Sistema de Detección de Malware en Android

TRABAJO DE FIN DE GRADO

Inés Heras Cáceres
Diego Sierra Liras

Directores:
Luis Javier García Villalba
Ana Lucila Sandoval Orozco

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Madrid, Junio de 2015

Agradecimientos

Quisiéramos agradecer a Luis Javier García Villalba y a Ana Lucila Sandoval Orozco, los Directores de este Trabajo, el apoyo brindado.

Asimismo, quisiéramos agradecer la dedicación de Jorge Maestre Vidal. Sin su inestimable ayuda, este Trabajo no habría sido posible.

Finalmente, nuestro más sincero agradecimiento al resto de miembros del Grupo GASS (Grupo de Análisis, Seguridad y Sistemas, http://gass.ucm.es), Grupo de Investigación del Departamento de Ingeniería del Software e Inteligencia Artificial de la Facultad de Informática de la Universidad Complutense de Madrid, por las facilidades ofrecidas.

Resumen

El presente trabajo trata de dar respuesta al problema de la detección del malware en los dispositivos móviles. Los sistemas de protección que actualmente poseen los dispositivos móviles inteligentes se han mostrado ineficaces. Para asegurarlos frente a una posible infección de software malicioso, debe completarse la seguridad que ofrecen las características del sistema o la criba de aplicaciones en los mercados de descarga oficiales. Siendo un área de investigación reciente, la mayor parte de los trabajos se han centrado en la detección del *malware* estudiando su comportamiento en ejecución, que viene determinado por las llamadas al sistema que realiza. La principal motivación de este trabajo procede de la escasez de trabajos que realizan un estudio rápido y desligado de la actividad del usuario. Para ello el sistema propuesto extrae las llamadas al sistema realizadas durante los primeros segundos de ejecución de una aplicación en un entorno aislado. Además se tiene en cuenta no sólo la cantidad de llamadas al sistema sino también la información que se puede extraer de su secuencia temporal, gracias a la aplicación de algoritmos de alineamiento de secuencias. Los experimentos realizados demuestran que la actividad del malware tiene presencia al inicio de las ejecuciones, consiguiéndose así su detección temprana en la mayoría de los casos y minimizándose el impacto sobre el sistema protegido.

Palabras Clave

Alineamiento de Secuencias, Android, Aplicación, Detección, Dispositivos Inteligentes, Llamadas al Sistema, Malicioso, Malware.

Abstract

This work seeks to solve the problem of detecting malware on mobile devices. Protection systems currently have smart mobile devices have proven ineffective. In order to insure against possible infection of malicious software, security must be completed features offered by the system or the official markets. Being an area of recent research, most of the work has focused on the detection of malware studying their behavior, which is determined by the system calls during the execution. The main motivation of this work comes from the scarcity of jobs performed quickly and detached study of user activity. For this, the proposed system extracts the system calls made during the first seconds of running an application in an isolated environment. Also it is taking into account not only the number of system calls but also information that can be gleaned from their timing, using sequence alignment algorithms. The results obtained from the experiments demonstrate the objectives were successfully completed. The proposal has shown that malware activity is present at the beginning of the executions. It has also got an early detection of malware in most cases, thus the impact on the protected system has been minimal.

Keywords

Android, Application, Detection, Malicious, Malware, Sequence Alignment, Smart Devices, System Calls.

Los abajo firmantes autorizan a la Universidad Complutense de Madrid (UCM) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores el presente Trabajo Fin de Grado: "Sistema de Detección de Malware en Android", realizado durante el curso académico 2014-2015 bajo la dirección de Luis Javier García Villalba y Ana Lucila Sandoval Orozco en el Departamento de Ingeniería del Software e Inteligencia Artificial, y a la Biblioteca de la UCM a depositarlo en el Archivo Institucional E-Prints Complutense con el objeto de incrementar la difusión, uso e impacto del trabajo en Internet y garantizar su preservación y acceso a largo plazo.

Inés Heras Cáceres

Diego Sierra Liras

Índice General

1. Introducción **3**

1.1. Malware para Móviles . . . 4

1.1.1. Comportamiento y Objetivos del Ataque . . . 4

1.1.2. Métodos de Distribución . . . 5

1.1.3. Adquisición de Privilegios . . . 6

1.2. Modelos de Seguridad . . . 6

1.2.1. Symbian . . . 8

1.2.2. BlackBerry . . . 8

1.2.3. Windows Mobile . . . 8

1.2.4. Android . . . 9

1.2.5. iOS . . . 9

1.3. Estructura de la Memoria . . . 9

2. Trabajos Relacionados **11**

2.1. Tratamiento de Datos . . . 11

2.2. Estrategias de Detección . . . 12

2.2.1. Análisis Estático . . . 12

2.2.2. Análisis Dinámico . . . 13

2.2.3. Análisis Mixto . . . 15

2.2.4. Análisis por Metadatos . . . 16

2.3. Conjuntos de Evaluación . . . 16

3. Métodos de Análisis **19**

3.1. Procesamiento de Datos . . . 19

3.1.1. Red Bayesiana . . . 19

3.1.2. Cadenas de Markov . . . 20

3.1.3. Máquinas de Vector Soporte . . . 21

3.1.4. Alineamiento de Secuencias . . . 22

3.1.4.1. Alineamiento Global . . . 22

3.1.4.2. Alineamiento Local . . . 24

3.2. Pruebas Estadísticas . . . 25

3.2.1. Prueba de los Rangos con Signo de Wilcoxon . . . 26
3.2.2. Prueba U de Mann-Whitney . . . 28
3.3. Modelos de Identificación de Poblaciones . . . 28

4. Detección de Malware en Android **31**
4.1. Extracción de Datos . . . 32
4.2. Tratamiento de Datos . . . 33
4.3. Experimentación . . . 34
4.3.1. Etapas . . . 35
4.3.2. Optimización . . . 35
4.4. Resultados . . . 38

5. Conclusiones y Trabajo Futuro **41**
5.1. Conclusiones . . . 41
5.2. Trabajo Futuro . . . 41

6. Contribuciones **43**

I Resumen en Inglés **51**

A. Introduction **53**
A.1. Mobile Malware . . . 54
A.1.1. Behavior and Objectives of the Attack . . . 54
A.1.2. Distribution Methods . . . 55
A.1.3. Acquisition of Privileges . . . 56
A.2. Current Security Models . . . 57
A.2.1. Symbian . . . 58
A.2.2. BlackBerry . . . 58
A.2.3. Windows Mobile . . . 58
A.2.4. Android . . . 58
A.2.5. iOS . . . 59

B. Conclusions and Future Work **61**
B.1. Conclusions . . . 61
B.2. Future Work . . . 62

Índice de Figuras

1.1. Tipos de ataques 5
1.2. Tipos de distribuciones 6
1.3. Formas de adquisición de privilegios 7
1.4. Ventas de aplicaciones 7

3.1. Esquema de decisión de las redes bayesianas 20
3.2. Plano de representación de datos en SVM 20
3.3. Alineamiento Global versus Alineamiento Local 22
3.4. Tabla de grado de semejanza y penalización por hueco 23
3.5. Matriz de decisión en el algoritmo de Needleman-Wunsch 24
3.6. Matriz de decisión en el algoritmo de Smith-Waterman 25
3.7. Esquema de pruebas estadísticas según sus características 26

4.1. Esquema del sistema 32
4.2. Esquema de funcionamiento de los sistema GNU/UNIX 33
4.3. Resultado de puntuaciones 36
4.4. Comparación de las medias 37
4.5. Diferencia de puntuaciones 38

A.1. Classification of attacks per operating system 55
A.2. Distribution methods per operating system 56
A.3. How privileges are aquired 56
A.4. Sales of applications per type of operating system 57

Índice de Tablas

3.1. Diferencia entre elementos en la prueba de Wilcoxon 26
3.2. Diferencias ordenadas en la prueba de Wilcoxon 27
3.3. Asignación de rangos de orden en la prueba de Wilcoxon 27
3.4. Resultado de T+, T- y T 27

4.1. Primeros resultados sin optimizar el sistema 37
4.2. Resultados 39

Capítulo 1

Introducción

En los últimos años se ha incrementado la aparición de distintos tipos de dispositivos inteligentes. Así, entre un 60 y un 80 por ciento de la población de los países en los que hay acceso a Internet, lo hacen a través de *Smartphones* o Tabletas [1], los cuales son utilizados una media de dos horas diarias. Además, se estima que en 2017 habrá en el mundo 1,4 dispositivos móviles por habitante y sus ventas superarán a las de las computadoras personales. A la luz de estos datos, es evidente que el interés y la importancia de estos dispositivos en la sociedad es cada vez más notable y, presumiblemente, seguirá avanzando a pasos agigantados.

De entre las características de los dispositivos móviles destaca la presencia de sensores tales como giroscopios, micrófonos, localizadores GPS, etc.; la capacidad de conexión a algún tipo de red telefónica, *Bluetooth*, Wi-Fi; o la posibilidad de adquirir y utilizar aplicaciones desarrolladas por terceros [2]. Pero a pesar de las evidentes ventajas que ofrecen, presentan también ciertos problemas que pueden llegar a ser cruciales para la seguridad del usuario. Esto es debido a que estos dispositivos almacenan información muy sensible y variada, que puede comprometer la seguridad, la privacidad e, incluso, la economía del propietario o la de terceros. Los sensores que incorporan estas tecnologías también pueden recoger datos sin que el usuario sea realmente consciente de la cantidad de información que el dispositivo está manejando. Es por esto por lo que la creación de software malicioso o *malware*, específico para este ámbito, se ha disparado a la misma velocidad que su uso.

Finalmente, es importante destacar que entre los usuarios no existe conciencia del peligro al que están expuestos ante un posible ataque. La inmensa mayoría desconoce incluso que el *malware* para móviles existe, y que las aplicaciones que instalan en sus dispositivos pueden presentar comportamientos malintencionados. Por ello, se hace necesario plantear modelos de seguridad más allá de la acción del usuario, controlando los mercados y los dispositivos.

1.1. Malware para Móviles

Malware es cualquier tipo de software o código de programa hostil, intrusivo o diseñado para usar un dispositivo sin el conocimiento del propietario. La evolución y proliferación del *malware* en dispositivos móviles está íntimamente ligada al aumento de sus capacidades de red y recursos informáticos. Así, se tienen evidencias de que el primer *malware* para móviles fue desarrollado en 2004 para atacar a dispositivos Symbian [3]. Sin embargo, es a partir de 2010 cuando empieza a crecer significativamente [4], estando dedicado principalmente a los sistemas Android e iOs. Sólo para Android en 2012 se estima que existían cerca de 35.000 programas maliciosos, siendo objetivo del 79 % del *malware* generado durante ese año frente al 11,5 % del año 2010 [5].

Para analizar los ataques y sus tipos, se deben tener en cuenta tres perspectivas: comportamiento y objetivos, formas de distribución del *malware* y métodos de adquisición de privilegios.

1.1.1. Comportamiento y Objetivos del Ataque

En la mayoría de los casos el *malware* persigue varios objetivos y además su propósito y comportamiento puede variar por medio de una orden remota emitida por quienes lo controlan. El más común de sus objetivos es el beneficio económico. En este sentido se pueden encontrar ejemplos varios, como en el caso del fraude u overbilling. Este ataque consiste en cargar gastos a la cuenta de la víctima (que son transferidos a la del atacante) por medio de llamadas o SMS enviados a números de tarificación adicional sin el consentimiento del usuario. Habitualmente, el usuario relaciona este tipo de ataques con un conflicto con la compañía proveedora del servicio, pasando por alto su verdadera naturaleza [6].

Otro tipo de ataque es el de la denegación de servicio o sabotaje, el llamado ataque DoS (del inglés Denial of Service) que trata de consumir la batería del dispositivo limitando su tiempo de operación, o bien de negarle al usuario el acceso a determinados recursos o redes [7]. Cuando se ejecuta de forma distribuida (DDoS), puede ocasionar perjuicios no sólo a uno o varios usuarios, sino incluso a diversas organizaciones [2]. Por ejemplo, se puede llegar a colapsar la conexión a Internet de una zona o servidor si se empiezan a enviar paquetes masivos a una red.

Por último, se puede tratar de comprometer la privacidad por medio de la técnica llamada sniffing [8], la cual aprovecha los datos que recogen los sensores y las redes de las que hace uso el dispositivo, pudiendo obtener imágenes [9], grabaciones telefónicas, contraseñas, mensajes de correo electrónico, datos bancarios o cualquier tipo de información que maneje, envíe o reciba el dispositivo. Esta es una de las amenazas más preocupantes

para los usuarios u organizaciones, puesto que en los dispositivos móviles se almacenan y se tratan datos privados y clasificados, tanto personales como empresariales.

En la Figura 1.1 se detalla la ocurrencia de cada tipo de objetivo por cada sistema operativo.

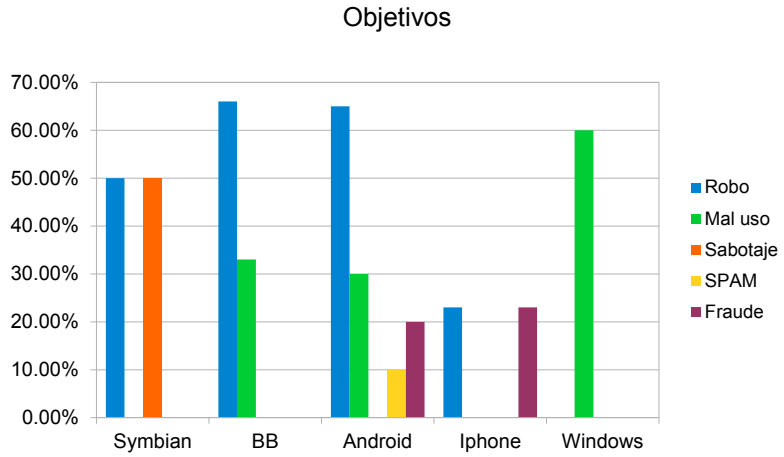

Figura 1.1: Tipos de ataques

1.1.2. Métodos de Distribución

Es importante conocer los medios por los que se distribuyen los programas maliciosos, para así atajar el contagio. Hay dos grandes aproximaciones [10]: la autopropagación y la ingeniería social. Dentro del primer tipo se agrupan los siguientes métodos: Aplicación a Dispositivo (Application to Device, A2D, el código malicioso se encuentra en una aplicación que luego infecta al dispositivo), SMS a Dispositivo (SMS to Device, S2D, se contagia el dispositivo por medio de un SMS), USB a Dispositivo (USB to Device, U2D), Red a Dispositivo (Network to Device, N2D), Dispositivo a Dispositivo (Device to Device, D2D) y Nube a Dispositivo (Cloud to Device, C2D). En estos casos el *malware* está programado para contaminar el dispositivo cuando se pone en contacto con él.

Por otro lado, la ingeniería social, agrupa métodos como Mercado a Dispositivo (Market to Device, M2D, el usuario descarga desde el mercado un programa malicioso) o el de Navegador Web a Dispositivo (Web-browser to Device, W2D).

Como se muestra en la Figura 1.2, la ingeniería social es el principal método de distribución de *malware*. Posiblemente esto se deba a la poca conciencia ante el peligro que muestran los usuarios; habitualmente son los que descargan las aplicaciones infectadas desde los mercados. También es frecuente la combinación de ambas estrategias.

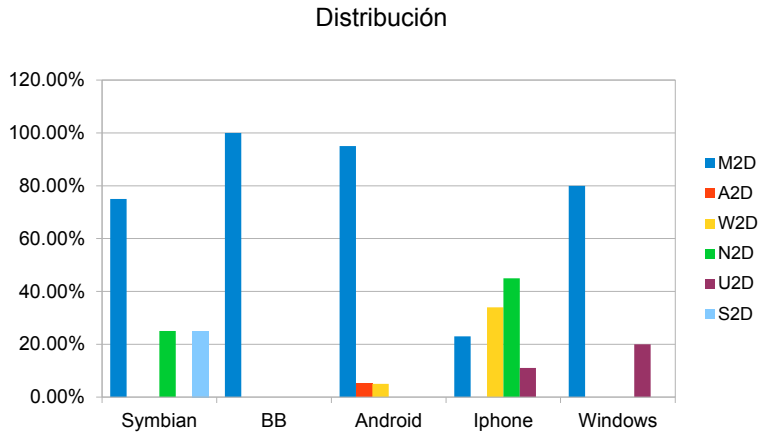

Figura 1.2: Tipos de distribuciones

1.1.3. Adquisición de Privilegios

Para que el *malware* sea efectivo no es suficiente con que se propague al dispositivo, también tiene que conseguir una serie de privilegios para poder acceder a las partes críticas del sistema y realizar las acciones necesarias para ejecutar su labor. Habitualmente, estos privilegios son directamente concedidos por los usuarios [10] al instalar aplicaciones en apariencia legítimas pero que esconden algún comportamiento malicioso. Los usuarios no son conscientes de la repercusión de transferir determinados permisos a programas desconocidos y, en ocasiones, ni tan siquiera prestan atención a las peticiones para su autorización. Por lo tanto, este método es altamente eficaz, tal y como se muestra en la Figura 1.3.

Otro procedimiento basado en la tecnología consiste en la explotación de las vulnerabilidades o errores en la configuración de la plataforma [11]. Se utilizan rootkits para detectar de qué forma se pueden aprovechar dichos errores. Este tipo de *malware* infecta el sistema operativo, por lo que puede ser considerablemente peligroso y, además, dejan la puerta abierta a futuras infecciones.

1.2. Modelos de Seguridad

Hoy en día, los mercados oficiales de distribución de aplicaciones cuentan con distintos mecanismos para intentar garantizar la seguridad de las mismas. Principalmente, utilizan pruebas de verificación para comprobar la legitimidad del código de la aplicación. Sin embargo, detectar el *malware* es demasiado complejo; no se conoce en detalle el funcionamiento interno de estas pruebas y la presencia de un considerable número de aplicaciones maliciosas evidencia que es un método insuficiente. Además se deben tener en cuenta las

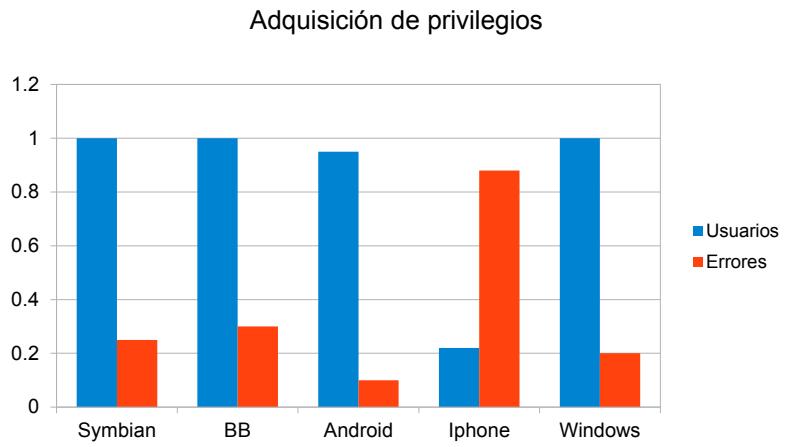

Figura 1.3: Formas de adquisición de privilegios

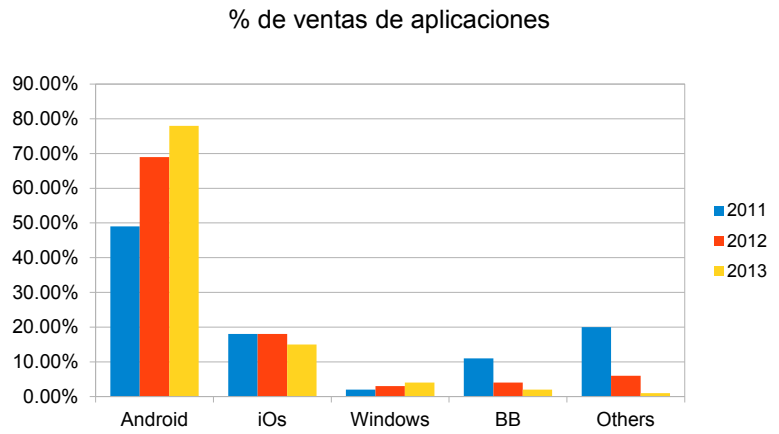

Figura 1.4: Ventas de aplicaciones

descargas desde mercados no oficiales que no poseen ningún tipo de filtro de seguridad y son altamente peligrosas.

Desde el punto de vista de la plataforma, una manera de tratar de garantizar la seguridad es la de restringir la comunicación entre aplicaciones y las acciones que éstas pueden realizar, incluyendo el acceso a datos y servicios. También se propone la técnica de aislar la ejecución de la aplicación en entornos controlados, llamada *sandboxing*.

En la Figura 1.4 se muestra que Android es el sistema operativo móvil que predomina, así como una tendencia a que continúe creciendo [5]. iOS es el segundo más vendido, aunque a cierta distancia de Android, y el resto cada vez tiene menos importancia en cuanto a ventas de dispositivos y aplicaciones.

A continuación, se analizan los métodos y decisiones adoptados por los cinco principales sistemas operativos que dominan actualmente.

1.2.1. Symbian

El modelo de seguridad de Symbian se basa en un sistema básico de permisos para controlar los recursos del dispositivo. Las aplicaciones se ejecutan en el espacio de usuario mientras que el sistema operativo lo hace en el espacio del núcleo.

Las aplicaciones que requieren acceso a bibliotecas protegidas y, por lo tanto, son más vulnerables, deben estar firmadas con un certificado expedido por Symbian [12], mientras que todas las demás pueden estar autofirmadas, no teniendo acceso a estas librerías peligrosas. De esta forma las aplicaciones ya están controladas y la seguridad de los mercados se hace innecesaria, siendo casi inexistente. Para la mejora de la seguridad de Symbian se ha propuesto el uso de algoritmos de aprendizaje automático que analicen el comportamiento de las aplicaciones [13]. A pesar de su alta fiabilidad, Symbian es de los sistemas operativos menos utilizados.

1.2.2. BlackBerry

La seguridad de BlackBerry se basa en un esquema amplio de permisos que ha demostrado ser muy seguro [14]. Inicialmente, las aplicaciones tienen un acceso muy limitado a los recursos y, posteriormente, pueden recibir autorización para escalar sus privilegios. Es necesaria la firma del fabricante para acceder a las bibliotecas, ofreciendo un sistema de protección básico para procesos y memoria. Este es probablemente el sistema más seguro de los que se han analizado, pero presenta el inconveniente de ser también muy cerrado, con un mercado muy limitado.

1.2.3. Windows Mobile

Microsoft [15] basa su modelo de seguridad en la validación de los desarrolladores y la reputación y valoración de cada aplicación. En las últimas versiones, cada aplicación se ejecuta en su propio *sandbox*, en el que se conceden los permisos a los que se ha dado autorización, de forma similar a como se relaciona en el sistema Android. Sin embargo, estos permisos se piden en la instalación y deben ser concedidos por los usuarios, los cuales no suelen tener en cuenta la importancia de los mismos. Además, es muy difícil modificarlos en tiempo de ejecución [16].

1.2.4. Android

El esquema de seguridad de Android se centra principalmente en el propio dispositivo, ya que los usuarios están autorizados a descargar aplicaciones desde cualquier plataforma de distribución o mercado. El formato de permisos se traduce en un manifiesto que debe ser autorizado por el usuario durante la instalación y adjudicado en tiempo de ejecución. Sin embargo, debido a la tolerancia del usuario respecto a la concesión de permisos, esta técnica es poco efectiva.

Android también utiliza la técnica del *sandboxing* [17] [18], aislando cada aplicación en su propia máquina Dalvik. Ésta genera un código de *bytes* y otorga a cada una un identificador de usuario distinto, a excepción de las aplicaciones de un mismo desarrollador, las cuales comparten identificador, lo que les permite compartir recursos. Sin embargo, las aplicaciones pueden compartir información entre ellas explícitamente mediante una interfaz para la comunicación.

Es interesante centrarse en este sistema en concreto puesto que es el más utilizado, y también el que más infecciones sufre.

1.2.5. iOS

A diferencia de Android, Apple dirige su seguridad hacia el mercado debido a que limita la capacidad de descarga al suyo propio [19]. Por lo tanto, las aplicaciones y los desarrolladores necesitan para su verificación una firma mediante un certificado expedido por Apple. Sin embargo, los detalles de las pruebas de verificación no son de dominio público, por lo que su eficacia es difícil de evaluar.

Por lo general, su seguridad a nivel de plataforma es más débil que en Android, o prácticamente inexistente, ya que las aplicaciones se ejecutan en un único entorno aislado, común para todas ellas, teniendo además acceso a la mayoría de los recursos del dispositivo. Por ello, en los últimos modelos se está tratando de ampliar este sistema, controlando el tráfico de datos personales.

1.3. Estructura de la Memoria

Esta memoria se estructura en 5 Capítulos, siendo el primero la presente introducción. En el Capítulo 2 se exponen los trabajos que previamente han tratado de dar respuesta al problema del *malware* en dispositivos móviles. En el Capítulo 3 se explican las principales herramientas que se han tenido en cuenta a la hora de tomar la decisión de si los datos observados se considera legítimos o maliciosos. En el Capítulo 4 queda detallada la propuesta que expone el proyecto. El Capítulo 5 muestra las conclusiones de este trabajo.

Capítulo 2

Trabajos Relacionados

Para la realización de este trabajo se han analizado los estudios y proyectos que previamente han tratado de dar solución al problema del *malware* en dispositivos móviles. Así, se puede dar una visión general de las utilidades necesarias para llevar a cabo esta labor y conocer de forma concisa cómo se puede implementar el proceso de detección de *malware* para dispositivos móviles.

En este capítulo se estudian los modos de tratar los datos una vez han sido recopilados y extraídos del dispositivo que se va a estudiar, así como los tipos de estrategias de detección de *malware* en dispositivos móviles desde cuatro perspectivas, dependiendo de la naturaleza de los datos. Éstas son: análisis estático, análisis dinámico, análisis mixto y análisis de metadatos. También se detallan los conjuntos de *malware* más utilizados parece realizar pruebas.

2.1. Tratamiento de Datos

Los dispositivos móviles cuentan con recursos bastante limitados, sobre todo en cuanto a la batería y velocidad de procesamiento, por lo que es importante determinar de forma correcta el lugar en el cual se van a analizar los datos extraídos del mismo y la forma en la que se va a realizar dicho análisis [20].

Una primera aproximación consiste en realizar el análisis en el propio dispositivo. Resulta una ventaja puesto que, al no ser necesaria una conexión con el exterior, no existe dependencia de otros servidores o de la velocidad de la conexión a Internet. Además, se evitan las brechas de seguridad derivadas de la transmisión de datos a través de la red, ya que pueden ser interceptados y modificados.

Sin embargo, la mayoría de los proyectos estudiados realizan el análisis en servidores remotos o en la nube. Se tiene en cuenta que la mayoría de los dispositivos móviles están

conectados a la red casi de forma constante, por lo que la conexión no debería suponer un excesivo inconveniente. Además, al hacerlo de este modo, no se consumen recursos del dispositivo móvil, disponiéndose de sistemas con altas capacidades de procesamiento.

2.2. Estrategias de Detección

Para poder extraer los datos de la aplicación o sistema y hacerlos útiles para obtener conclusiones sobre su legitimidad, hay que elegir qué subconjunto de información es el que se va a analizar. Hay múltiples elementos que pueden ser monitorizados en un dispositivo móvil. Por ejemplo, componentes hardware, las comunicaciones, los sensores, el sistema (que incluye los procesos, la memoria, el planificador y el almacenamiento) o la actividad del usuarios.

A continuación, se detallan las cuatro estrategias de análisis utilizadas en los trabajos relativos a esta temática, en cada una de las cuales se monitoriza uno o varios de los elementos anteriores.

2.2.1. Análisis Estático

El análisis estático se centra en el estudio del archivo APK que contiene la aplicación móvil que se desea analizar. Un archivo APK viene definido por tres aspectos: el fichero AndroidManifest donde se declaran los permisos que requerirá la aplicación, el código fuente Java que posee la funcionalidad y, por último, los recursos necesarios para un correcto funcionamiento como pueden ser bases de datos, imágenes o diseños.

De esta forma el análisis estático investiga acerca de los permisos concedidos a la aplicación, su código fuente, sus filtros de intención, los sensores hardware que utilizará (ya que suelen sugerir su funcionalidad) y, por último, las direcciones de red asignadas evitando un posible ataque de comando y control.

Una primera aproximación a este método es proporcionada por Androguard [21], que permite descompilar el código Dalvik, y dados dos archivos APK, compara la similitud de su código fuente en Java.

Una forma de concluir el estudio del código fuente es identificar el *malware* mediante una firma. Esta firma supone haber encontrado una secuencia de llamadas en el código que se ha demostrado maliciosa. Sin embargo, este método es altamente vulnerable a la ofuscación, pues una nueva aplicación con un cambio de orden en esta secuencia será indetectable.

DroidMat [22], en cambio, centra su estudio en los filtros de intención recogidos en el AndroidManifest. Para ello procesa los datos mediante la fórmula de la probabilidad condicionada de Bayes y los algoritmos de agrupamiento k-means y vecino más próximo. A su vez, DREBIN [23] utiliza el AndroidManifest para descubrir los recursos hardware que la aplicación usará. Si por ejemplo necesita 3G y GPS puede tratarse de un *malware* que registra la localización del usuario. El inconveniente de estas propuestas es que es difícil conseguir una gran precisión basándose en sólo un aspecto concreto de los que conforman el análisis estático como es el AndroidManifest.

Otro método que estudia más características es el proporcionado por TrustDroid [24]. En él se describe una estrategia que consiste en aislar y clasificar las aplicaciones en distintas categorías en función de su procedencia, la firma del desarrollador, las certificaciones que posea, los permisos requeridos o la información de la que se disponga sobre la aplicación en una base de datos común que se iría creando de manera colaborativa. De esta forma se puede regir el comportamiento de cada categoría y la interacción entre ellas. Cabe decir que este método es sencillo, pero aunque permite identificar *malware* previamente conocido gracias a la información de la base de datos, sería incapaz de detectar una muestra maliciosa desconocida.

El análisis estático presenta la ventaja de la sencillez del proceso de extracción de los datos necesarios, residentes en el archivo APK. Además, dado que Android funciona bajo un núcleo Linux, el estudio de los permisos concedidos a la aplicación supone la primera línea de defensa. Sin embargo, un estudio centrado en los permisos no consigue una alta precisión, pues no permite definir las actividades que realmente se están llevando a cabo. Por lo tanto, se hace necesario estudiar el código, ya que es la fuente de las acciones maliciosas. Sin embargo, esta es una labor compleja, pues es vulnerable a técnicas de ofuscación o cifrado para evadir los sistemas de detección. De esta forma un estudio completo de las características estáticas se convierte en un trabajo con cada vez más dificultad, que requiere de un desarrollo agresivo de nuevos métodos.

2.2.2. Análisis Dinámico

El análisis dinámico consiste en monitorizar el comportamiento del sistema para extraer la actividad de las aplicaciones instaladas a través de los siguientes identificadores: llamadas al sistema, interacción con el usuario, uso de los componentes hardware o tráfico de la red. Posteriormente, se realiza una fase de entrenamiento en la que se aprende cuál es el comportamiento adecuado para poder clasificar de forma correcta los datos obtenidos y, de este modo, poder distinguir posteriormente las actividades legítimas de las maliciosas.

El modo de análisis dinámico más extendido es el estudio de las llamadas al sistema [5]. Esto es debido a que el estudio completo de las características del tráfico de red supone investigar la gran cantidad de registros donde se almacenan los cambios, lo que puede llegar a ser muy costoso computacionalmente.

Dado que el comportamiento del *malware* debe verse reflejado en varios ámbitos es importante poder seleccionar las características más importantes. Esto permite hacer el análisis más sencillo y efectivo. En este sentido, Andromaly [25] propone una plataforma de trabajo dedicada a la detección ligera de *malware*, llevando a cabo la recopilación de datos concretos tales como el consumo de batería, número de paquetes enviados a través de Wi-Fi, el número de procesos en ejecución o el consumo de CPU. Más tarde, la información es procesada según el tipo de amenaza conocida, concluyendo con la emisión de alertas al usuario y la toma de acciones automáticas que mitiguen la amenaza.

Otro ejemplo de este método es MADAM [26]. En él se mide la frecuencia de ejecución de cada llamada al sistema en T períodos, seleccionando las 12 llamadas que se han considerado como más representativas (open, ioctl, brk, read, write, exit, close, sendto, recvfrom, recvmdg). También se registra el número de SMS enviados, así como si el usuario ha estado usando el dispositivo o, en cambio, ha estado en modo inactivo.

Por lo tanto, la información a tratar es representada mediante un vector de datos de 14 posiciones, donde las 12 primeras corresponden a las llamadas y las dos últimas indican si el dispositivo ha estado activo y cuántos SMS se han enviado. Este vector es analizado y comparado con los resultados aprendidos anteriormente, para así determinar si el comportamiento del dispositivo está siendo adecuado (estándar) o malicioso (sospechoso). El mayor inconveniente de esta aproximación es la incapacidad de determinar posteriormente, de forma sencilla, cuál es la aplicación que está provocando el comportamiento malicioso.

En Crowdroid [27] se proponen registrar las llamadas al sistema realizadas por una aplicación durante la interacción del usuario en su totalidad. A partir de ello es posible la generación de un vector cuya longitud sea el número de llamadas que existen en Android (dependiendo de su versión puede variar entre 250 y 380). Para cada uso de una aplicación, se registra el número de veces que se ha producido cada una de ellas. Después, se envían los datos a un servidor remoto. En él se agrupan los datos obtenidos en las diversas ejecuciones mediante el algoritmo K-means y se componen los resultados del uso legítimo de cada aplicación. De esta forma, una vez construido un esquema de cómo debería ser su uso normal, éste se podrá comparar con los datos extraídos de ejecuciones desconocidas de las aplicaciones para las cuales existan datos en el sistema.

Esta propuesta presenta la ventaja de que, al realizarse el análisis en un servidor externo al dispositivo, no se consume batería ni recursos del mismo. Además, permite comparar una ingente cantidad de datos, ya que podrían haber muchos dispositivos enviando los suyos. Sin embargo, la idea de utilizar vectores de llamadas conlleva un claro inconveniente, y es que no resulta portable entre diferentes versiones de Android.

En VirusMeter [28] la detección mediante el análisis del uso de recursos se centra en el análisis de la batería del dispositivo. Los autores asumen la premisa de que cualquier actividad maliciosa debe consumir energía. Basándose en un modelo de potencia predefinido para el gasto de la batería en una actividad normal, se supervisan las actividades del sistema, y si estas han producido un gasto mayor a un umbral especificado, se advierte al usuario de la situación de riesgo.

En general, esta estrategia de detección se hace compleja y costosa en dispositivos con recursos limitados, como son los dispositivos móviles. Además, deben de ser capaces de ignorar su propia actividad, centrándose únicamente en las características auditadas provenientes de la actividad externa. Sin embargo, han demostrado ser los más efectivos pues permiten identificar el inevitable rastro que el *malware* deja en la actividad del sistema, puesto que enmascarar la actividad maliciosa durante la ejecución conlleva un mayor grado de dificultad que hacerlo en las características estáticas, susceptibles a la ofuscación y cifrado.

2.2.3. Análisis Mixto

Los entornos más complejos y más escasos para la detección de *malware* utilizan conjuntamente técnicas de análisis estático y dinámico.

En AASandbox [18] se extraen los permisos y el código de Java desde el propio archivo APK. Éstos son utilizados como características estáticas. A continuación, se instala la aplicación estudiando el sistema de registro de llamadas. Así son consideradas como características dinámicas.

La propuesta de ProfileDroid [29] consiste en examinar el AndroidManifest y el código Java como características estáticas. Además, se estudia la interacción del usuario, el sistema de registro de llamadas y el tráfico de red como características dinámicas.

Estos sistemas, aunque completos, son muy costosos de desarrollar y mantener debido a la gran cantidad de trabajo que abarcan.

2.2.4. Análisis por Metadatos

Por último, para detectar muestras maliciosas unas pocas propuestas se centran en la información de la aplicación a la que un usuario puede acceder antes de su descarga. Entre ellos destacan la descripción de la aplicación y su valoración en el mercado de descargas, la identificación del creador, la categoría a la que pertenece, los permisos que solicita, nombre del paquete, vídeo promocional, sitio web de contacto o precio.

En WHYPER [30] se recogen únicamente los metadatos relacionados con los permisos solicitados. En cambio, en [31] se recopilan además una gran cantidad de datos respecto a la última vez que se modificó la aplicación, tales como el precio o el número de descargas.

El inconveniente de estas técnicas es que requieren de nuevos sistemas que permitan la interpretación de los datos, como es el caso de sistemas de procesamiento de lenguaje natural, con el fin de identificar elementos disonantes en las opiniones o en la descripción de una aplicación.

2.3. Conjuntos de Evaluación

Cada experimento necesita usar un conjunto de datos que serán la base sobre la que los autores probarán su sistema. El *malware* en Android es un área de investigación reciente: el primer *malware* para Android se descubrió en 2010 [4] [31]. Debido a esto, inicialmente los investigadores no tenían un conjunto sólido estándar de muestras con las que trabajar. En su lugar, se veían obligados a escribir su propio *malware* para utilizarlo en su sistema. Otros investigadores trataron de recopilar muestras a través de numerosas páginas web para después compartirlas, como es el caso del proyecto Contagio [32]. Por lo tanto, la limitación de muestras de *malware* era una brecha que concluía en sistemas de evaluación poco fiables.

Para solventar este problema, en 2012 se liberó el proyecto *Android Malware Genome* [33]. Este repositorio contiene 1.260 muestras de *malware* para Android pertenecientes a 49 familias, recogidas desde Agosto de 2010 hasta Octubre de 2011. Su uso en más de 20 proyectos desde 2012 [5] demuestra que los investigadores lo consideran un conjunto de muestras sólido, convirtiéndose en un estándar.

Sin embargo, dada la naturaleza cambiante del *malware*, era necesario actualizar el conjunto de datos para adaptarlo a las nuevas amenazas centradas en el robo y cifrado de la información que guarda el teléfono. En este contexto, en 2014 se introdujo el proyecto DREBIN [23]. Se trata de una colección de 5.560 muestras de *malware* de 179 diferentes

familias, que fueron recogidas desde 2010 hasta octubre de 2012 y utilizadas en 19 proyectos durante 2014 [5].

Un conjunto menos popular es el proporcionado por VirusShare [32]. Además, existen sistemas en línea que escanean una aplicación para determinar su legitimidad como son VirusTotal [34] y AndroTotal [35].

Capítulo 3

Métodos de Análisis

Para llevar a cabo este proyecto se han estudiado distintos métodos de análisis y clasificación de datos. En este capítulo se explican los algoritmos que se han tenido en cuenta para procesar y comparar los datos utilizados como referencia con aquellos de origen desconocido, con el fin de determinar su naturaleza. Éstos son: redes bayesianas, máquinas de vector soporte y alineamiento de secuencias.

Además, se describen las pruebas estadísticas estudiadas que permiten, a partir de las puntuaciones obtenidas de los algoritmos, calcular el estadístico que revela a qué población pertenece una muestra. Por último, se detalla el funcionamiento de métodos para la creación de modelos que definan las poblaciones de muestras que se toman para el estudio.

3.1. Procesamiento de Datos

Para procesar los datos es necesaria la aplicación de algoritmos que conviertan la información extraída en información útil con la que identificar los puntos más relevantes para la identificación del *malware*. A continuación se explican los algoritmos cuyo uso está más extendido en este ámbito.

3.1.1. Red Bayesiana

Las redes bayesianas [36] consisten en un tipo de modelo de probabilidad en el que se relacionan una serie de hechos aleatorios que pueden ocurrir y la relación de probabilidad entre ellos. Se representa como un grafo en el que cada nodo supone un suceso y las aristas implican dependencia condicional. Cada nodo tiene una distribución de probabilidad que depende de los datos de entrada y tiene como salida la probabilidad de que ese suceso ocurra. Las redes bayesianas pueden entenderse como la relación de causa-efecto entre hechos que pueden ocurrir. Así, en la Figura 3.1 se representa que si el suceso B ocurre,

existe una cierta probabilidad de que ocurra A y, a su vez, C. Además, la probabilidad de que ocurra C será:

$$P(C) = P(B)P(A)$$

Este tipo de redes son utilizadas en muy diversos campos, como bioinformática, procesamiento de imágenes y textos o sistemas de toma de decisiones. En este último caso, las redes bayesianas resultan muy útiles cuando no se tienen muchos parámetros de entrada. Sin embargo, no tienen en cuenta la secuencia de hechos, simplemente se necesita conocer el estado anterior, lo cual puede suponer un problema en el caso de los programas informáticos, en los que el orden en el que ocurren los sucesos suele ser relevante.

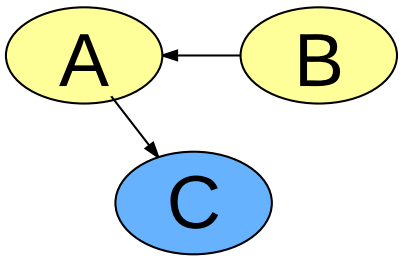

Figura 3.1: Esquema de decisión de las redes bayesianas

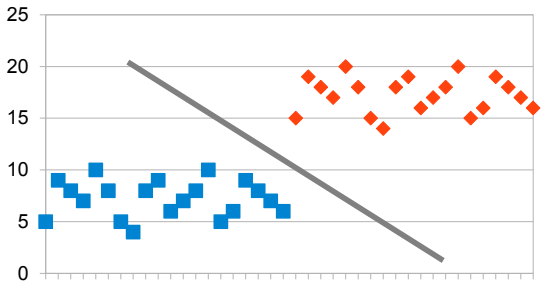

Figura 3.2: Plano de representación de datos en SVM

3.1.2. Cadenas de Markov

Para solucionar el inconveniente de no tener en cuenta la secuencia temporal de hechos presente en las redes bayesianas, se pueden utilizar las cadenas o modelos de Markov [37].

Se trata de un tipo de proceso en el que la probabilidad de que un evento llegue a ocurrir depende única y exclusivamente del suceso anterior, cumpliendo así la llamada propiedad de Markov. Sin embargo, aunque un estado dependa solamente del anterior, el historial de eventos ocurridos queda representado en cada estado, de forma que se dispone de toda la información necesaria para determinar la probabilidad de que ocurran los estados siguientes.

La cadena de Markov se compone de una secuencia de eventos aleatorios X_1, X_2, ..., X_n, X_{n+1}. Por la propiedad de Markov, la probabilidad de X_{n+1} depende únicamente de X_n:

$$P(X_{n+1} = x_{n+1}|x_n, X_{n-1} = x_{n-1}, \ldots, X_2 = x_2, X_1 = x_1) = P(X_{n+1} = x_{n+1}|X_n = x_n)$$

A pesar de que de este método es utilizado en muy diversos campos como medicina, física o juegos de azar y, aunque ya se ha utilizado en otros proyectos de detección de *malware*, pueden alcanzar una complejidad considerable cuando se trata de cadenas con una alta cantidad de posibles eventos aleatorios.

3.1.3. Máquinas de Vector Soporte

Las máquinas de vector soporte (o SVM, del inglés *Support Vector Machines*) son un conjunto de algoritmos supervisados que pertenecen a la familia de los clasificadores lineales. Estos algoritmos representan los datos como puntos en un plano, lo cual permite crear clasificaciones dependiendo de las agrupaciones de puntos. Para crear un correcto modelo de clases, debe existir una fase de entrenamiento que permita definir mediante la generación de un hiperplano una separación óptima entre conjuntos de puntos [38], tal y como se muestra en la Figura 3.2. Una vez obtenido el modelo, dada una muestra desconocida permite diferenciar de forma clara los datos analizados para determinar a qué clase pertenecen.

Este tipo de algoritmo ya ha sido utilizado en otros proyectos de detección de *malware* [13], lo cual supone una ventaja puesto que ya han sido probados. Sin embargo, en la fase de entrenamiento es complicado determinar cuál debe ser la diferencia para considerar un dato como perteneciente a un grupo o a otro, lo cual puede dar lugar a errores. Además, como en el caso de redes bayesianas, solamente se tienen en cuenta los hechos ocurridos, y no la secuencia temporal de los mismos, lo cual es importante a la hora de analizar la ejecución de programas.

3.1.4. Alineamiento de Secuencias

Los algoritmos de alineamiento de secuencias [39] tratan de comparar dos cadenas de elementos para resaltar sus zonas de similitud, rellenando con huecos las partes no coincidentes. Habitualmente, son utilizados en bioinformática para comparar cadenas de ADN y ARN. Sin embargo, son perfectamente aplicables en otros ámbitos.

Las cadenas a comparar se colocan de tal forma que se hagan coincidir las zonas de mayor similitud. Finalmente, resulta una puntuación que determina el grado de semejanza entre ambas. Esto se consigue añadiendo huecos o espacios, que serán los que se penalizan negativamente. Además, existen varios tipos de alineamientos de secuencias, entre los que destacan el alineamiento global y el local. Aunque para cadenas suficientemente similares, los resultados entre ellos no se diferencian. En la Figura 3.3 se muestran ambos, visualizándose el sistema de adición de huecos en cada caso.

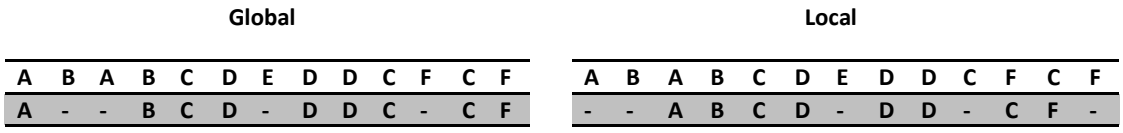

Figura 3.3: Alineamiento Global versus Alineamiento Local

Habitualmente, las secuencias están formadas por los elementos de un alfabeto finito de símbolos. En un principio, es necesario establecer un grado de semejanza entre estos símbolos, lo que se puede conseguir mediante una matriz cuadrada de similitud. En esta matriz de similitud (S) se encuentra en cada celda $S(i, j)$ el valor indicado para los símbolos i y j del alfabeto.

Adicionalmente, se requiere de un parámetro (d) que indique la penalización por hueco o gap. En la Figura 3.4 se muestra un ejemplo de alineamiento y los parámetros utilizados.

3.1.4.1. Alineamiento Global

El alineamiento global sirve para cotejar cadenas en su conjunto forzando la comparación desde su inicio hasta ocupar la longitud total. Por este motivo, funciona especialmente bien cuando las secuencias iniciales son similares y globalmente son de tamaño parecido.

Un algoritmo habitual para su implementación y que consigue la ordenación de las secuencias es el algoritmo Needleman-Wunsch [40], basado en programación dinámica.

-	A	B	C	D
A	2	-3	4	1
B	-3	5	0	1
C	4	0	8	-7
D	1	1	-7	-5

d=-1

A	B	A	B	C	D
A	-	-	B	C	C

S(A,A) + 2*d + S(B,B) + S(C,C) + S(D,C)=

2 + (-2) + 5 + 8 + (-7) = 6

Figura 3.4: Tabla de grado de semejanza y penalización por hueco

Resulta fiable porque siempre termina, funciona de forma independiente a las secuencias de entrada y además asegura la mejor solución.

El algoritmo ordena ambas secuencias mediante una matriz (F), de tamaño la longitud de la primera secuencia $|A| = m$ y la longitud de la segunda secuencia $|B| = n$, esto es, ($|A| \times |B|$). En esta matriz el valor de la celda $F(i, j)$ corresponde a la puntuación del mejor alineamiento entre los subsegmentos de los primeros i elementos de la secuencia A y los j primeros elementos de la secuencia B. Es decir, la matriz F guarda el valor de los alineamientos parciales. De esta forma el valor de $F(m, n)$ es el óptimo para el alineamiento de ambas secuencias en su totalidad.

La primera fila y columna de (F) vienen determinadas por los elementos de las secuencias y la penalización por hueco de la siguiente forma:

$$F(0, j) = d * j; F(i, 0) = d * j$$

A partir de esos datos es posible completar el resto de la matriz. El valor de una celda F(i,j) viene determinado por el de la fila, columna o diagonal anteriores de la siguiente forma:

$$\text{máx}\,[F(i-1, j) + d, F(i, j-1) + d, F(i-1, j-1) + S(A_i, B_j)]$$

Una vez completada la matriz (F), para ordenar las secuencias y añadir los huecos pertinentes se parte de la posición $F(m, n)$, como se aprecia en la Figura 3.5. En este caso se debe retroceder en la matriz, eligiendo como siguiente posición el mayor valor de entre $F(i-1, j)$, $F(i, j-1)$ y $F(i-1, j-1)$. De esta forma se comprueba cuál fue la solución óptima usada a la hora de construir la celda $F(i, j)$. Si el valor elegido es $F(i-1, j)$, debe alinearse $A(i)$ con un hueco, si fue $F(i, j-1)$ es $B(i)$ el que debe alinearse con un hueco y, finalmente, si se eligió $F(i-1, j-1)$, A_i y B_i están alineados entre sí. El proceso se repite hasta acabar en un valor de la primera fila o columna [41].

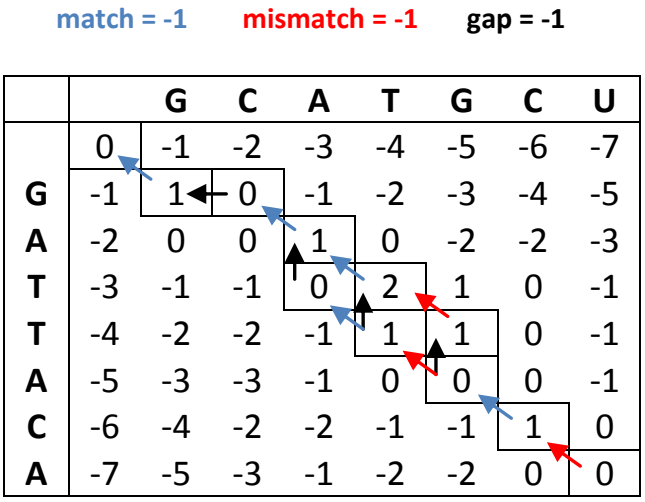

Figura 3.5: Matriz de decisión en el algoritmo de Needleman-Wunsch

3.1.4.2. Alineamiento Local

El alineamiento local es más apropiado para encontrar subsecuencias similares dentro de un contexto mayor, al contrario que el global que compara las secuencias en su conjunto. En este caso, al intentar buscar secciones dispersas de gran similitud, la diferencia de longitud entre las cadenas no perjudica excesivamente el resultado final.

Para llevarlo a cabo se utiliza habitualmente el algoritmo recursivo de Smith-Waterman [42]. El ordenamiento de las secuencias se consigue recorriendo la matriz (F) de la misma forma en la que se recorría en el algoritmo Needleman-Wunsch. Sin embargo, las características de (F) son distintas. La primera diferencia es la forma de completar la primera columna y fila que, en este caso, valen 0, esto es:

$$F(0, j) = 0; \quad F(i, 0) = 0$$

Además, el valor óptimo para el alineamiento se puede encontrar en cualquier celda de

la matriz y se empieza a recorrer a partir de ella en vez de $F(m, n)$ hasta que se encuentra un 0, como se expone en la Figura 3.6.

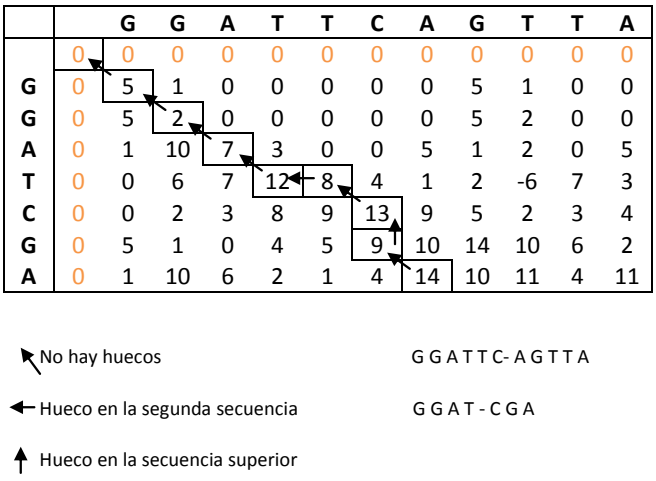

Figura 3.6: Matriz de decisión en el algoritmo de Smith-Waterman

3.2. Pruebas Estadísticas

Una vez se dispone de una o varias poblaciones de datos, se lleva a cabo un contraste para calcular el estadístico que permita decidir si los resultados de una muestra desconocida analizada pertenecen a alguna de ellas o, si por el contrario, no se tienen datos suficientes para decidirlo.

Como se muestra en la Figura 3.7, existen multitud de pruebas estadísticas indicadas para distintos tipos de distribuciones y datos. En el caso de los datos que vamos a analizar relativos a las llamadas al sistema, no se puede asumir que su distribución vaya a ser normal, por lo tanto se tratan como datos no normalizados.

Por ello, se han estudiado dos pruebas populares para variables continuas como son la prueba de los rangos con signo de Wilcoxon y la prueba U de Mann-Whitney. Al no suponer ninguna distribución, utilizan la mediana de las muestras para comparar; además también trabajan con rangos de orden.

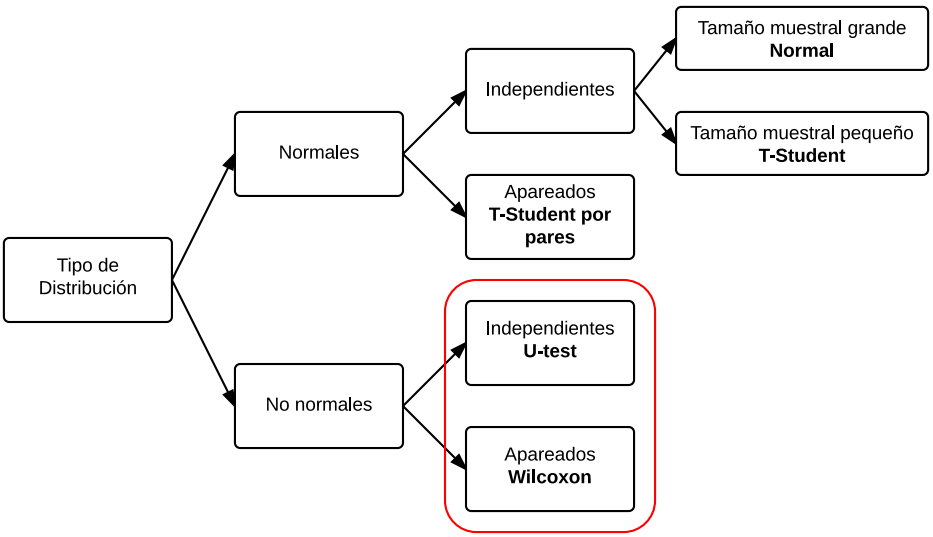

Figura 3.7: Esquema de pruebas estadísticas según sus características

3.2.1. Prueba de los Rangos con Signo de Wilcoxon

La prueba de Wilcoxon [43] trabaja sobre vectores con datos apareados, es decir, se comparan entre sí los datos con el mismo índice. Esto supone que el tamaño de los vectores (n) debe ser muy similar para las dos muestras.

La hipótesis nula viene definida como que la mediana de las diferencias de los elementos del vector sea igual a 0. El proceso que sigue permite obtener un valor p para decidir si no se cumple la hipótesis nula, esto es, que dicha diferencia no se deba al azar y, por lo tanto, se pueda determinar que ambas poblaciones son la misma.

1. Primero se obtienen las diferencias de los pares de elementos que se están comparando uno a uno

Tabla 3.1: Diferencia entre elementos en la prueba de Wilcoxon

A	4	3	6	7	9	4	2	5
B	3	4	6	6	5	4	7	3
Diferencia	1	-1	6	1	4	0	-5	2

2. Las diferencias se ordenan sin tener en cuenta los signos.

Tabla 3.2: Diferencias ordenadas en la prueba de Wilcoxon

Diferencias ordenadas sin signo	0	0	1	1	1	2	4	5

3. Una vez ordenadas, se les asigna un rango. Si existen elementos repetidos, el rango se determina en la media de los rangos correspondientes.

Tabla 3.3: Asignación de rangos de orden en la prueba de Wilcoxon

Diferencias ordenadas sin signo	0	0	1	1	1	2	4	5
Rangos	-	-	2	2	2	4	5	6

4. Se suman por un lado los rangos de las diferencias positivas ($T+$) y por otro lado los rangos de las diferencias negativas ($T-$) y se calcula el estadístico contraste T como el mínimo de ambos: $T = \text{mín}[T+, T-]$.

Tabla 3.4: Resultado de T+, T- y T

A	4	3	6	7	9	4	2	5
B	3	4	6	6	5	4	7	3
Diferencia	1	-1	6	1	4	0	-5	2
Rangos	2	2	-	2	5	-	6	4
	$T+ = 13$			$T- = 8$			$T = 8$	

5. Una vez obtenidos estos 3 parámetros, a partir del estadístico T se obtiene un p valor. Para tamaños muestrales pequeños ($n < 20$) el p valor se calcula gracias a la tabla estadística de la distribución de Wilcoxon. Para tamaños muestrales mayores se consigue mediante Z que es una aproximación normal del valor T por la siguiente fórmula:

$$Z = \frac{\frac{T - n(n+1)}{4}}{\sqrt{\frac{n(n+1)(2n+1)}{24}}}$$

En función del intervalo de confianza utilizado, normalmente 0,05 o 0,01, y de la p obtenida, se puede determinar si ambos vectores son similares o si la diferencia es significativa. Si $p <$intervalo, se deduce que la diferencia entre poblaciones no se debe al azar y,

por lo tanto, la diferencia si es significativa. En caso contrario, se puede determinar que no se dispone de la información suficiente para hacer esta afirmación.

3.2.2. Prueba U de Mann-Whitney

La prueba U de Mann-Whitney [44], a diferencia de Wilcoxon, no trabaja vectores con datos apareados. En este caso no se comparan los elementos con el mismo índice, es decir, las dos muestras son independientes y por tanto pueden tener tamaños muestrales (n) distintos.

La hipótesis nula es que la mediana de las muestras son iguales. El estadístico de contraste $U = \text{mín}(U_1, U_2)$ viene determinado por la siguiente fórmula:

$$U_1 = n_1 n_2 + \frac{n_1(n_1+1)}{2} - R_1$$

$$U_2 = n_1 n_2 + \frac{n_2(n_2+1)}{2} - R_2$$

donde R_1 y R_2 son la suma de los rangos de las muestras.

Para observaciones de tamaño muestral bajo ($n < 20$) se calcula el p valor con el estadístico U y su tabla estadística. Para pruebas mayores se utiliza la aproximación normal O definida como sigue.

$$Z = \frac{\frac{U-(n_1,n_2)}{2}}{\sqrt{\frac{n_1,n_2(n+1))}{12}}}$$

De igual forma el valor de p nos permite decidir si las diferencias son significativas.

3.3. Modelos de Identificación de Poblaciones

Una vez obtenidos los resultados del procesamiento, se puede recurrir a 3 estrategias distintas con el fin de determinar las poblaciones que se van a tener en cuenta para decidir si la muestra de datos que se está estudiando es legítima o maliciosa [45] [46].

En primer lugar, la identificación basada en anomalía consiste en realizar dinámicamente un modelo del uso correcto de la aplicación o sistema, de tal forma que dicha población pueda ser comparada posteriormente con los datos extraídos para saber si son legítimos. Si se determina que esos datos no son legítimos, se clasifican directamente como malignos. Esta idea resulta interesante si no se disponen de muestras maliciosas con las que comparar. Sin embargo, presentan gran cantidad de fallos y falsos positivos cuando las muestra legítimas demuestran comportamientos poco habituales.

También puede darse el enfoque opuesto, es decir, la identificación basada en el mal uso, en la que se modelan los comportamientos incorrectos. En general, este tipo de sistemas presentan un claro problema a la hora de mantener actualizada la base de datos de señales que determinan si un comportamiento es o no malicioso. Sin embargo, resultan más precisos, ya que son más sencillos de detectar los síntomas concretos que suelen presentar las aplicaciones maliciosas.

En último lugar, la identificación basada en especificación consiste en mantener una lista de comportamientos autorizados para cada aplicación o sistema. De esta forma, todas las actividades que se salen de las lista de autorizadas, se consideran inmediatamente como maliciosas. Es una opción muy sencilla, pero poco efectiva debido a que las aplicaciones legítimas podrían incurrir en alguna acción fuera de esta lista.

Capítulo 4

Detección de Malware en Android

Tras estudiar el estado del arte en relación al problema de la detección de *malware* en dispositivos móviles, se ha decidido optar por una estrategia de detección mediante análisis dinámico. Se ha pensado que, a diferencia de muchos de los trabajos realizados hasta el momento que sólo se fijaban en el número de veces que ocurre un determinado evento, en este caso, una llamada al sistema, es importante tener en cuenta la secuencia en la que ocurren estas llamadas, ya que grandes diferencias en el orden en el que suceden las cosas pueden ser determinantes a la hora de distinguir si una muestra tomada es legítima o no.

Además, se ha decidido estudiar las llamadas al sistema realizadas al comienzo de la ejecución de la aplicación, es decir, durante su lanzamiento. De esta forma, el análisis se hace independiente de la acción del usuario y permite detectar la actividad maliciosa en su fase más temprana.

En consecuencia, dado que las secuencias de llamadas de distintos lanzamientos de una aplicación siempre van a tener un tamaño similar y un mismo patrón, la forma idónea de alineamiento para compararlas es mediante el algoritmo de Needleman-Wunsch.

Por último, se ha determinado utilizar un modelo basado en anomalías que recopila el comportamiento legítimo de la aplicación para así no depender de muestras anómalas para el análisis. La prueba estadística usada es la de Wilcoxon debido a que las muestras no son independientes, sino que se compara cada una con una ejecución legitima. En la Figura 4.1 se muestra un esquema de los pasos que sigue el sistema desde que la aplicación es descargada hasta que se toma la determinación de si es legítima o maliciosa.

En este capítulo se habla de la extracción y procesamiento de los datos, del entrenamiento del sistema y de los resultados que se han obtenido tras realizar diversas pruebas.

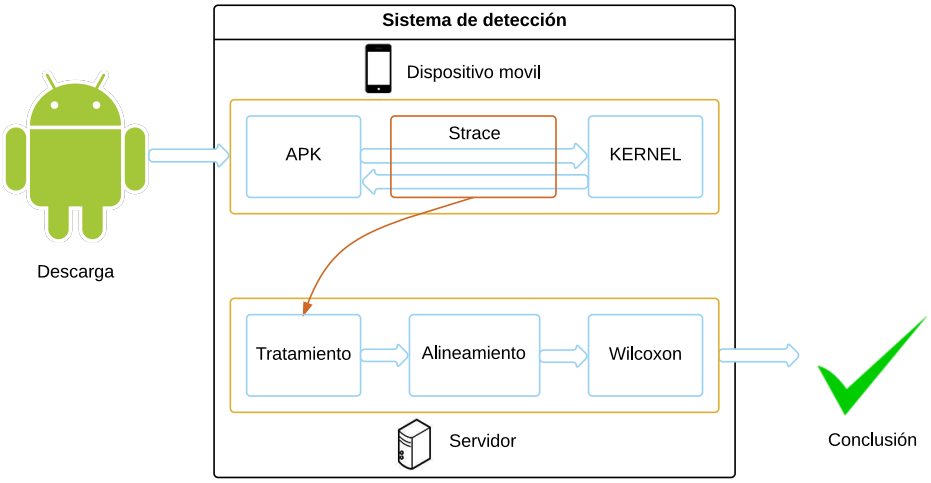

Figura 4.1: Esquema del sistema

4.1. Extracción de Datos

En varios de los proyectos estudiados [26] [27] se hace uso de la herramienta de diagnóstico Strace, existente en los sistemas GNU/LINUX, como es Android. Esta herramienta permite registrar las llamadas al sistema realizadas por un programa o proceso. Esto se logra realizando una monitorización de la interfaz de llamadas al sistema, la cual permite la comunicación entre el kernel y las capas superiores del sistema operativo, como se muestra en la Figura 4.2. Además, de las llamadas al sistema se pueden extraer también las señales emitidas por el programa. Strace es especialmente útil para detectar problemas y es muy utilizado por investigadores y desarrolladores.

Para poder registrar todas las actividades realizadas por un programa desde el momento de su lanzamiento, incluyendo aquellos procesos que se deriven del mismo, se debe monitorizar el proceso padre o cigoto (*zygote*) de Android, del cual parten todos los demás. Se trata de un demonio desencadenado por el proceso init de android y a su vez el encargado de generar todos los procesos de las aplicaciones. *Zygote* queda escuchando a la espera en un **socket**. Al recibir una petición crea un nuevo hijo mediante un proceso fork creando una máquina virtual Dalvik para una nueva aplicación, con todas las librerías precargadas.

En este trabajo se ha creado una apk cuya función es lanzar un proceso Strace que monitorice *zygote*. De forma paralela, inmediatamente lanza la aplicación que se va a estudiar. Para tratar de garantizar que, en caso de que la aplicación sea maliciosa provoque

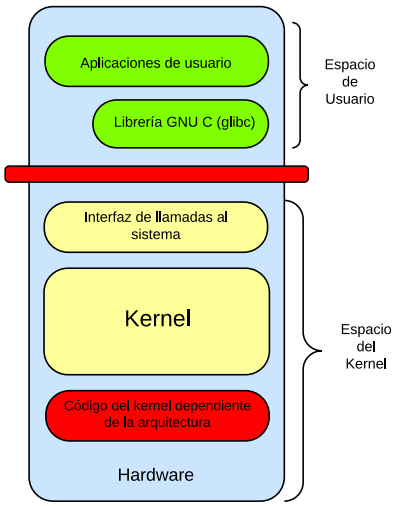

Figura 4.2: Esquema de funcionamiento de los sistema GNU/UNIX

los mínimos problemas en el sistema, y además eliminar las variaciones propias de la actividad del usuario, los datos se deben extraer inmediatamente después de la instalación de la aplicación y sólo durante los primeros segundos de ejecución, los correspondientes al lanzamiento de la misma. Los resultados de la monitorización realizada por Strace quedan registrados en un fichero de texto en el que se muestran las llamadas que se han producido, el proceso al que pertenecen y algunas de las señales que manejan dichas llamadas.

4.2. Tratamiento de Datos

Tras haber extraído los datos del dispositivo se ha decidido realizar el procesamiento en un servidor remoto. Los escasos recursos con los que cuentan los dispositivos móviles así como la limitación de no disponer de una fuente de alimentación constante, suponen un gran problema si se quiere realizar el trabajo en el propio equipo. Además, al realizar la extracción inmediatamente después de la descarga e instalación, se puede presuponer que existe una conexión a Internet suficiente para enviar los datos.

Hay que tener en cuenta que las llamadas al sistema realizadas en el arranque de la aplicación puede estar en una horquilla de entre 2.000 para las aplicaciones más sencillas y 50.000 en las más complejas. Tal cantidad de datos deben ser tratados y simplificados para facilitar su análisis. Para ello se ha creado un programa que hace corresponder cada llamada al sistema con un número, de forma que el resultado será un vector, con todas las

llamadas al sistema que han ocurrido y en el orden que han sucedido, representadas por números, de forma que resultará más sencillo de procesar.

Tras esta simplificación, se ha optado por una estrategia de identificación basada en anomalía. Se comparan los datos desconocidos con los conocidos como legítimos mediante un algoritmo de alineamiento de secuencias global descrito en el apartado 3.1.4.

Como resultado del algoritmo se obtiene una puntuación de similitud entre vectores. A continuación, las puntuaciones que se conocen como legítimas y las desconocidas son sometidas a una prueba no paramétrica de Wilcoxon (descrita en el apartado 3.2.1), y en función de la p obtenida y de la diferencia de medias, se determina si la muestra es legítima o maliciosa.

4.3. Experimentación

La fase de experimentación se ha realizado utilizando una máquina virtual de Android 4.4 KitKat. Se han realizado las pruebas con 19 aplicaciones distintas de las que se disponen de una versión legítima y otra contaminada, que se han obtenido del repositorio *Android Malware Genome* y *Drebin dataset*. En concreto las aplicaciones son Diner Dash 2, Jaro, Mash, Plumber, Fruits Maching, Scrambled Net, Solitaire, Tap and Furious, Robotic Space Rock, Basketball shot, Monkey Jump 2, Whites out, Super touch down, Tilt Mazes, Helix, DailyMoney, Sanity, Best Voice Changer y Z-test. Las versiones maliciosas están contaminadas por los siguientes tipos de *malware*

- DroidKungFu: inhibe la actividad de los programas antivirus, y además crea una puerta trasera que permite a los atacantes hacerse con el control del dispositivo.
- Plankton: envía datos del teléfono a un servidor remoto el cual, a lo largo del tiempo, transmite órdenes para obtener información privada del usuario
- Geinimi: lee, envía y borra SMS; envía a un servidor la información de los contactos y la localización; descarga ficheros sin el conocimiento del usuario y abre páginas en el navegador.
- GinMaster: envía información confidencial a un servidor remoto e instala aplicaciones sin el permiso del usuario.
- Cogos: *malware* genérico de pruebas

- jSMSHider: un gusano que, además de enviar y recibir mensajes de texto, tiene la capacidad de abrir páginas webs y aplicaciones sin el conocimiento del usuario. Las consecuencias de ambos son de extrema gravedad.
- VdLoader: envía notificaciones por SMS y transmite la información de las aplicaciones instaladas en el dispositivo a un servidor remoto.
- Gapev: descarga otras aplicaciones y suscribe al usuario a un servicio de SMS premium.
- Gamex: descarga ficheros sin el conocimiento del usuario

4.3.1. Etapas

Para cada una de las aplicaciones la experimentación se ha realizado en tres etapas.

- **Etapa 1**: En esta primera etapa se extraen los datos de 30 ejecuciones distintas de la aplicación en su versión legítima. A continuación, esas 30 muestras se han cruzado entre sí (Figura 4.3) usando el algoritmo de alineamiento, obteniendo para cada una un vector de 30 puntuaciones. Se ha calculado la media de cada uno de esos vectores, logrando finalmente 30 medias (marcadas en azul en la Figura 4.3).
- **Etapa 2**: En la segunda etapa se extraen los datos de las ejecuciones desconocidas cuya naturaleza se quiera definir. Para este proyecto se han tomado varias decenas de muestras de la aplicación legítima y de la aplicación que contiene *malware*. Cada una de estas muestras se ha cruzado con las 30 ejecuciones legítimas iniciales, obteniendo así un vector de 30 posiciones con una puntuación por fichero legítimo.
- **Etapa 3**: Por último, se compara mediante Wilcoxon el vector de 30 puntuaciones de cada fichero desconocido con el vector de 30 puntuaciones de medias legítimas, como se muestra en la Figura 4.4.

4.3.2. Optimización

En un primer intento se ha llevado a cabo la experimentación con las aplicaciones DailyMoney y DinerDash. Para ello se ha aplicado alineamiento global sobre el total de la secuencia de llamadas al sistema de todos los archivos examinados. Para el alineamiento global se ha definido -1 como la penalización por hueco o cuando dos elementos no son iguales y $+1$ cuando dos elementos coinciden.

	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12	L13	L14	L15	L16	L17	L18	L19	L20	L21	L22	L23	L24	L25	L26	L27	L28	L29	L30
L1		1496	1522	1427	1479	1309	1373	1485	1437	1382	1379	1404	1385	1316	1471	1433	1596	1463	1496	1363	1424	1417	1423	1446	1505	1353	1482	1410	1468	1463
L2	1496		1377	1327	1353	1439	1517	1498	1474	1429	1456	1392	1499	1412	1450	1502	1424	1345	1402	1488	1427	1537	1437	1283	1390	1561	1588	1509	1654	1426
L3	1522	1377		1355	1334	1303	1339	1428	1316	1395	1449	1316	1431	1315	1362	1335	1419	1336	1462	1392	1468	1339	1362	1438	1395	1363	1396	1382	1415	1463
L4	1427	1327	1355		1550	1317	1396	1367	1440	1258	1273	1376	1282	1418	1533	1332	1429	1531	1512	1229	1378	1374	1427	1421	1303	1285	1354	1359	1352	1381
L5	1479	1353	1334	1550		1366	1421	1323	1362	1283	1284	1295	1247	1386	1587	1334	1483	1633	1477	1255	1419	1416	1403	1445	1281	1264	1426	1389	1369	1412
L6	1309	1439	1303	1317	1366		1512	1417	1488	1353	1349	1485	1376	1492	1405	1336	1359	1334	1447	1425	1275	1532	1429	1241	1212	1510	1459	1464	1493	1434
L7	1373	1517	1339	1396	1421	1512		1472	1532	1512	1411	1501	1462	1591	1499	1438	1503	1495	1530	1551	1389	1593	1674	1308	1354	1523	1602	1615	1517	1619
L8	1485	1498	1428	1367	1323	1417	1472		1429	1549	1512	1336	1552	1418	1533	1332	1429	1531	1512	1229	1378	1374	1427	1421	1303	1285	1354	1359	1352	1381
L9	1437	1474	1316	1440	1362	1488	1532	1429		1430	1380	1417	1392	1477	1561	1412	1481	1424	1510	1433	1374	1577	1536	1336	1378	1453	1484	1520	1521	1479
L10	1382	1429	1395	1258	1283	1353	1512	1549	1430		1547	1487	1500	1385	1427	1498	1440	1320	1377	1511	1359	1394	1437	1382	1485	1455	1525	1507	1440	1425
L11	1379	1456	1449	1273	1284	1349	1411	1512	1380	1547		1509	1426	1364	1374	1615	1367	1310	1406	1489	1412	1395	1412	1377	1405	1382	1595	1495	1519	1395
L12	1404	1392	1316	1376	1295	1485	1501	1336	1417	1487	1509		1401	1359	1432	1464	1404	1349	1454	1379	1283	1359	1469	1366	1479	1335	1474	1429	1394	1474
L13	1385	1499	1431	1282	1247	1376	1462	1552	1392	1500	1426	1401		1419	1375	1477	1367	1235	1390	1450	1393	1424	1429	1278	1421	1439	1508	1524	1533	1421
L14	1316	1412	1315	1418	1386	1492	1591	1452	1477	1385	1364	1359	1419		1413	1353	1391	1372	1451	1514	1411	1501	1475	1287	1228	1435	1531	1514	1533	1511
L15	1471	1450	1362	1533	1587	1405	1499	1418	1561	1427	1374	1432	1375	1413		1386	1626	1510	1549	1344	1359	1518	1533	1400	1514	1309	1381	1536	1481	1457
L16	1433	1502	1335	1332	1334	1336	1438	1504	1412	1498	1615	1464	1477	1353	1386		1406	1340	1353	1401	1430	1429	1407	1368	1448	1452	1660	1416	1569	1375
L17	1596	1424	1419	1429	1483	1359	1503	1399	1481	1440	1367	1404	1367	1391	1626	1406		1534	1648	1311	1396	1483	1553	1395	1530	1323	1437	1491	1403	1518
L18	1463	1345	1336	1531	1633	1334	1495	1382	1424	1320	1310	1349	1235	1372	1510	1340	1534		1493	1288	1448	1398	1404	1466	1367	1285	1440	1436	1365	1435
L19	1496	1402	1462	1512	1477	1447	1530	1404	1510	1377	1406	1454	1390	1451	1549	1353	1648	1493		1361	1363	1494	1556	1367	1433	1378	1479	1474	1424	1611
L20	1363	1488	1392	1229	1255	1425	1551	1515	1433	1511	1489	1379	1450	1514	1344	1401	1311	1288	1361		1367	1499	1444	1312	1348	1543	1556	1483	1506	1480
L21	1424	1427	1468	1378	1419	1275	1389	1504	1374	1359	1412	1283	1393	1411	1359	1430	1396	1448	1363	1367		1428	1359	1412	1358	1376	1457	1421	1463	1395
L22	1417	1537	1339	1374	1416	1532	1593	1516	1577	1394	1395	1359	1424	1501	1518	1429	1483	1398	1494	1499	1428		1592	1281	1324	1536	1484	1596	1540	1535
L23	1423	1437	1362	1427	1403	1429	1674	1462	1536	1437	1412	1469	1429	1475	1533	1407	1553	1404	1556	1444	1359	1592		1299	1363	1480	1464	1489	1469	1510
L24	1446	1283	1438	1421	1445	1241	1308	1345	1336	1382	1377	1366	1278	1287	1400	1368	1395	1466	1367	1312	1412	1281	1299		1376	1269	1384	1288	1355	1327
L25	1505	1390	1395	1303	1281	1212	1354	1348	1378	1485	1405	1479	1421	1228	1514	1448	1530	1367	1433	1348	1358	1324	1363	1376		1271	1449	1367	1372	1396
L26	1353	1561	1363	1285	1264	1510	1523	1477	1453	1455	1382	1335	1439	1435	1309	1452	1323	1285	1378	1543	1376	1536	1480	1269	1271		1569	1426	1566	1452
L27	1482	1588	1396	1354	1426	1459	1602	1556	1484	1525	1595	1474	1508	1531	1381	1660	1437	1440	1479	1556	1457	1484	1464	1384	1449	1569		1508	1655	1488
L28	1410	1509	1382	1359	1389	1464	1615	1565	1520	1507	1495	1429	1524	1514	1536	1416	1491	1436	1474	1483	1421	1596	1489	1288	1367	1426	1508		1505	1543
L29	1468	1654	1415	1352	1369	1493	1517	1617	1521	1440	1519	1394	1533	1533	1481	1569	1403	1365	1424	1506	1463	1540	1469	1355	1372	1566	1655	1505		1489
L30	1463	1426	1463	1381	1412	1434	1619	1455	1479	1425	1395	1474	1421	1511	1457	1375	1518	1435	1611	1480	1395	1535	1510	1327	1396	1452	1488	1543	1489	
MEDIAS	1434	1451	1386	1378	1388	1398	1491	1458	1450	1430	1423	1407	1415	1424	1455	1430	1452	1404	1458	1421	1398	1465	1458	1353	1382	1416	1496	1471	1482	1461

Figura 4.3: Resultado de puntuaciones

MEDIAS	1434	1451	1386	1378	1388	1398	1491	1458	1450	1430	1423	1407	1415	1424	1455	1430	1452	1404	1458	1421	1398	1465	1458	1353	1382	1416	1496	1471	1482	1461
	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19	D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D30
Desconocida	1531	1562	1521	1495	1527	1456	1488	1591	1552	1563	1558	1577	1318	1507	1524	1580	1333	1548	1585	1607	1518	1499	1057	1581	1539	1508	1518	1436	1565	1459

Figura 4.4: Comparación de las medias

Dada la baja precisión inicial obtenida (Tabla 4.1), especialmente en las secuencias más largas como en el caso de DinerDash (alrededor de 25.000), se ha supuesto que la actividad maliciosa podría estar localizada en un corto rango inicial de llamadas. Por ello, comparar las secuencias completas podría enmascarar esa variación de comportamiento al inicio. Además, como la cantidad de llamadas al sistema varía en una pequeño rango dependiendo de la ejecución, supone que el alineamiento global trabaja sobre vectores de distinta magnitud lo que merma su efectividad. Así, se decide limitar las secuencias a analizar a las 2000 primeras llamadas al sistema generadas en cada ejecución, repitiéndose la experimentación. En el caso de aplicaciones muy sencillas con pocas llamadas, se ha limitado a 1000

Tabla 4.1: Primeros resultados sin optimizar el sistema

	Muestra legítima	**Muestra malware**	**Detección legítima**	**Detección malware**	**Falsos negativos**	**Falsos positivos**	**Precisión**
DailyMoney	47	47	68 %	100 %	32 %	0 %	84 %
DinerDash	36	36	27 %	58 %	73 %	42 %	43 %

Además, los valores aplicados en el alineamiento tienen como consecuencia que el vector de medias presente una desviación típica elevada. Para evitar esto se propone puntuar con -1 cuando se produce un hueco, con 0 cuando dos elementos no coinciden y +1 cuando sí.

Cabe mencionar la importancia de automatizar todo este proceso. Para ello se ha creado un archivo apk que ejecuta una aplicación en el dispositivo y extrae las llamadas al sistema en ficheros txt. Posteriormente, estos ficheros se incluyen en el servidor en una carpeta determinada, dependiendo de su naturaleza (legítima o desconocida). Se ha desarrollado un script que a partir de los datos de esas carpetas desarrolla las tres etapas del proceso. En primer lugar convierte los datos en bruto en secuencias de números. Tras eso compara las secuencias resultantes mediante alineamiento global. Por último, lleva a cabo la prueba estadística de Wilcoxon creando un fichero con la conclusión final.

4.4. Resultados

Una vez ejecutado el algoritmo de alineamiento de secuencias con las correcciones que se han explicado anteriormente, se puede observar en la Figura 4.5 que las puntuaciones obtenidas al cruzar ejecuciones legítimas con las de referencia, distan de forma muy significativa de las puntuaciones obtenidas con las ejecuciones maliciosas.

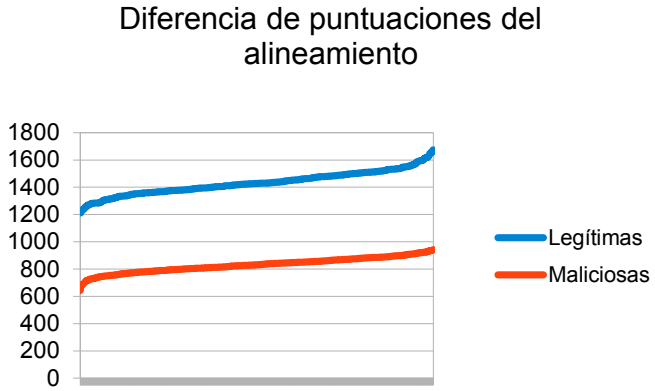

Figura 4.5: Diferencia de puntuaciones

De manera formal, se aplica la prueba de Wilcoxon sobre las medias de las muestras de referencias y las puntuaciones obtenidas de cada una de las muestras desconocidas. En función del parámetro p y de la diferencia de medias (*mean difference*) que se obtienen gracias a esta prueba, se determina si una muestra desconocida es legítima o no.

Las muestras maliciosas deben tener una diferencia de medias positiva (ya que las puntuaciones legítimas de referencia serán mayores que las maliciosas) y $p \leq 0.001$. Siguiendo este método se ha conseguido que las 40 muestras maliciosas que se han tomado de cada una de las 4 aplicaciones sean siempre catalogadas de forma correcta, logrando un acierto del 100 % en la detección de muestras maliciosas.

En cambio, en las muestras legítimas se comprueba el valor de p, que debe ser $p > 0.001$ para que se considere legítima. Si esto no se cumple se comprueba el valor de su media que puede ser positiva o negativa. Si es negativa (es decir, las puntuaciones de la muestra desconocida son superiores que las de referencia) se puede asumir que es legítima. Sin embargo, si la diferencia de medias es positiva no se puede asegurar que el comportamiento sea el correcto.

De esta forma se obtienen los resultados de la Tabla 4.2. Es importante destacar que aunque exista una tasa de falsos positivos, dado que la detección de *malware* es completa,

una segunda ejecución de la aplicación analizada podría clarificar los resultados puesto que la posibilidad de que se produzca dos veces seguidas un falso positivo es improbable.

Tabla 4.2: Resultados

	Muestra legítima	**Muestra malware**	**Detección legítima**	**Detección malware**	**Falsos positivos**	**Falsos negativos**	**Precisión**
DroidKungFu							
Basketball shot	50	50	96 %	100 %	4 %	0 %	98 %
RoboticSpaceRock	50	50	100 %	100 %	0 %	0 %	100 %
TiltMazes	50	50	90 %	100 %	10 %	0 %	95 %
Scrambled Net	50	50	94 %	100 %	6 %	0 %	97 %
Solitaire	50	50	94 %	100 %	6 %	0 %	97 %
DailyMoney	47	47	98 %	100 %	2 %	0 %	99 %
DinerDash	36	36	84 %	100 %	16 %	0 %	92 %
Jaro	36	36	92 %	100 %	8 %	0 %	96 %
Plankton							
BestVoiceChanger	50	50	100 %	100 %	0 %	0 %	100 %
SuperTouchDown	50	50	96 %	100 %	4 %	0 %	98 %
Geinimi							
Helix	50	50	100 %	100 %	0 %	0 %	100 %
Monkey Jump 2	50	50	90 %	100 %	10 %	0 %	95 %
Tap and Furious	50	50	92 %	100 %	8 %	0 %	96 %
Ginmaster							
Whites Out	50	50	98 %	100 %	2 %	0 %	99 %
Cogos							
Fruits Maching	50	50	92 %	100 %	8 %	0 %	96 %
jSMSHider							
Mash	49	49	82 %	100 %	18 %	0 %	91 %
Vdloader							
Plumber	50	50	90 %	100 %	10 %	0 %	95 %
Gapev							
Sanity	50	50	96 %	100 %	4 %	0 %	98 %
Gamex							
Z-test	50	50	88 %	100 %	12 %	0 %	94 %

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

Finalizado el proyecto y después de haber expuesto el contexto social por el que es necesario, los estudios relacionados y la propuesta realizada, es el momento de hacer una evaluación general.

Creemos que se han cumplido los objetivos marcados al principio del proyecto. Se ha demostrado que la actividad maliciosa de una aplicación móvil deja gran parte de su huella al inicio de su ejecución y que estudiar las llamadas al sistema únicamente durante ese instante consigue identificarlo. Esto permite evitar una monitorización continuada y además desliga el análisis de la actividad que pueda realizar el usuario. De esta forma se puede hacer un análisis que no requiera una gran cantidad de recursos y que sea rápido para evitar lo antes posible cualquier perjuicio que el *malware* pudiera causar.

En definitiva, creemos que el trabajo realizado ha merecido la pena y que se ha demostrado la contribución del sistema en la investigación de una identificación temprana de *malware*. Sin embargo, de forma crítica reconocemos que el proyecto tiene aún otros caminos que recorrer para convertirse en una herramienta usable tanto a nivel particular como empresarial.

5.2. Trabajo Futuro

Como posibles trabajos futuros pueden señalarse los siguientes:

- **Estrategia de detección basada en el mal uso.** En este caso es interesante que la primera etapa de la experimentación utilice como referencia diversas muestras de aplicaciones infectadas con el mismo *malware*. Esto permite, no sólo conocer si una

aplicación es maliciosa o no, sin necesidad de tener una versión legítima de la misma, sino además saber cuál es el *malware* del que está infectada.

- **Creación de un modelo global.** El estudio de una gran cantidad de aplicaciones puede derivar en la creación de un modelo único de comportamiento legítimo con el que se compararse cada aplicación desconocida. De esta forma se elimina la limitación por la cual cada aplicación solo puede compararse consigo misma en su versión legítima.

- **Búsqueda de patrones.** El uso de alineamiento global permite estudiar en su totalidad las cadenas similares resultantes de una ejecución corta. Sin embargo debe contemplarse aplicar a su vez alineamiento local que permita identificar subsecuencias repetidas, relacionadas con un comportamiento malicioso, dentro de la cadena mayor.

- **Estudio de llamadas al sistema.** Un estudio exhaustivo de las llamadas al sistema en Linux permite conocer el grado de semejanza de las acciones que llevan a cabo. Con esta información se puede completar la matriz de similitud necesaria para el alineamiento de secuencias con puntuaciones más precisas, que dieran lugar a un análisis más exacto.

Capítulo 6

Contribuciones

Participación Inés Heras

En este proyecto se ha intentado repartir la carga de trabajo de forma equitativa.

En un principio se empezó por buscar y analizar la documentación de los proyectos relativos a este asunto (capítulos 1 y 2), labor que se ha realizado de forma conjunta entre los dos integrantes del equipo. Una vez acabada esta tarea, se procedió a buscar diversas herramientas que podrían ser utilizadas en el proyecto (las detalladas en el capítulo 3). Tras consultar con miembros del GASS, entre los dos llegamos a la conclusión de que el alineamiento de secuencias podría ser una buena opción.

Entonces procedí a buscar más información sobre los tipos de alineamientos, encontrando varias implementaciones de dicho algoritmo que posteriormente mi compañero adaptó a las necesidades del proyecto.

También nos dimos cuenta de que el procesamiento de las llamadas era necesario, por lo que realicé un estudio sobre las llamadas que existen en Android e implementé un programa que hace corresponder cada llamada con un número, y además elimina las llamadas duplicadas.

Además, en el mes de noviembre yo personalmente encontré el repositorio Android Malware Genome, administrado por un equipo de la Universidad de Carolina del Norte, y nos pusimos en contacto con ellos para que nos lo facilitaran. Sin embargo, no fue hasta 5 meses después cuando lo tuvimos disponible.

También ha sido complicada la parte de conocimiento y control de la herramienta

Strace. En un primer momento se hacían las extracciones de forma manual, por lo que hubo que hacer una ardua tarea de investigación para saber cuál era la forma de automatizar ese proceso. Tras varias semanas de búsqueda sobre el asunto y después de recopilar entre ambos toda la información, mi compañero implementó una apk que realiza esta tarea.

Una vez tuvimos muestras de malware disponibles, el alineamiento implementado y la apk para la extracción de los datos, procedimos a obtener los resultados del alineamiento. En este punto fue necesaria una labor de documentación y recolección y análisis de datos en la que he estado muy implicada.

Tras concretar algunas de las mejoras que fueron necesarias para que el sistema fuese óptimo, hice investigaciones sobre distintos tipos de pruebas estadísticas, hasta que llegué a la conclusión de que era la prueba de Wilcoxon la que mejor se ajustaba al tipo de datos con los que estábamos trabajando.

Por último, la fase de experimentación se ha hecho de manera totalmente coordinada entre los dos integrantes del equipo. Yo investigué las más de 6 mil muestras de malware de las que disponíamos, eligiendo las 19 con las que finalmente hemos podido trabajar. Posteriormente hicimos la extracción los datos de las llamadas y realizamos todo el proceso del alineamiento y la prueba de Wilcoxon.

En cuanto a la labor de documentación y creación de la memoria, se ha hecho de forma progresiva y conjunta, volcando los conocimientos que íbamos adquiriendo sobre todos los puntos a lo largo de todo el proceso. Yo me he implicado especialmente en la redacción de los puntos 1, 3 y 4, así como en la creación de las tablas e imágenes y del manejo de la herramienta LATEX.

Participación Diego Sierra

El trabajo se ha realizado de forma conjunta y equitativa en cada uno de los puntos que lo conforman. Estos son, el estudio del contexto tecnológico, los trabajos relacionados y las herramientas de detección, además del desarrollo de la herramienta junto con la experimentación pertinente.

De forma más detallada se explican los temas en los que tengo una mayor participación dentro de cada apartado.

El inicio de mi trabajo consistió en un estudio global de los dispositivos inteligentes. De esta forma me informé sobre sus características y tendencias de mercado para entender el interés de los desarrolladores de malware sobre ellos y decidir en qué segmento se debía desarrollar nuestro trabajo.

A continuación, comenzó el estudio del malware en dispositivos inteligentes y mi primera aproximación consistió en analizar los métodos de distribución y sus técnicas para escalar privilegios.

Una vez hecho esto, continué analizando la política de seguridad que había adoptado cada sistema operativo móvil de entre los más populares y sus características particulares.

Posteriormente realicé un estudio profundo de las estrategias de detección presentes en los trabajos relacionados con esta materia. De esta forma se concluyeron las características del malware en las que se centran otros desarrolladores para su detección.

Una vez obtuve la clasificación habitual para estrategias de detección, realicé un análisis superficial de entre 10 y 20 proyectos de temática similar a la nuestra y un análisis profundo de los que más relacionados estaban con nuestra propuesta.

En relación a los métodos de análisis llevé a cabo un estudio profundo sobre la lógica necesaria y el funcionamiento del alineamiento de secuencias como sistema para comparar dos cadenas, en nuestro caso relacionadas con el malware estudiado. Además hice un estudio superficial sobre las máquinas de vector soporte y los motivos por los que no se ajustaban a nuestros objetivos.

También me encargué de un estudio detallado de la prueba estadística U-test.

De esta forma fue labor de mi compañera el estudio de otros métodos como las cadenas de Markov o las redes Bayesianas. De igual forma ella estudió de forma detallada la prueba estadística Wilcoxon.

Una vez estudiadas en conjunto las herramientas que se iban a usar, estas debían ser implementadas. Me encargué de desarrollar el programa móvil que ejecutara la aplicación que se quería analizar y que además extrajera las llamadas con las que se comunicaba con el núcleo del sistema.

Fue labor de mi compañera desarrollar el programa que trataba las llamadas al sistema y las convertía en una secuencia de números con los que pudiéramos trabajar.

Posteriormente una vez obtenida esta información, me encargué de desarrollar el programa que realizara el alineamiento de secuencias Needleman-Wunsch sobre las cadenas de números para que devolviera una puntuación.

Además, mediante la ayuda de librerías externas desarrollé el programa que aplicaba la prueba estadística Wilcoxon sobre los resultados del alineamiento para dar una valoración final.

Por último implementé un sistema que automatizara el proceso de detección desde que se extraían las llamadas al sistema del dispositivo móvil, con el fin de que ejecutara todo el análisis mediante un solo comando.

La experimentación inicial se hizo de forma equitativa hasta decidir los parámetros con los que el sistema obtenía mejores resultados. Una vez determinados fue labor de mi compañera el estudio de los dataset que poseíamos para poder obtener las aplicaciones con las que íbamos a trabajar.

En posesión de estas aplicaciones en general fue mi labor la extracción de las llamadas al sistema que realizaban sobre Android y fue la suya el aplicarlas el proceso de análisis para llegar a los resultados.

Bibliografía

[1] Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2012–2017, 2013.

[2] Mariantonietta La Polla, Fabio Martinelli, and Daniele Sgandurra. A Survey on Security for Mobile Devices. *Communications Surveys & Tutorials, IEEE*, 15(1):446–471, 2013.

[3] Bluetooth-Worm: SymbOS/Cabir, 2004.

[4] Juniper Networks Global Threat Center. Malicious Mobile Threats Report 2010/2011, 2011.

[5] Ali Feizollah, Nor Badrul Anuar, Rosli Salleh, and Ainuddin Wahid Abdul Wahab. A Review on Feature Selection in Mobile Malware Detection. *Digital Investigation*, 13:22–37, 2015.

[6] Christos Xenakis and Lazaros Merakos. Vulnerabilities and Possible Attacks Against the GPRS Backbone Network. In *Critical Information Infrastructures Security*, pages 262–272. Springer, 2006.

[7] Radmilo Racic, Denys Ma, and Hao Chen. Exploiting MMS Vulnerabilities to Stealthily Exhaust Mobile Phone's Battery. In *Proceedings of the Second International Conference on Security and Privacy in Communication Network*, pages 1–10. IEEE, 2006.

[8] Liang Cai, Sridhar Machiraju, and Hao Chen. Defending Against Sensor-Sniffing Attacks on Mobile Phones. In *Proceedings of the 1st ACM workshop on Networking, systems, and applications for mobile handhelds*, pages 31–36. ACM, 2009.

[9] Nan Xu, Fan Zhang, Yisha Luo, Weijia Jia, Dong Xuan, and Jin Teng. Stealthy video capturer: a new video-based spyware in 3g smartphones. In *Proceedings of the Second ACM Conference on Wireless Network Security*, pages 69–78. ACM, 2009.

[10] Guillermo Suarez-Tangil, Juan E Tapiador, Pedro Peris-Lopez, and Arturo Ribagorda. Evolution, Detection and Analysis of Malware for Smart Devices. *IEEE Communications Surveys & Tutorials*, 16(2):961–987, 2014.

[11] Lucas Davi, Alexandra Dmitrienko, Ahmad-Reza Sadeghi, and Marcel Winandy. Privilege Escalation Attacks on Android. In *Information Security*, pages 346–360. Springer, 2011.

[12] Kari Kostiainen, Elena Reshetova, Jan-Erik Ekberg, and N Asokan. Old, New, Borrowed, Blue: A Perspective on the Evolution of Mobile Platform Security Architectures. In *Proceedings of the First ACM Conference on Data and Application Security and Privacy*, pages 13–24. ACM, 2011.

[13] Aubrey-Derrick Schmidt. *Detection of Smartphone Malware*. PhD thesis, Berlin Institute of Technology, 2011.

[14] James OConnor. Blackberry Security: Ripe for the Picking? *Symantec White Paper*, 111, 2006.

[15] Microsoft Corporation. Windows Phone 8 Security Overview, 2012.

[16] Kathy Yee Au Wain, Yi Fan Zhou, Zhen Huang, Phillipa Gill, and David Lie. A Look at Smartphone Permission Models. In *Proceedings of the 1st ACM workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 63–68. ACM, 2011.

[17] Lok-Kwong Yan and Heng Yin. DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis. In *Proceedings of the 21st USENIX Conference on Security Symposium*, pages 569–584, 2012.

[18] Thomas Bläsing, Leonid Batyuk, Aubrey-Derrick Schmidt, Seyit Ahmet Camtepe, and Sahin Albayrak. An Android Application Sandbox System for Suspicious Software Detection. In *Proceedings of the 5th International Conference on Malicious and Unwanted Software*, pages 55–62. IEEE, 2010.

[19] iOs Security Guide.

[20] D. G. Suarez Tangil. Mining Structural and Behavioral Patterns in Smart Malware. Phd. thesis, Universidad Carlos III de Madrid, 2014.

[21] Anthony Desnos and Geoffroy Gueguen. Android: From Reversing to Decompilation. *Black Hat Abu Dhabi*, pages 77–101, 2011.

[22] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. Droidmat: Android Malware Detection Through Manifest and API Calls Tracing. In *Proceedings of the Seventh Asia Joint Conference on Information Security*, pages 62–69. IEEE, 2012.

[23] 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 *Proceedings of the Annual Symposium on Network and Distributed System Security*, 2014.

[24] Zhibo Zhao and Fernando C. Colon Osono. TrustDroid: Preventing the Use of SmartPhones for Information Leaking in Corporate Networks Through the Used of Static Analysis Taint Tracking. In *Proceedings of the 7th International Conference on Malicious and Unwanted Software (MALWARE)*, pages 135–143. IEEE, 2012.

[25] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. Andromaly: A Behavioral Malware Detection Framework for Android Devices. *Journal of Intelligent Information Systems*, 38(1):161–190, 2012.

[26] Gianluca Dini, Fabio Martinelli, Andrea Saracino, and Daniele Sgandurra. MADAM: A Multi-level Anomaly Detector for Android Malware. In *Proceedings of the 6th International Conference on Mathematical Methods, Models and Architectures for Computer Network Security*, volume 12, pages 240–253. Springer, 2012.

[27] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. Crowdroid: Behavior-Based Malware Detection System for Android. In *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, pages 15–26. ACM, 2011.

[28] Lei Liu, Guanhua Yan, Xinwen Zhang, and Songqing Chen. Virusmeter: Preventing your cellphone from spies. In *Proceedings of the Recent Advances in Intrusion Detection*, pages 244–264. Springer, 2009.

[29] Rahul Pandita, Xusheng Xiao, Wei Yang, William Enck, and Tao Xie. WHYPER: Towards Automating Risk Assessment of Mobile Applications. In *Proceedings of the 22nd USENIX Security Symposium*, volume 13, 2013.

[30] Peter Teufl, Michaela Ferk, Andreas Fitzek, Daniel Hein, Stefan Kraxberger, and Clemens Orthacker. Malware Detection by Applying Knowledge Discovery Processes to Application Metadata on the Android Market (Google Play). *Security and Communication Networks*, 2013.

[31] Microsoft Corporation. Geinimi, Sophisticated New Android Trojan Found in Wild, 2012.

[32] VirusShare.

[33] Yajin Zhou and Xuxian Jiang. Dissecting Android Malware: Characterization and Evolution. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 95–109. IEEE, 2012.

[34] VirusTotal.

[35] Federico Maggi, Andrea Valdi, and Stefano Zanero. AndroTotal: A Flexible, Scalable Toolbox and Service for Testing Mobile Malware Detectors. In *Proceedings of the Third ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*, pages 49–54. ACM, 2013.

[36] Norman Fenton and Martin Neil. Combining Evidence in Risk Analysis Using Bayesian Networks. *Agena White Paper W.*, 704:8–13, 2004.

[37] Gely P. Basharin, Amy N. Langville, and Valeriy A. Naumov. The Life and Work of A. A. Markov. *Linear Algebra and its Applications*, 386:3–26, 2004.

[38] Marti A. Hearst, Susan T. Dumais, Edgar Osman, John Platt, and Bernhard Scholkopf. Support Vector Machines. *IEEE Intelligent Systems and their Applications*, 13(4):18–28, 1998.

[39] Thomas D Schneider and R Michael Stephens. Sequence Logos: A New Way to Display Consensus Sequences. *Nucleic Acids Research*, 18(20):6097–6100, 1990.

[40] Sara A Shehab, Arabi Keshk, and Hany Mahgoub. Fast Dynamic Algorithm for Sequence Alignment Based on Bioinformatics. *International Journal of Computer Applications*, 37(7):54–61, 2012.

[41] Tahir Naveed, Imitaz Saeed Siddiqui, and Shaftab Ahmed. Parallel Needleman-Wunsch Algorithm for Grid. In *Proceedings of the PAK-US International Symposium on High Capacity Optical Networks and Enabling Technologies*, 2005.

[42] M. Zhao, W. P. Lee, E. P. Garrison, and G. T. Marth. SSW Library: An SIMD Smith-Waterman C/C++.

[43] Frank Wilcoxon. Individual Comparisons by Ranking Methods. *Biometrics Bulletin*, pages 80–83, 1945.

[44] Simon J. Mason and N. E. Graham. Areas Beneath the Relative Operating Characteristics (ROC) and Relative Operating Levels (ROL) Curves: Statistical Significance and Interpretation. *Quarterly Journal of the Royal Meteorological Society*, 128(584):2145–2166, 2002.

[45] Pedro Garcia-Teodoro, J. Diaz-Verdejo, Gabriel Maciá-Fernández, and Enrique Vázquez. Anomaly-Based Network Intrusion Detection: Techniques, Systems and Challenges. *Computers & Security*, 28(1):18–28, 2009.

[46] Juan M. Estevez-Tapiador, Pedro Garcia-Teodoro, and Jesus E Diaz-Verdejo. Anomaly Detection Methods in Wired Networks: A Survey and Taxonomy. *Computer Communications*, 27(16):1569–1584, 2004.

Parte I

Resumen en Inglés

Apéndice A

Introduction

In recent years it has increased the occurrence of different types of smart devices. Thus, between 60 and 80 percent of the population in the countries where there is access to the Internet, made via Smartphones or Tablets [1], which are used an average of two hours a day. In addition, it is estimated that in 2017 there in the world 1.4 mobile devices per capita and sales will exceed those of PCs. In light of these data, it is clear that the interest and the importance of these devices in society is increasingly noticeable and presumably will continue to advance rapidly.

Among the characteristics of mobile devices is the presence of sensors such as gyroscopes, microphones, GPS locators, etc.; the possibility to connect to any type Bluetooth, Wi-Fi telephone network; or the ability to acquire and use applications developed by third parties [2]. But despite the obvious advantages, they also have certain problems that can become crucial for safety. This is because these devices store very sensitive and varied information which may compromise the security, privacy and even the economy of the owner or third parties. The sensors that incorporate these technologies can also collect data without the user being really aware of how much information the device being managed. It is for this reason that the creation of malicious software or malware, specific for this area has increased at the same speed to use.

Finally, it is important to note that among users there is no awareness of the danger they are exposed to a potential attack. The vast majority even unknown mobile malware exists, and that applications installed on their devices may exhibit malicious behavior . Therefore, it is necessary to raise security models beyond the user action, and controlling devices markets.

A.1. Mobile Malware

Malware is any type of software or code hostile, intrusive, or a designed for use the device without the owner's knowledge. The development and proliferation of mobile malware is closely linked to increased network capabilities and computing resources. So, the first evidence that mobile malware was developed in 2004 to attack Symbian [3] devices. However, it is from 2010 when it begins to grow significantly [4], being mainly dedicated to Android and iOS systems. Only for Android in 2012 it is estimated that there were about 35,000 malware, being objective of 79 % of malware generated during this year compared to 11.5 % in 2010 [5].

To analyze the attacks and their types, should be considered three perspectives: performance and objectives, malware distribution forms and methods of acquiring privileges.

A.1.1. Behavior and Objectives of the Attack

In most cases the malware has several objectives and also their purpose and behavior can vary by a remote command sent by those who control it. The most common of its objectives is the economic benefit. In this sense several examples can be found, as in the case of fraud or overbilling. This attack is to charge expenses to the account of the victim (which are transferred to the attacker) through calls or SMS sent to premium rate numbers without the user's consent. Usually, the user associated with such attacks a conflict with the company supplying the service, ignoring its true nature [6].
Another type of attack is the denial of service or sabotage, called DoS attack that tries to consume the device battery operating time limit or deny the user access to certain resources or networks [7]. When running in a distributed way (DDoS), may cause damage not only to one or more users, but also to various organizations [2]. For example, the Internet server of an area can be collapsed if they begin sending packets to a massive network.

Finally, the privacy could be compromised through a technique called sniffing [8], which leverages the data collected by sensors and networks which uses the device can obtain images [9], telephone recordings, passwords, emails, bank details or any information the device sends or receives. This is one of the most concern to users or organizations threats, as on mobile devices and sensitive data are stored and classified. In Fig. A.1 the occurrence of each type of target, for each operative system is detailed.

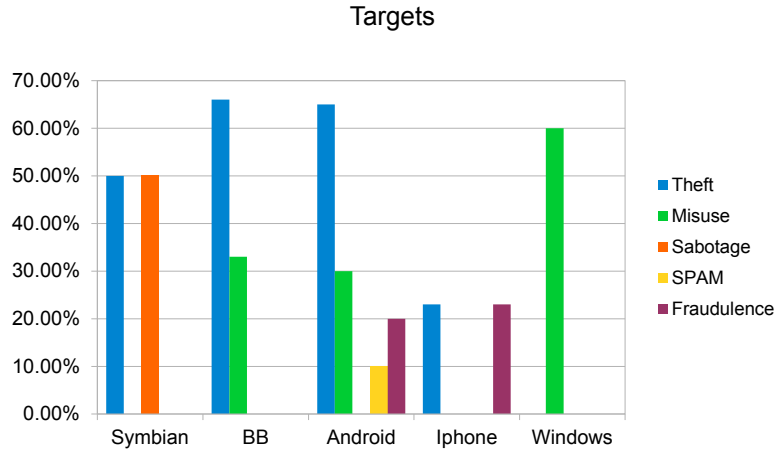

Figure A.1: Classification of attacks per operating system

A.1.2. Distribution Methods

It is important to know the means by which malicious programs are distributed in order to tackle the contagion. There are two major approaches [10]: self-propagation and social engineering . The first type are grouped the following methods: Application to Device, A2D, the malicious code is in an application that infects the Device; SMS to Device, S2D, the device is spread by SMS; USB to Device, U2D; Network to Device, N2D; Device to Device, D2D y Cloud to Device, C2D. In these cases the malware is programmed to contaminate the device when placed in contact with it.

On the other hand, social engineering, group methods Market to Device, M2D, the user downloads a malicious program from the market or Web -browser to Device, W2D.

As shown in Fig. A.2, social engineering is the main method of distributing malware. Possibly this is due to poor awareness of the danger that show users; They are usually those who download infected apps from the markets. It is also often a combination of both strategies.

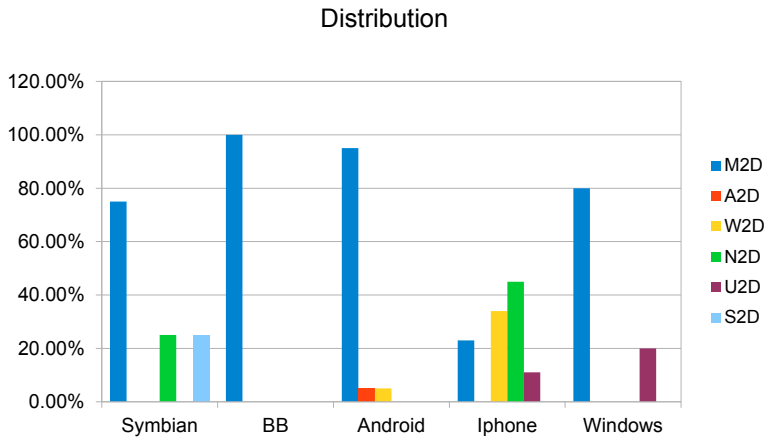

Figure A.2: Distribution methods per operating system

A.1.3. Acquisition of Privileges

For the malware to be effective is not enough to spread the device, it must also get a number of privileges to gain access to critical parts of the system and make the necessary actions to implement its work. Usually, these privileges are granted directly by users [10] to install applications on legitimate appearance but hide malicious behavior. Users are unaware of the impact of transferring certain permissions to unknown programs and sometimes not even pay attention to requests for authorization. Therefore, this method is highly effective, as shown in Fig. A.3.

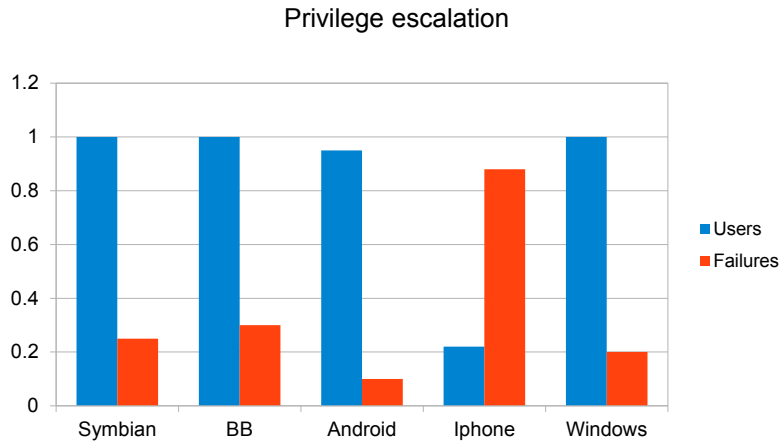

Figure A.3: How privileges are aquired

Another method based on the technology involves exploiting vulnerabilities or configuration errors platform [11]. Rootkits are used to detect how these errors can be exploited. This type of malware infects the operating system, so it can be considerably dangerous and also leave the door open to future infections.

A.2. Current Security Models

Today, the official application distribution markets have different mechanisms to try to ensure the safety of them. Mainly used verification tests to check the legitimacy of the application code. However, detecting malware is too complex; not known in detail the inner workings of these tests and the presence of a considerable number of malicious applications is insufficient evidence method. In addition, they should take into account the downloads from unofficial markets that do not have any safety filter and are highly dangerous.

From the point of view of the platform, a way of trying to ensure security is to restrict communication between applications and the actions they can perform, including access to data and services. Technique to isolate the execution of the application in controlled environments is also proposed, called sandboxing.

Fig. A.4 shows that Android is the mobile operating system that dominates, and a tendency to continue to grow [5]. iOS is the second best-selling, though at a distance of Android, and the rest is increasingly less important in sales of devices and applications.

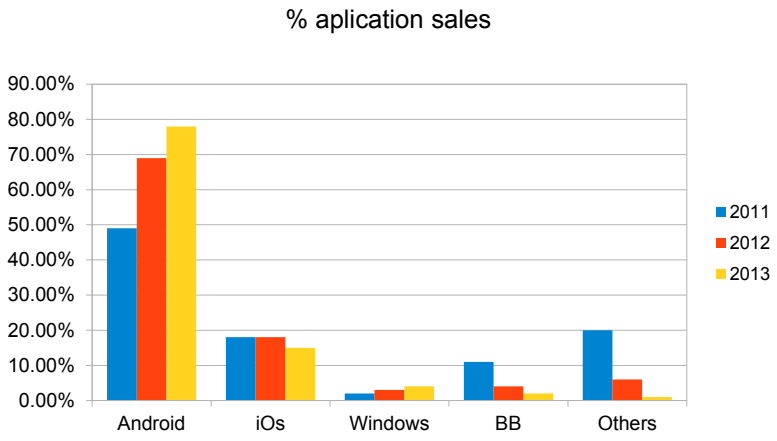

Figure A.4: Sales of applications per type of operating system

Then, methods and decisions adopted by the five major operating systems that dominate are analyzed.

A.2.1. Symbian

The Symbian security model is based on a basic set of permissions to control the device's resources. Applications run in user space while the operating system at the kernel space.

Applications that require access to protected libraries and therefore are more vulnerable, must be signed with a certificate from Symbian [12], while all others can be self-signed, not having access to these dangerous bookstores. In this way applications are already controlled and security of markets becomes unnecessary, being almost nonexistent. To improve the security of Symbian has proposed the use of machine learning algorithms to analyze the behavior of applications [13]. Despite its high reliability, Symbian is the least used operating systems.

A.2.2. BlackBerry

BlackBerry security is based on a comprehensive scheme of permissions that has proven very safe [14]. Initially, applications have very limited access to resources and, subsequently, can be allowed to escalate their privileges. Manufacturer firm to access libraries, providing basic protection for system processes and memory is required. This is probably the safest of which have been discussed, but has the disadvantage of being also very close, with a very limited market system.

A.2.3. Windows Mobile

Microsoft [15] based security model validation developers and the reputation and value of each application. In the latest versions, each application runs in its own sandbox, which permits those who have been given authorization, similar to how it relates to the Android system are granted. However, these permissions are requested in the installation and must be granted by users, who tend to disregard the importance of them. It is also very difficult to change at runtime [16].

A.2.4. Android

The Android security scheme focuses mainly on the device itself, since users are allowed to download applications from any platform or market distribution. The format permits

results in a manifesto that should be authorized by the user during installation and awarded at runtime. However, due to user tolerance regarding permitting, this technique is ineffective.

Android also uses sandboxing technique [17] [18], isolating each application on their own machine Dalvik. This generates a bytecode and gives each a different user ID, with the exception of the applications from the same developer, which share identifier, allowing them to share resources. However, applications can share information between them explicitly using an interface for communication.

It is interesting to focus on this particular system because it is the most used, and also suffer more infections.

A.2.5. iOS

Unlike Android, Apple turns its security to the market because it limits the ability to download to their own [19]. Therefore, applications and developers need a signature for verification by a certificate issued by Apple. However, details of the verification tests are not public, so its effectiveness is difficult to assess.

Usually, security at the platform level is weaker than in Android, or virtually nonexistent, because applications run on a single, common to all isolated environment, besides having access to most of the resources of the device. Therefore, in recent models it is trying to extend this system, controlling the traffic of personal data.

Apéndice B

Conclusions and Future Work

B.1. Conclusions

Ended and after discussing the social context for which it is necessary, related studies and proposal made, it's time to make a general assessment.

We believe that they have met the objectives set at the beginning of the project. It has been shown that malicious activity in a mobile application leaves much of a mark at the beginning of its implementation and to study the system calls only during that time gets identified. This prevents continuous monitoring of system calls and also detaches the analysis of the activity that the user can perform. This way can be make an analysis that does not require a lot of resources and it quickly as soon as possible to avoid any damage that malware might cause.

The project has worked to automate the process and a lot of applied technologies. Since Java for Android application, to a script in a Linux context that automates the process, through programming in C and C ++ for global alignment or the Wilcoxon test.

Also, the documentation work done in a totally unknown subject for the development team as it was at the beginning of academic year malware on mobile devices.

This work has been valuable, if not essential, to achieve the objectives set. Noteworthy in this regard the aid received by the various members of the Analysis Group, Security and Systems (GASS, http://gass.ucm.es) Department of Software Engineering and Artificial Intelligence (DISIA) of the Faculty Computing of the Universidad Complutense de Madrid (UCM), which have provided valuable information on the methods and valid research sources.

In short, we believe that the work has paid off and has demonstrated the system's

contribution in the investigation of early identification of malware. However, critically we recognize that the project still has other ways to go to become a usable tool both owners and enterprise levels.

B.2. Future Work

Possible future work can be identified as coming next.

- **Detection strategy based on misuse.** In this case it is interesting that the first stage of experiment using different reference samples infected with the same malware applications. This allows not only to know whether an application is malicious or not, without having a legitimate version of it, but also know what the malware they are infected.
- **Creating a global model.** The study of a large number of applications can lead to the creation of a single model of legitimate behavior with each unknown application is comparable. Thus the limitation for which each application can only be compared with itself in its legitimate version is removed.
- **Finding patterns.** Using global alignment allows to study in full the similar chains resulting from a short run. However it must be seen to apply in turn to identify local alignment repeated subsequences related to malicious behavior within the larger chain.
- **Study of system calls.** A comprehensive study of the system calls in Linux allows to know the degree of similarity of the actions carried out. With this information can be completed the similarity matrix necessary for the alignment of sequences with more accurate ratings, which would lead to a more accurate analysis.