halonen, and K. Hatonen. Host-based intrusion detection for advanced mobile devices. In *Advanced Information Networking and Applications, 2006. AINA 2006. 20th International Conference on*, volume 2, pages 72–76. Ieee, 2006. 47, 56
- [MHLB04] Anthony McGregor, Mark Hall, Perry Lorier, and James Bruns-kill. Flow clustering using machine learning techniques. In *Passive and Active Network Measurement*, pages 205–214. Springer, 2004. 92
- [Mit10] Zeus Mitmo. Man-in-the-mobile. *Retrieved October*, 16:2011, 2010. 46
- [MKK07] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 421–430. IEEE, 2007. 47
- [Moh14] Sonal Mohite. A survey on mobile malware: War without end. *International Journal of Computer Science and Business Informatics*, 9(1), 2014. 28

- [Mor08] Dan Morrill. Announcing the android 1.0 sdk, release 1, 2008. 30
- [MP05] Andrew W Moore and Konstantina Papagiannaki. Toward the accurate identification of network applications. In *Passive and Active Network Measurement*, pages 41–54. Springer, 2005. 90
- [MS11] Kevin D Mitnick and William L Simon. *The art of deception: Controlling the human element of security*. John Wiley & Sons, 2011. 24
- [MSZ⁺07] Karim Mattar, Ashwin Sridharan, Hui Zang, Ibrahim Matta, and Azer Bestavros. *TCP over CDMA2000 networks: A cross-layer measurement study*. Springer, 2007. 91
- [Mur08] Mark L Murphy. *The busy coder’s guide to Android development*. United States: CommonsWare, 2008., 2008. 38
- [MV06] C. Mulliner and G. Vigna. Vulnerability analysis of mms user agents. In *Computer Security Applications Conference, 2006. AC-SAC’06. 22nd Annual*, pages 77–88. IEEE, 2006. 58
- [MVVK06] D. Mutz, F. Valeur, G. Vigna, and C. Kruegel. Anomalous system call detection. *ACM Transactions on Information and System Security (TISSEC)*, 9(1):61–93, 2006. 49, 57
- [MW06] Alok Madhukar and Carey Williamson. A longitudinal study of p2p traffic classification. In *Modeling, Analysis, and Simulation of Computer and Telecommunication Systems, 2006. MASCOTS 2006. 14th IEEE International Symposium on*, pages 179–188. IEEE, 2006. 90
- [MZ05] Andrew W Moore and Denis Zuev. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS Performance Evaluation Review*, volume 33, pages 50–60. ACM, 2005. 92

BIBLIOGRAFÍA

- [NA06] Thuy TT Nguyen and Grenville Armitage. Training on multiple sub-flows to optimise the use of machine learning classifiers in real-world ip networks. In *Local Computer Networks, Proceedings 2006 31st IEEE Conference on*, pages 369–376. IEEE, 2006. 92
- [NA08] Thuy TT Nguyen and Grenville Armitage. A survey of techniques for internet traffic classification using machine learning. *Communications Surveys & Tutorials, IEEE*, 10(4):56–76, 2008. 90
- [NC14] Mozilla Developer Network and Individual Contributors. Firefox os architecture, 2014. 5
- [NMHH05] D.C. Nash, T.L. Martin, D.S. Ha, and M.S. Hsiao. Towards an intrusion detection system for battery exhaustion attacks on mobile computing devices. In *Pervasive Computing and Communications Workshops, 2005. PerCom 2005 Workshops. Third IEEE International Conference on*, pages 141–145. Ieee, 2005. 57
- [Pas10] Anabel Pasiego. Blackberry os 6: Rim afronta una importante renovación de su sistema. *PC Actual. Personal computer*, (236):112–113, 2010. 17
- [Pax94] Vern Paxson. Empirically derived analytic models of wide-area tcp connections. *IEEE/ACM Transactions on Networking (TON)*, 2(4):316–336, 1994. 91
- [Pax99] Vern Paxson. Bro: a system for detecting network intruders in real-time. *Computer networks*, 31(23):2435–2463, 1999. 87
- [Poc11] Paul Pocatilu. Android applications security. *Informatica Economică*, 15(3):163–171, 2011. 35
- [Pos80] Jon Postel. User datagram protocol. *Isi*, 1980. 86
- [Pos81] Jon Postel. Internet protocol. 1981. 86

- [Pos83] Jon Postel. The tcp maximum segment size and related topics. 1983. 87
- [Pow07] DM Powers. Evaluation: From precision, recall and f-factor to roc, informedness, markedness & correlation (tech. rep.). *Adelaide, Australia*, 2007. 99
- [PPJP05] Kostas Pentikousis, Marko Palola, Marko Jurvansuu, and Pekka Perala. Active goodput measurements from a public 3g/umts network. *Communications Letters, IEEE*, 9(9):802–804, 2005. 91
- [Pro03] N. Provos. Improving host security with system call policies. In *Proceedings of the 12th USENIX Security Symposium*, volume 1, page 10. Washington, DC, 2003. 49
- [PSY10] Phillip Porras, Hassen Saidi, and Vinod Yegneswaran. An analysis of the ikee. b iphone botnet. In *Security and Privacy in Mobile Information and Communication Systems*, pages 141–152. Springer, 2010. 45
- [PYY14] Sancheng Peng, Shui Yu, and Aimin Yang. Smartphone malware and its propagation modeling: A survey. *Communications Surveys & Tutorials, IEEE*, 16(2):925–941, 2014. 45
- [Qui86] J. Ross Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986. 69
- [Qui93] J.R. Quinlan. *C4. 5: programs for machine learning*. Morgan kaufmann, 1993. 70
- [R⁺99] Martin Roesch et al. Snort: Lightweight intrusion detection for networks. In *LISA*, volume 99, pages 229–238, 1999. 87
- [RMC06] Radmilo Racic, Denys Ma, and Hao Chen. Exploiting mms vulnerabilities to stealthily exhaust mobile phone’s battery. In *Secu-recomm and Workshops, 2006*, pages 1–10. IEEE, 2006. 44

BIBLIOGRAFÍA

- [RN95] Stuart Russell and Peter Norvig. Artificial intelligence: a modern approach. 1995. 98, 99
- [Rol12] Michael Roland. Software card emulation in nfc-enabled mobile phones: great advantage or security nightmare. In *Fourth International Workshop on Security and Privacy in Spontaneous Interaction and Mobile Phone Use (IWSSI/SPMU 2012)*, page 6, 2012. 37
- [RSSD04] Matthew Roughan, Subhabrata Sen, Oliver Spatscheck, and Nick Duffield. Class-of-service mapping for qos: a statistical signature-based approach to ip traffic classification. In *Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 135–148. ACM, 2004. 90
- [RT11] Ittipon Rassameeroj and Yuzuru Tanahashi. Various approaches in analyzing android applications with its permission-based security models. In *Electro/Information Technology (EIT), 2011 IEEE International Conference on*, pages 1–6. IEEE, 2011. 51
- [RU05] Peter Reichl and Martina Umlauft. Project wisqy: A measurement-based end-to-end application-level performance comparison of 2.5 g and 3g networks. In *Wireless Telecommunications Symposium, 2005*, pages 9–14. IEEE, 2005. 91
- [SA08] A. Schmidt and S. Albayrak. Malicious software for smartphones. *Technische Universität Berlin, DAI-Labor, Tech. Rep. TUB-DAI*, 2:08–01, 2008. 29
- [SABZ05] Lawrence Stewart, Grenville Armitage, Philip Branch, and Sebastian Zander. An architecture for automated network control of qos over consumer broadband links. In *TENCON 2005 2005 IEEE Region 10*, pages 1–6. IEEE, 2005. 87
- [Sal94] Steven L. Salzberg. C4.5: Programs for machine learning by j. ross quinlan. morgan kaufmann publishers, inc., 1993. *Machine Learning*, 16:235–240, 1994. 10.1007/BF00993309. 70

- [San08] Vaibhav N Sanas. *Financial Management Application for RIM Blackberry Mobile Device*. PhD thesis, California State University, Chico., 2008. 4
- [SBS⁺09] A-D Schmidt, Rainer Bye, H-G Schmidt, Jan Clausen, Osman Kiraz, Kamer A Yuksel, Seyit Ahmet Camtepe, and Sahin Albayrak. Static analysis of executables for collaborative malware detection on android. In *Communications, 2009. ICC'09. IEEE International Conference on*, pages 1–5. IEEE, 2009. 63, 64
- [SBS⁺11] Igor Santos, Felix Brezo, Borja Sanz, Carlos Laorden, and Pablo Garcia Bringas. Using opcode sequences in single-class learning to detect unknown malware. *IET information security*, 5(4):220–227, 2011. 64
- [SDLPSPL⁺12] Igor Santos, Jorge De-La-PeñA-Sordo, Iker Pastor-López, Patxi Galán-García, and Pablo G Bringas. Automatic categorisation of comments in social news websites. *Expert Systems with Applications*, 39(18):13417–13425, 2012. 67
- [Ser10] Nicolas Seriot. iphone privacy. *Black Hat DC*, page 30, 2010. 40
- [SEZS01] M.G. Schultz, E. Eskin, F. Zadok, and S.J. Stolfo. Data mining methods for detection of new malicious executables. In *Security and Privacy, 2001. S&P 2001. Proceedings. 2001 IEEE Symposium on*, pages 38–49. IEEE, 2001. 50
- [SFE10] Asaf Shabtai, Yuval Fledel, and Yuval Elovici. Automated static code analysis for classifying android applications using machine learning. In *Computational Intelligence and Security (CIS), 2010 International Conference on*, pages 329–333. IEEE, 2010. 63, 64
- [SFK⁺09] Asaf Shabtai, Yuval Fledel, Uri Kanonov, Yuval Elovici, and Shlomi Dolev. Google android: A state-of-the-art review of security mechanisms. *arXiv preprint arXiv:0912.5101*, 2009. 34

BIBLIOGRAFÍA

- [Sha51] Claude E Shannon. Prediction and entropy of printed english. *Bell system technical journal*, 30(1):50–64, 1951. 71
- [Sha01] Claude Elwood Shannon. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review*, 5(1):3–55, 2001. 71
- [SKFT09] Wook Shin, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka. Towards formal analysis of the permission-based security model for android. In *Wireless and Mobile Communications, 2009. ICWMC'09. Fifth International Conference on*, pages 87–92. IEEE, 2009. 51, 62, 64
- [SKFT10] Wook Shin, Shinsaku Kiyomoto, Kazuhide Fukushima, and Toshiaki Tanaka. A formal model to analyze the permission authorization and enforcement in the android framework. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 944–951. IEEE, 2010. 62, 64
- [SKM09] Y. Singh, A. Kaur, and R. Malhotra. Comparative analysis of regression and machine learning methods for predicting fault proneness models. *International Journal of Computer Applications in Technology (2009)*, 35(2):183–193, 2009. 67, 79, 100
- [SLCS08] Dong-Her Shih, Binshan Lin, Hsiu-Sen Chiang, and Ming-Hung Shih. Security aspects of mobile phone virus: a critical survey. *Industrial Management & Data Systems*, 108(4):478–494, 2008. 29
- [SLTW04] Vladimir Svetnik, Andy Liaw, Christopher Tong, and Ting Wang. Application of breiman’s random forest to modeling structure-activity relationships of pharmaceutical molecules. In *Multiple Classifier Systems*, pages 334–343. Springer, 2004. 69, 96
- [SM97] D. Samfat and R. Molva. Idamn: an intrusion detection architecture for mobile networks. *Selected Areas in Communications, IEEE Journal on*, 15(7):1373–1380, 1997. 47, 55

- [Smi13] Aaron Smith. Smartphone ownership–2013 update. *Pew Research Center: Washington DC*, 12, 2013. 2
- [Son07] Marta Serrano Sonería. Seguridad de la información y protección de datos personales recabados a través de internet. In *Mundo Internet 2007: libro de ponencias*, pages 327–332, 2007. 113
- [Spa89] Kent A Spackman. Signal detection theory: Valuable tools for evaluating inductive learning. In *Proceedings of the sixth international workshop on Machine learning*, pages 160–163. Morgan Kaufmann Publishers Inc., 1989. 101
- [SPDB09] I. Santos, Y.K. Penya, J. Devesa, and P.G. Bringas. N-Grams-based file signatures for malware detection. In *Proceedings of the 11th International Conference on Enterprise Information Systems (ICEIS), Volume AIDSS*, pages 317–320, 2009. 11
- [SPL⁺09] A.D. Schmidt, F. Peters, F. Lamour, C. Scheel, S.A. Çamtepe, and Ş. Albayrak. Monitoring smartphones for anomaly detection. *Mobile Networks and Applications*, 14(1):92–106, 2009. 47, 54
- [SSL⁺13] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Álvarez. Puma: Permission usage to detect malware in android. In *International Joint Conference CISIS’12-ICEUTE’12-SOCO’12 Special Sessions*, pages 289–298. Springer, 2013. 52, 62, 64, 79
- [SSW04] Subhabrata Sen, Oliver Spatscheck, and Dongmei Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *Proceedings of the 13th international conference on World Wide Web*, pages 512–521. ACM, 2004. 90
- [Sta04] William Stallings. *Comunicaciones y redes de computadores*. 2004. 84
- [Sta15] Statista. Worldwide market share forecast of smartphone operating systems from 2010 to 2015, 2015. 28

BIBLIOGRAFÍA

- [TEMLP09] Patrick Traynor, William Enck, Patrick McDaniel, and Thomas La Porta. Mitigating attacks on open functionality in sms-capable cellular networks. *Networking, IEEE/ACM Transactions on*, 17(1):40–53, 2009. 44
- [TJHJ11] Wei Tang, Guang Jin, Jiaming He, and Xianliang Jiang. Extending android security enforcement with a security distance model. In *Internet Technology and Applications (iTAP), 2011 International Conference on*, pages 1–4. IEEE, 2011. 52
- [ÜMB06] Bülent Üstün, Willem J Melssen, and Lutgarde MC Buydens. Facilitating the application of support vector regression by using a universal pearson vii function based kernel. *Chemometrics and Intelligent Laboratory Systems*, 81(1):29–40, 2006. 97
- [VH08] D. Venugopal and G. Hu. Efficient signature based malware detection on mobile devices. *Mobile Information Systems*, 4(1):33–49, 2008. 47, 48
- [Wat08] Darren Waters. Google bets on Android future, February 2008. <http://news.bbc.co.uk/2/hi/technology/7266201.stm>. 61
- [WFP99] C. Warrender, S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: Alternative data models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*, pages 133–145. IEEE, 1999. 49
- [Whi10] Martin White. Information anywhere, any when: The role of the smartphone. *Business Information Review*, 27(4):242–247, 2010. 46
- [WMBW08] Daniel Willkomm, Sridhar Machiraju, Jean Bolot, and Adam Wolisz. Primary users in cellular networks: A large-scale measurement study. In *New frontiers in dynamic spectrum access networks, 2008. DySPAN 2008. 3rd IEEE symposium on*, pages 1–11. IEEE, 2008. 92

- [WT99] Alma Whitten and J Doug Tygar. Why johnny can't encrypt: A usability evaluation of pgp 5.0. In *Usenix Security*, volume 1999, 1999. 47
- [WWH08] T.Y. Wang, C.H. Wu, and C.C. Hsieh. A virus prevention model based on static analysis and data mining methods. In *Computer and Information Technology Workshops, 2008. CIT Workshops 2008. IEEE 8th International Conference on*, pages 288–293. IEEE, 2008. 50
- [YY12] Lok-Kwong Yan and Heng Yin. Droidscape: Seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In *USENIX Security Symposium*, pages 569–584, 2012. 52
- [Zei02] Kurt D Zeilenga. Internet assigned numbers authority (iana) considerations for the lightweight directory access protocol (ldap). 2002. 90
- [ZGZY11] Min Zhao, Fangbin Ge, Tao Zhang, and Zhijian Yuan. Antimal-droid: An efficient svm-based malware detection framework for android. In *Information Computing and Applications*, pages 158–166. Springer, 2011. 53
- [Zim80] Hubert Zimmermann. Osi reference model—the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4):425–432, 1980. 84
- [ZLH03] Y. Zhang, W. Lee, and Y.A. Huang. Intrusion detection techniques for mobile wireless networks. *Wireless Networks*, 9(5):545–556, 2003. 50
- [ZR07] Q. Zhang and D.S. Reeves. Metaaware: Identifying metamorphic malware. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 411–420. IEEE, 2007. 50

BIBLIOGRAFÍA

- [ZWZJ12] Yajin Zhou, Zhi Wang, Wu Zhou, and Xuxian Jiang. Hey, you, get off of my market: Detecting malicious apps in official and alternative android markets. In *NDSS*, 2012. 50
- [ZZJN12] Wu Zhou, Yajin Zhou, Xuxian Jiang, and Peng Ning. Detecting repackaged smartphone applications in third-party android marketplaces. In *Proceedings of the second ACM conference on Data and Application Security and Privacy*, pages 317–326. ACM, 2012. 63, 64

Esta tesis doctoral se terminó de escribir
en Bilbao el 30 de julio de 2015

