TELEMÁTICA
REVISTA DIGITAL de las Tecnologías de la Información y las Telecomunicaciones
Revista Telemática. Vol. 17. No. 2, mayo-agosto, 2018, p.24- 35 ISSN 1729-3804

DETECCIÓN DE MALWARE EN ANDROID UTILIZANDO APRENDIZAJE AUTOMÁTICO

Alejandro Pérez Fals[1], Mónica Montero Ceballos[2], Victoria Isabel Pérez Plana[3]

Segurmática, Calle Zanja 651 esquina a Soledad.
[1]alejandro@segurmatica.cu

RESUMEN

Con el constante incremento del uso de dispositivos móviles, la necesidad de algoritmos efectivos en la detección de malware está en constante crecimiento. La investigación que se presenta en este trabajo aplica aprendizaje automático en la detección de malware en Android. Este enfoque se basa en las llamadas API de Java y en los permisos requeridos por las aplicaciones como características para generalizar el comportamiento de estas y diferenciar si es bueno o malicioso. Para la clasificación de las muestras se utilizan los métodos de aprendizaje supervisado SVM e ID3. Los resultados de la experimentación sobre un gran conjunto de aplicaciones son alentadores.

PALABRAS CLAVE: Android Malware, Android, Aprendizaje Automático.

GUIDELINES FOR PREPARING ARTICLES FOR THE TELEMATICS JOURNAL

ABSTRACT

Based on the constant increase use of mobile devices, the need for effective algorithms in the malware detection is constantly growing. The research presented in this paper applies machine learning in Android malware detection. This approach is based on the Java API calls and the permissions required by the applications. These represent main characteristics to generalize behavior of applications to distinguish good from malicious applications. The supervised learning methods SVM and ID3 are used for the classification of samples. Results of experimentations on a large set of applications exhibits encourage outcomes.

KEYWORDS: Android, Malware Detection, Machine Learning.

INTRODUCCIÓN

Los dispositivos móviles son la primera vía de comunicación y acceso a la información para muchas personas en todo el mundo. De acuerdo al estudio realizado por la Asociación Internacional de Operadores de Móviles GSMA [1], el número de usuarios de telefonía móvil a nivel mundial superará los 5 mil millones a finales de 2017, en comparación con los 4 mil 800 millones con que cerró 2016 y serán unos 5 mil 700 millones a finales de la década. A medida que el mercado y consumo de telefonía móvil ha ido creciendo de una manera desorbitada, también ha aumentado la vulnerabilidad de sus sistemas operativos contra ataques informáticos. Por eso, la seguridad de estos dispositivos debe ser una prioridad tanto en las empresas como en el uso personal.

Existen numerosos sistemas operativos móviles en el mercado: Android, iOS y Windows Phone; pero Android es el más usado en el mundo. Su popularidad es evidente, instalado en millones de dispositivos y con miles de aplicaciones, este sistema operativo ha cautivado a millones de personas. En mayo de 2017, el número de usuarios que usa Android llegó a los 2 mil millones. Ya se usan más dispositivos Android que equipos con Windows [2]. Los ordenadores han ido cediendo gran parte del protagonismo tecnológico a los teléfonos inteligentes y tabletas, por ello, es lógico que los atacantes que antes se centraban en afectar a los ordenadores se concentren ahora también en afectar a los móviles.

En 2016, los productos de Kaspersky Lab para la protección de dispositivos móviles [3] registraron un aumento significativo en el número de instalaciones de paquetes de malware, que superó los 8 millones y medio, casi el triple que en 2015. En la actualidad, el malware móvil va en aumento y cada vez es más complejo. También detectaron más de 120 mil troyanos bancarios móviles y más de 250 mil troyanos extorsionadores [3]. Los efectos de estos ataques tienen consecuencias muy desagradables como robo de información, robo financiero o problemas en el uso de los terminales. En comparación: en el periodo comprendido de 2004 a 2013 se encontraron más de 10 millones de paquetes de instalación de malware y alrededor de 2.5 millones en el 2014.

Los avances en Ciencias de la Computación, así como las mejoras de hardware, han hecho posible que el aprendizaje automático se aplique de manera efectiva en la extracción de información de grandes conjuntos de datos y, consecuentemente, en la detección de malware en Android. Para ello se han propuesto varias soluciones, que utilizan técnicas de aprendizaje de máquina tales como SVM (Support Vector Machines), Redes Neuronales y Árboles de Decisión.

En [4] se propone la herramienta ANASTASIA, un algoritmo de aprendizaje automático que utiliza un análisis estático para la detección de malware en Android. En esta solución se extraen tantas características de las muestras como sean posibles y se experimenta con distintos algoritmos de clasificación para determinar cuál se comporta mejor.

Una solución basada en el algoritmo SVM se propone en [5], la cual integra permisos riesgosos y llamadas API vulnerables. También se hace una investigación basada en los permisos que requiere la aplicación en [6], donde se utilizan los clasificadores Naive Bayes, Random Forest y J48.

En [7] se hace un análisis dinámico y se combina SVM con Predicción Conformal para mejorar la precisión de la clasificación cuando la información no es suficiente como para identificar una sola familia de malware. También en [8] se utiliza Random Forest y experimentan con el número de árboles de

decisión, la profundidad de cada árbol y la cantidad de características obteniendo buenos resultados. La herramienta MALINE se desarrolla en [9]. Esta realiza una clasificación automática basada en llamadas al sistema.

En esta investigación se presenta un sistema de detección de malware en Android con un enfoque basado en los permisos requeridos por las aplicaciones y llamadas API de Java, de donde se obtiene información más específica acerca de las acciones que intenta ejecutar la aplicación [10]. Para el paso de clasificación de las muestras se utilizan los métodos de aprendizaje supervisado SVM e ID3.

El contenido de este trabajo está organizado de la siguiente forma: en la sección 2.1 se realiza un análisis sobre las muestras escogidas para experimentar en la investigación, en la sección 2.2 y 2.3 se explica el proceso de extracción y selección de características, en 2.4 se mencionan los métodos estudiados y más adelante en 2.5 se habla sobre los detalles de implementación. En 2.6 se presenta una validación del modelo construido y en 2.7 se exponen los resultados alcanzados.

DESCRIPCIÓN DE LA PROPUESTA

Muestras

Para la selección de las muestras nos apoyamos en el repositorio de aplicaciones de la empresa Segurmática. Este cuenta con más de 10 mil aplicaciones “sanas”, o sea, que no contienen acciones consideradas malignas, y otras 600 mil que si son malignas y que representan a más de 2220 tipos de malware de diferentes familias. Inicialmente se realizó una selección de 150 aplicaciones buenas y otras 150 aplicaciones con comportamiento malicioso tratando de que abarcaran cierta aleatoriedad.

El grupo de aplicaciones benignas fue escogido intentando que fueran diversas y reflejaran los distintos tipos de aplicaciones presentes en el mercado Android; y además que fueran proporcionales al número de muestras que existen en cada tipo de aplicación. Los aspectos que se tuvieron en cuenta para la recopilación del conjunto de muestras malignas fueron los siguientes:

- Diferentes clasificaciones de acuerdo al comportamiento de las que tienen mayor impacto: troyanos sms, descargadores y bancarios, rooteadores (adquieren privilegios de administrador y toman el control del dispositivo), extorsionadores (ransomware de bloqueo y/o cifrado), adware (software publicitario) y herramientas malignas (malware tool).
- Diferentes variantes dentro de una misma familia de programas malignos, y más de una familia (nombres según los antivirus) dentro de una clasificación según el comportamiento.
- Aplicaciones de tamaños similares y diferentes dentro de un mismo nombre y variantes de programa maligno.

Con este grupo se realizaron las pruebas iniciales que nos permitieron valorar la solución escogida.

Para un análisis más minucioso se realizó una selección de 5 335 muestras del mismo repositorio, de las cuales 3 153 son malignas y 2 202 son benignas. En este grupo también se tuvieron en cuenta las condiciones anteriores. Además, para el conjunto de muestras con comportamiento malicioso se estableció que hubiera al menos una de cada variante por cada familia presente en el repositorio; y que la cantidad representativa de cada variante en la selección fuera proporcional a la cantidad presente en el repositorio. Con este grupo se realizó la experimentación final y estos son los resultados que se presentan en el epígrafe 3 del documento.

Extracción de características

Para proteger a los usuarios de Android de las graves amenazas de malware se han propuesto varias soluciones. En muchas de estas la extracción de características está basada en los permisos requeridos por la aplicación. Sin embargo, los mecanismos que están basados solo en permisos fallan por diversas causas [11]:

- La existencia de un cierto permiso en el archivo AndroidManifest.xml de la aplicación no necesariamente significa que realmente se use dentro del código.
- Gran cantidad de los permisos solicitados, sobre todo los críticos, no se utilizan dentro del código de la aplicación en sí, sino que son requeridos por los paquetes de publicidad.
- El malware puede tener un comportamiento malicioso sin requerir ningún permiso [12].

El objetivo de este trabajo es superar las deficiencias de un mecanismo basado solamente en permisos y crear un clasificador para aplicaciones de Android que pueda ser usado en la detección de malware. Para ello, en la etapa de extracción de características se decidió combinar permisos y llamadas API.

Permisos

En el sistema Android, los permisos solicitados por la aplicación juegan un papel vital en el control de los derechos de acceso. De manera predeterminada, las aplicaciones no tienen permiso para acceder a los datos del usuario y afectar la seguridad del sistema. Durante la instalación, el usuario debe permitir que la aplicación acceda a todos los recursos solicitados por ella.
Los desarrolladores deben mencionar los permisos solicitados para los recursos en el AndroidManifest.xml. La estructura para declarar un permiso en dicho archivo se muestra en la Figura 1.

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

Figura 1: Declaración de los permisos en el An-droidManifest.xml.

Para usarlos como entrada en los algoritmos de aprendizaje automático, se procesa el archivo AndroidManifest.xml, se buscan las etiquetas uses-permission y se extrae la cadena que declara el tipo del permiso. Luego se genera un vector binario que indica si el permiso está presente o no en la aplicación analizada como muestra la Figura 2.

```
...
<uses-permission android:name="an-
droid.permission.SEND_SMS " />
<uses-permission android:name="an-
droid.permission.INTERNET " />
<uses-permission android:name="an-
droid.permission.READ_CONTACTS" />
...
```

Figura 2: Ejemplo de declaración de permisos

Llamadas API

Además de los permisos, existen varias características de las aplicaciones Android que pueden ser indicativas de su malicia como el código. Si bien el código fuente no suele siempre estar disponible, se puede conseguir mucha información desensamblando los archivos del tipo jar.

Los métodos llamados en las API de Android y Java son una de las características que se puede obtener de estos archivos. Para obtener el código desensamblado se siguieron los siguientes pasos:

- Descomprimir el archivo .apk.
- Convertir el archivo classes.dex en .jar usando la herramienta d2j-dex2jar.sh.
- Escribir el código desensamblado en un archivo usando una combinación de los comandos jar y javap.

Luego, se procesa el código y se extraen las llamadas API que se realizan para añadirlas al vector binario que se genera cuando se extraen los permisos.

La principal dificultad al obtener esta información es el hecho de que muchos desarrolladores utilicen técnicas de ofuscación, ya sea para proteger su código o para evitar este tipo de análisis. Las técnicas de ofuscación usadas comúnmente funcionan teniendo clases y métodos con una sola letra (‘a’, ‘b’, etc.) y llamando a los métodos deseados indirectamente a través de estos. En [10] se desarrolla una heurística simple para eliminar tales clases: se ignora cualquier clase con un nombre de menos de 4 caracteres o cuya clase padre tenga una sola letra.

Otra dificultad es que si se incluyen las clases de las API de Android (ej. android.app.Activity) se pierde precisión en la clasificación. Esto se debe al gran tamaño de la API (varios miles de clases) en comparación con el código que la llama, lo que conlleva a ruido en los datos. Por tal motivo, en esta investigación solo se contemplan las clases de las API de Java. Estas brindan información más específica sobre las acciones que intentan realizar las aplicaciones.

Selección de características

La selección de características es una parte fundamental del aprendizaje automático. Este paso hace referencia al proceso de reducir las entradas para su procesamiento y análisis, o de encontrar las entradas más significativas. Este paso es muy importante por diversas razones. Uno de estos motivos es que la selección de características implica cierto grado de reducción de cardinalidad para imponer un corte en el número de atributos que se tendrán en cuenta al crear un modelo [13]. El exceso de atributos puede llevar a sobreaprendizaje, pues incrementa la complejidad del modelo en relación al número de datos disponibles [14]. Además, si el conjunto de datos es de dimensiones elevadas, la mayoría de los algoritmos de minería de datos necesitarán un conjunto de datos de aprendizaje mucho más grande.

Con este paso no solo mejora la calidad del modelo, sino que también hace que el proceso de modelado sea más eficiente. Si usa columnas innecesarias al generar el modelo, se necesitará más CPU y memoria RAM durante el proceso de entrenamiento, así como más espacio de almacenamiento para el modelo completado [13].

Otro motivo la importancia de la selección de características es que los datos redundantes o poco relevantes hacen que resulte más difícil detectar patrones significativos. En el proceso de extracción de características se obtuvieron un total de 1 234 permisos y 1 998 llamadas API, lo que hacía que el vector de atributos tuviera gran dimensión y presentara los inconvenientes antes mencionados. Para resolver este problema se decidió asignarle un peso a cada atributo. Este se calcula por la diferencia entre el porciento de aplicaciones malignas y benignas que presentan una característica determinada con respecto al total de aplicaciones. De esta forma, los pesos con signo positivo corresponden a los atributos que se presentan con mayor frecuencia en las aplicaciones con comportamiento malicioso y los que tienen signo negativo corresponden a los más recurrentes en las sanas.

En esta investigación se decidió seleccionar las características donde el valor absoluto de los pesos sea mayor que un umbral determinado. De esta manera, se escogen los atributos con mayor índice de benignidad y malicia. Las tablas 1 a 4 muestran un resumen de los pesos otorgados a los atributos.

Tabla 1: Top 10 llamadas API más frecuentes en aplicaciones benignas.

#	Llamadas API	Peso
1	java/util/GregorianCalendar	-25
2	java/security/SecureRandom	-22
3	java/util/concurrent/ConcurrentLinkedQueue	-22
4	java/util/Scanner	-21
5	java/lang/IndexOutOfBoundsException	-20
6	java/lang/Number	-20
7	java/io/FilterOutputStream	-19
8	java/util/concurrent/CopyOnWriteArrayList	-19
9	java/util/logging/Logger	-18
10	java/nio/channels/FileChannel	-17

Tabla 2: Top 10 llamadas API más frecuentes en aplicaciones malignas.

#	Llamadas API	Peso
1	java/util/zip/Deflater	34
2	java/lang/Process	30
3	java/util/zip/Inflater	29

4	java/lang/InstantiationException	27
5	java/lang/ProcessBuilder	25
6	java/lang/NoSuchMethodException	25
7	java/lang/ClassNotFoundException	24
8	java/net/Proxy	23
9	java/io/FileReader	23
10	java/security/spec/PKCS8EncodedKeySpec	22

Tabla 3: Top 10 Permisos más frecuentes en aplicaciones benignas.

#	Permisos	Peso
1	com.android.vending.BILLING	-17
2	com.android.vending.CHECK_LICENSE	-9
3	com.google.android.c2dm.permission. RECEIVE	-7
4	android.permission.SET_WALLPAPER	-3
5	android.permission.USE_CREDENTIALS	-2
6	android.permission.MODIFY_AUDIO_ SETTINGS	-2
7	android.permission.BLUETOOTH_ADMIN	-2
8	android.permission.WRITE_SYNC_ SETTINGS	-2
9	android.permission.READ_SYNC_ SETTINGS	-2
10	android.permission.RECORD_AUDIO	-1

Tabla 4: Top 10 Permisos más frecuentes en aplicaciones malignas.

#	Permisos	Pes

		o
1	android.permission.SYSTEM_ALERT_ WINDOW	65
2	android.permission.RECEIVE_BOOT_ COMPLETED	65
3	android.permission.GET_TASKS	64
4	com.android.launcher.permission.INSTALL_ SHORTCUT	60
5	android.permission.SEND_SMS	60
6	android.permission.READ_PHONE_STATE	58
7	android.permission.RECEIVE_SMS	57
8	android.permission.READ_SMS	52
9	android.permission.CHANGE_WIFI_STATE	50
10	android.permission.CHANGE_NETWORK_ STATE	50

Métodos de aprendizaje automático

El aprendizaje automático o aprendizaje de máquinas1 es la rama de la Inteligencia Artificial que se dedica al estudio de los agentes/programas que aprenden o evolucionan basados en su experiencia. De forma más concreta, se trata de crear programas capaces de generalizar comportamientos a partir de una información suministrada en forma de ejemplos. En este trabajo se utilizan métodos de clasificación del aprendizaje supervisado. Los sistemas de clasificación supervisados son aquellos en los que, a partir de un conjunto de ejemplos clasificados (conjunto de entrenamiento), se asigna una clasificación a un segundo conjunto de ejemplos. En este caso los ejemplos son aplicaciones de Android y la clasificación es si es benigna o maligna.

Árboles de decisión

Este tipo de aprendizaje utiliza un árbol de decisión como modelo predictivo. Se mapean observaciones sobre un objeto con conclusiones sobre el valor final de dicho objeto [15]. Los nodos internos denotan condiciones con respecto a las variables de un problema, mientras que los nodos u hojas finales representan la decisión final del algoritmo [16]. En esta investigación, los nodos internos representan las características que se extraen de la aplicación (permisos y llamadas API) descritas anteriormente, mientras que las hojas representan la clasificación de la muestra en benigna o maligna. Los arcos representan los valores posibles del nodo padre, en este caso si está presente o no la característica.

Para este trabajo se emplea específicamente el algoritmo ID3. La idea básica de este algoritmo es determinar, para un conjunto de ejemplos dados, el atributo más importante, o sea, aquel que posea el mayor poder discriminatorio para dicho conjunto; este atributo es usado para la clasificación de la lista de objetos, basados en sus valores asociados. Para ello se apoya en técnicas matemáticas y probabilísticas e introduce el concepto de entropía, la cual es una medida de incertidumbre o de desorden, y es usado para ayudar a decidir qué atributo debe ser el siguiente en seleccionarse.

SVM

El algoritmo SVM divide un espacio n-dimensional en regiones usando un hiperplano. Este hiperplano maximiza el margen entre esas regiones o clases. Este margen es definido como la mayor distancia entre los ejemplos de las clases y se calcula en función de la distancia entre las instancias más cercanas de las regiones, que se denominan vectores de soporte [17]. Más formalmente, SVM construye un hiperplano o conjunto de hiperplanos en un espacio de dimensionalidad muy alta (o incluso infinita) que puede ser utilizado en problemas de clasificación o regresión. Una buena separación entre las clases permitirá una clasificación precisa.

En lugar de usar hiperplanos lineales, es común usar las llamadas funciones del kernel. Estas conducen a superficies de clasificación no lineales, como superficies polinomiales, radiales o sigmoideas. Pare este trabajo se utilizó SVM con un kernel gaussiano.

Detalles de implementación

La aplicación fue implementada en la plataforma Microsoft .Net 4.5 sobre el lenguaje C#. Los algoritmos de aprendizaje supervisado utilizados fueron los que aporta Accord.Net [18]. Esta plataforma combina aprendizaje automático con procesamiento de audio e imágenes y está conformada por un conjunto de bibliotecas escritas completamente en el lenguaje C#.

El proceso de extracción de características de un archivo apk puede ser lento, ya que depende de varios subprocesos tales como la extracción del archivo classes.dex y AndroidManifest.xml, la conversión de dex a jar y la extracción del código. Por este motivo se crea una base de datos sobre Microsoft SQL Server que almacena el listado de permisos y llamadas API de cada aplicación. Este procedimiento se realiza en paralelo, aprovechando la forma centralizada que aporta SQL Server, para el cual se utilizaron 10 estaciones de trabajo durante 5 días. De esta manera, la construcción del vector de características que almacena esta información es mucho más rápida. El uso de una base de datos también favoreció el paso de selección de características descrito con anterioridad. Sobre esta se realiza el análisis para escoger las llamadas API y permisos que aportan más información al modelo.

Validación del modelo

Para validar el modelo se realizó un análisis en función del tamaño del conjunto de aprendizaje, lo que se conoce en la literatura como curvas de aprendizaje. Este análisis permite representar el rendimiento de una técnica determinada para distintos tamaños del conjunto de entrenamiento. Los errores del modelo pueden descomponerse en términos de sesgo, varianza y ruido [19]. El sesgo de un estimador es su error promedio para diferentes conjuntos de entrenamiento. La varianza de un estimador indica cuán sensible es a los diferentes conjuntos de entrenamiento. El ruido es una propiedad de los datos.

Una curva de aprendizaje muestra la puntuación de validación y capacitación de un estimador para diferentes números de muestras de entrenamiento [19]. Es una herramienta para descubrir cuánto nos beneficiamos al agregar más datos de capacitación y si el estimador sufre más de un error de varianza o un error de sesgo. Con esta técnica logramos cercioramos que el modelo presentado no sufre de los errores antes mencionados.

El conjunto de muestras se validó utilizando la técnica de validación cruzada K-fold [20]. Con esta técnica los datos de muestra se dividen en K subconjuntos. Uno de los subconjuntos se utiliza como datos de prueba y el resto (K-1) como datos de entrenamiento. El proceso de validación cruzada es repetido durante k iteraciones, con cada uno de los posibles subconjuntos de datos de prueba. Finalmente se realiza la media aritmética de los resultados de cada iteración para obtener un único resultado. En este trabajo utilizamos la validación cruzada de 20 iteraciones.

RESULTADOS

En esta sección se muestra un análisis de los resultados obtenidos con cada uno de los métodos de aprendizaje automático mencionados anteriormente utilizando permisos y llamadas API.

Tabla 5: Precisión del modelo con los distintos algoritmos clasificadores.

Algoritmo	Precisión entrenamiento (%)	Precisión prueba (%)
ID3	98	91
SVM	96	95

Como se puede apreciar en la Tabla 5 los resultados de la investigación en general son muy alentadores. Estos experimentos se realizaron sobre un grupo de 5 335 muestras, 3 153 malignas y 2 202 benignas. Se alcanza mayor precisión con el algoritmo SVM en la etapa de prueba, mientras que en el entrenamiento se comporta mejor ID3, pero en general, los dos clasificadores presentan un buen desempeño.

En trabajos futuros se pretende experimentar con otros algoritmos de aprendizaje automático como Random Forest y Redes Neuronales, que muestran buenos comportamientos en investigaciones similares.

CONCLUSIONES

El malware móvil es una amenaza constante para los usuarios de Android. A medida que estos dispositivos toman protagonismo en nuestra vida cotidiana, se vuelve más importante su seguridad y protección. Por lo tanto, el desarrollo de nuevas técnicas efectivas para la detección de malware debe ser una prioridad.

En esta investigación se desarrolla una técnica basada en aprendizaje automático, una estrategia que ha venido cobrando fuerza en los últimos años con gran éxito. Para ello se utiliza una combinación entre permisos y llamadas API de Java. Además, se realizó un estudio para encontrar los atributos más indicativos de benignidad y malicia. Con este trabajo se alcanzaron resultados considerables, un 95% de precisión en la etapa de prueba en el mejor de los casos. Sin embargo, cada día las técnicas de evasión adoptadas por los atacantes se siguen perfeccionando, lo que significa que se debe seguir trabajando en esta dirección.

Las características antes mencionadas demostraron ser un buen indicador en la clasificación de malware. Por tanto, en investigaciones futuras se pretende efectuar un análisis más exhaustivo sobre estas. Por ejemplo, no solo extraer los permisos que aparecen frecuentemente en aplicaciones malignas, sino buscar también combinaciones inteligentes de estos como indicio de malicia. Además, como se mencionó anteriormente, se quiere experimentar con otros algoritmos clasificadores que igualmente muestran buenos resultados en la literatura estudiada.

REFERENCIAS

[1] GSMA: “The Mobile Economy 2017”, 2017.

[2] R. Simpson: “statcounter”, GlobalStats, 21 marzo 2017. [En línea]. Available: http://gs.statcounter.com/press/android-overtakes-windows-for-first-time. [Último acceso: 19 octubre 2017].

[3] R. Unuchek: “SECURELIST”, Kaspersky Lab, 28 febrero 2017. [En línea]. Available: https://securelist.lat/mobile-malware-evolution-2016/84689/. [Último acceso: 17 Octubre 2017].

[4] D. Y. H. Fereidooni, M. Conti y A. Sperduti: “ANASTASIA: ANdroid mAlware detection using STatic analySIs of Applications”, 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS), pp. 01-05, 2016.

[5] W. Li, J. Ge y G. Dai: “Detecting Malware for Android Platform: An SVM-Based Approach”, de IEEE 2nd International Conference on Cyber Security and Cloud Computing (CSCloud), New York, NY, USA , 2015.

[6] A. Altaher: “Classification of Android Malware Applications using Feature Selection and Classification Algorithms”, VAWKUM Transactions on Computer Sciences, vol. Volume 10, nº 1, pp. 01-05, mayo-junio 2016.

[7] S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder y L. C.: “DroidScribe: Classifying Android Malware Based on Runtime Behavior”, de IEEE Security and Privacy Workshops (SPW), San Jose, CA, USA, 2016.

[8] M. S. Alam y S. T. Vuong: “Random Forest Classification for Detecting Android Malware”, de Green Computing and Communications (GreenCom), 2013 IEEE and Internet of Things (iThings/CPSCom), IEEE International Conference on and IEEE Cyber, Physical and Social Computing, Beijing, China , 2013.

[9] M. Dimjaševič, S. Atzeni, I. Ugrina y Z. Rakamarič: “Android Malware Detection Based on System Calls”, University of Utah, Salt Lake City, USA, 2015.

[10] M. Leeds, M. Keffeler y T. Atkison: “Examining Features for Android Malware Detection”, de Securit and Management, Las Vegas, 2017.

[11] Y. Aafer, W. Du y H. Yin: “DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android”, Dept. of Electrical Engineering & Computer Science , Syracuse University, New York, USA, 2012.

[12] M. Grace, Y. Zhou, Z. Wang y X. Jiang: “Systematic Detection of Capability Leaks in Stock Android Smartphones”, de NDSS, 2012.

[13] Microsoft: “Microsoft Docs”, [En línea]. Available: https://docs.microsoft.com/es-es/sql/analysis-services/data-mining/feature-selection-data-mining. [Último acceso: 11 noviembre 2017].

[14] R. Aler Mur: “Aprendizaje Automático para el Análisis de Datos”, de GRADO EN ESTADÍSTICA Y EMPRESA, Madrid.

[15] “Wikipedia” Fundación Wikimedia, Inc., 24 octubre 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Aprendizaje_autom%C3%A1tico. [Último acceso: 30 octubre 2017].

[16] J. R. Quinlan: “Induction of Decision Trees”, Machine Learning, nº 1, pp. 81-106, 1986.

[17] V. N. Vapnik: The Nature of Statistical Learning Theory, Springer, 2000.

[18] “Accord.NET Framework”, Creative Commons, 25 octubre 2017. [En línea]. Available: http://accord-framework.net/intro.html. [Último acceso: 10 noviembre 2017].

[19] “scikit-learn”, INRIA y otros, [En línea]. Available: http://scikit-learn.org/stable/index.html. [Último acceso: 10 noviembre 2017].

[20] R. Kohavi: “A Study of Cross-Validation and Bootstrap for Accuracy Estimation andn Model Selection”, de International Conference on Artificial Intelligence (IJCAI), Montreal, Quebec, 1995.