Framework for malware analysis in Android

Article · August 2016

DOI: 10.18046/syt.v14i37.2241

2 authors:

Christian Camilo Urcuqui López
ICESI University
31 PUBLICATIONS **46** CITATIONS
SEE PROFILE

Andres Navarro
ICESI University
141 PUBLICATIONS **305** CITATIONS
SEE PROFILE

Discussion Paper / Articulo de Reflexión - Tipo 2

Framework for malware analysis in Android

Christian Camilo Urcuqui López / ccurcuqui@icesi.edu.co

Andrés Navarro Cadavid / anavarro@icesi.edu.co

Grupo de Investigación i2t, Universidad Icesi, Cali, Colombia

ABSTRACT Android is a open source operating system with more than a billion of users, including all kind of devices (cell phones, TV, smart watch, etc). The amount of sensitive data "using" this technologies has increased the cyber criminals interest to develop tools and techniques to acquire that information or to disrupt the device's smooth operation. Despite several solutions are able to guarantee an adequate level of security, day by day the hackers skills grows up (because of their growing experience), what means a permanent challenge for security tools developers. As a response, several members of the research community are using artificial intelligence tools for Android security, particularly machine learning techniques to classify between healthy and malicious apps; from an analytic review of those works, this paper propose a static analysis *framework* and machine learning to do that classification.

KEYWORDS Framework; machine learning; security; Google; malware.

Framework para análisis de software malicioso en Android

RESUMEN Android es un sistema operativo de código abierto con más de mil millones de usuarios activos para todos sus dispositivos (móviles, televisión, relojes inteligentes, entre otros). La cantidad de información sensible que se utiliza en estas tecnologías genera un interés particular de los cibercriminales para el desarrollo de técnicas y herramientas que permitan la adquisición de la información o alteren el buen funcionamiento del dispositivo. Hoy por hoy existen distintas soluciones que permiten tener un nivel razonable de seguridad sobre la información, pero con el transcurrir de los días, la habilidad de los atacantes crece gracias a una mayor experiencia, lo que genera un reto permanente para los desarrolladores de herramientas de seguridad. Debido a la problemática detectada, algunos trabajos han optado por el uso de técnicas de inteligencia artificial en la seguridad en Android, un ejemplo de ello es el uso de algoritmos de aprendizaje de máquina para la clasificación de aplicaciones benignas y malignas, con base en una revisión y análisis de ellas, este artículo se propone un *framework* de análisis estático y aprendizaje de máquina para clasificación de software benigno y malicioso en Android.

PALABRAS CLAVE Framework; aprendizaje de máquina; seguridad; Google; software malicioso.

Quadro de referência para analise de software malicioso para Android

RESUMO Android é um sistema operacional de código aberto com mais de um bilhão de usuários ativos, somando dispositivos móveis, televisão e relógios inteligentes, entre outros. A quantidade de informação sensível utilizada nestas tecnologias incentiva os cibercriminosos ao desenvolvimento de técnicas e ferramentas que permitam a aquisição desta informação ou alterem o bom funcionamento do dispositivo. E embora existam soluções que permitem um razoável nível de segurança da informação, com o passar dos dias a experiência dos atacantes cresce a uma taxa maior do que a dos trabalhos em segurança. Devido aos problemas detectados, alguns optaram por usar técnicas de inteligência artificial na segurança para Android, como o uso de algoritmos de aprendizado de máquina para a classificação de aplicações benignas e malignas. Este artigo propoe um *framework* de análise estática e aprendizado de máquina para a classificação de software benigno e malicioso para Android.

PALAVRAS-CHAVE Framework; aprendizado de máquina; segurança; Google; software malicioso.

Received / Recepción: Mayo 2, 2016 - Accepted / Aceptación: Junio 10, 2016 *doi. 10.18046/syt.v14i37.2241*

I. Introduction

Android is an open-source operating system for mobile devices, televisions, smartwatches, and automobiles, with more than one billion active users (Pichai, 2014). The operating system presents a modification of the Linux kernel and an architecture divided into five components (Elenkov, 2014).

For the protection of the information and devices, Android has several security mechanisms; the most relevant are (Drake et al., 2014): a sandbox environment at the kernel level to prevent access to the file system and other resources; an API of permissions that controls the privileges of the applications in the device; security mechanisms at the applications development level; and a digital distribution platform (Google Play), where the processes are implemented to limit the dissemination of malicious code.

Each application is compiled in an Android Application Package [APK] file, which includes the code of the application (".dex" files), resources, and the AndroidManifest.xml file. This latter is an important element, since it provides most of the information of the security features and configuration of each application (Peiravian & Zhu, 2013). It also includes the information of the API regarding permissions, activities, services, content providers, and the receiving broadcasts.

In the last few years, the development of malicious software (malware) for Android has substantially increased for several reasons. Among these reasons, we have the Android philosophy, its positioning in the mobile market, and the amount of sensitive information produced in these technologies. There are several tools and techniques for the analysis of threats for this operating system. Between the most representative, we have static and dynamic analysis.

Static analysis is a technique that assesses malicious behavior in the source code, the data, or the binary files without the direct execution of the application (Batyuk et al., 2011). Its complexity has increased due to the experience that cybercriminals have gained in the development of applications. However, it has been demonstrated that it is possible to avoid this using obfuscation techniques (Sharif, Lanzi, & Giffin, 2008).

Dynamic analysis is a set of methods that studies the behavior of the malware in execution through gesture simulations. In this technique, the processes in execution, the user interface, the network connections and sockets opening are analyzed (Drake et al., 2014). Alternatively, there already exist some techniques to avoid the processes performed by dynamic analysis, where the malware has the capacity to detect sandbox-like environments and to stop its malicious behavior (Petsas, Voyatzis, Athanasopoulos, Polychronakis, & Ioannidis, 2014).

I. Introducción

Android es un sistema operativo de código abierto para dispositivos móviles, televisores, relojes inteligentes y automóviles; con más de mil millones de usuarios activos (Pichai, 2014). El sistema operativo cuenta con una modificación del kernel de Linux y una arquitectura que se subdivide en cinco componentes (Elenkov, 2014).

Para la protección de la información y de los dispositivos, Android incorpora varios mecanismos de seguridad, entre los más destacados se encuentran (Drake et al., 2014): un entorno sandbox al nivel kernel para prevenir el acceso al file-system y otros recursos; una API de permisos que controla los privilegios de las aplicaciones sobre el dispositivo; mecanismos de seguridad a nivel del desarrollo de aplicaciones; y una plataforma de distribución digital –Google Play–, en donde se implementan los procesos para limitar la difusión de código malicioso.

Cada aplicación esta compilada en un archivo Android Application Package [APK] que incluye el código de la aplicación (archivos .dex), recursos y el archivo AndroidManifest.xml. El archivo AndroidManifest.xml es un elemento importante ya que provee la información de las características y la configuración de seguridad de cada aplicación (Peiravian & Zhu, 2013), he incluye la información de la API de permisos, actividades, servicios, proveedores de contenido y los broadcast receptores.

Durante los últimos años el desarrollo de software malicioso para Android se ha incrementado, por distintas razones, entre ellas: la filosofía de Android, su posicionamiento en el mercado de móviles y la cantidad de información sensible que se produce en estas tecnologías. Existen distintas herramientas y técnicas para el análisis de amenazas para este sistema operativo. Entre las más representativas se encuentran el análisis estático y el análisis dinámico.

El análisis estático es una técnica que evalúa los comportamientos maliciosos en el código fuente, los datos o los archivos binarios, sin ejecutar directamente la aplicación (Batyuk et al., 2011). Su complejidad ha aumentado debido a la experiencia que han adquirido los cibercriminales en el desarrollo de aplicaciones, aunque se ha demostrado que es posible evitarlo a partir de técnicas de ofuscación (Sharif, Lanzi, Giffin, & Lee, 2008).

El análisis dinámico es un conjunto de métodos que estudia el comportamiento del malware en ejecución, mediante simulación de gestos; en esta técnica se analizan los procesos en ejecución, la interfaz de usuario, las conexiones de red y la apertura de sockets, entre otros (Drake et al., 2014). Por otra parte, ya existen técnicas que permiten evadir el proceso realizado por el análisis dinámico, en donde el malware tiene la capacidad de detectar ambientes sandbox y detener su comportamiento malicioso (Petsas et al., 2014).

Se han explorado además distintas técnicas de inteligencia artificial para la seguridad en Android, entre ellas el aprendizaje de máquina. Con este estudio se busca proveer a los sistemas la capacidad de aprender cómo identificar a un software malicioso sin ser programado de forma explícita. Gran cantidad de propuestas hacen uso de los algoritmos de clasificación, como por ejemplo: SVM, Bagging, Neural Network, Decision Tree y Naive Bayes.

El software convencional de seguridad requiere, en su proceso de identificación, de un esfuerzo humano que implica tiempo y recursos, esto podría mejorarse a través del uso de herramientas inteligentes. Uno de los caminos más prometedores es la aplicación de algoritmos de aprendizaje de máquina que permita ser más eficiente la labor de identificación de nuevas amenazas (Chan & Lippmann, 2006; Metz, 2016).

Actualmente se han propuesto distintos trabajos relacionados con la aplicación de clasificadores para el análisis de software malicioso en Android, gran parte de ellos utiliza herramientas para la descarga de aplicaciones y diferentes tecnologías, lo que dificulta la comparación de sus resultados.

Teniendo como base un *framework* de análisis estático, se propone una adaptación que integra bases de datos de aplicaciones utilizadas en artículos publicados con distintas herramientas de libre acceso. Adicionalmente, se evalúa el *framework* a través de la aplicación de distintas configuraciones de seis algoritmos de clasificación: Naive, Bayes, Bagging, KNeighbors, Support Vector Machines [SVM], Stochastic Gradient Descent [SGD] y Decisión Tree. Finalmente, los resultados de este trabajo sirven como base para proponer nuevos retos en el desarrollo de nuevas herramientas y exploración de más características para la predicción de la evaluación de software malicioso para Android. En la sección II se presentan los estudios más representativos que sirvieron como base para el desarrollo de este trabajo; la sección III contiene la información sobre nuestro *framework* de análisis estático y los dataset que se utilizaron para la evaluación de los algoritmos; en la sección IV se incluyen los resultados del experimento; finalmente, en la sección V se presentan las conclusiones y las propuestas de trabajo futuro.

II. Trabajos relacionados

En trabajos anteriores hemos encontrado que gran parte de las propuestas de seguridad se están encaminando hacia la aplicación de técnicas de inteligencia artificial (Fuentes & Gómez, 2014; Londoño, Urcuqui, Amaya, Gómez, & Cadavid, 2015), para abarcar este tema, se realizó una primera exploración de la aplicación de una sola configuración de seis algoritmos de clasificación, obteniendo como resultado que el algoritmo con mejor desempeño en la clasificación fue KNeighbors (Urcuqui & Cadavid, 2016).

Other artificial intelligence techniques have also been explored for safety in Android; one is so-called machine learning. Here, the capacity of the systems to learn how to identify malware without being programmed in an explicit way is presented. A large number of proposals use classification algorithms like SVM, Bagging, Neural Network, Decision Tree, and Naive Bayes.

Conventional security software requires, on its identification process, human effort that implies time and resources. This might be improved through the use of intelligent tools. One of the more promising paths is the application of machine learning algorithms, which entails an increase in the efficiency in the identification of new threats (Chan & Lippmann, 2006; Metz, 2016).

At the present time, some proposals have arisen related to the application of classifiers for the analysis of malicious software in Android; a large number use tools for the download of applications and technologies; hence, the comparison of their results is more complicated.

Having as a foundation a *framework* of static analysis, we propose an adaptation that uses application databases in published articles with different open-source tools. Additionally, we assessed the *framework* through the application of several configurations of six classification algorithms: Naive Bayes, Bagging, KNeighbors, Support Vector Machines [SVM], Stochastic Gradient Descent [SGD], and Decision Tree. Finally, our results serve as a basis for the proposition of new challenges in the development of new tools and for the exploration of more features focused on the prediction of the assessment of malware in Android. In Section II, we present the most relevant studies taken as a foundation for the execution of this study; Section III contains the information of our static analysis *framework* and the datasets used for the evaluation of the algorithms. Section IV displays the results of the experiment and Section V concludes the study.

II. Related work

In previous works, we have found that a considerable number of security proposals feature the application of artificial intelligence techniques (Fuentes & Gómez, 2014; Londoño, Urcuqui, Amaya, Gómez, & Cadavid, 2015). In order to address this topic, we performed a first exploration of the application only with a unique configuration of six classification algorithms, obtaining the best result with the KNeighbors algorithm in the classification task (Urcuqui & Cadavid, 2016).

Not all the machine learning algorithms operate in an efficient way for the same problem; for this reason, it is necessary to perform a study of the most suitable ones for our use. Additionally, these tools require a dataset for learning and tests. In the analysis of malware for Android, it is important to consider a

representative number of applications, both malicious and benign. Through dynamic or static analysis, we can obtain several data of an application, but the challenge is the abstraction of the most representative information, which allows an algorithm to classify it as malicious or benign.

In the following sections, we mention some studies related to the use of machine learning algorithms with static and dynamic analysis, and some others to the development of datasets of Android applications.

A. Machine learning with dynamic analysis

DroidDolphin (Wu & Hung, 2014) is a dynamic analysis *framework* that uses the GUI, big data, and machine learning for the detection of malicious applications in Android. Its analysis process consists of the extraction of information of the calls to the API and 13 activities, whilst the application executed in virtual environments. The SVM machine learning algorithm with the LIBSVM public library (Chang & Lin, 2011) was used, a training dataset of 32.000 benign and malign applications, and a testing dataset of 3.000 healthy applications and 1.000 malicious. The preliminary results showed a precision of 86.1% and an F-score of 0.875.

From their results and conclusions, we derive the following recommendations: consider an equilibrium between healthy applications and the malicious ones; the quality of the prediction depends on the number of applications in the dataset, the high resource consumption when virtual environments are used, and the malicious codes with techniques detecting virtual environments.

Feizollah et al. (2014) conducted an assessment of five machine learning classification algorithms (viz. NB, KNN, DT, MLP, and SVM) with 100 malware samples of the Android Genome Project and 12 samples of healthy applications. That study began with dynamic analysis to extract information of the network traffic in order to detect the malicious activities from the mobile devices. They used the WEKA machine learning software (Hall, 2009), concluding that the best algorithm was KNN with a TPR index of 99.94% against an FPR result of 0.06%.

B. Machine learning with static analysis

Sahs and Khan (2012) propose a machine learning system to detect malware in Android devices with the One-Class SVM classification algorithm and static analysis as a technique to obtain the information from the applications. During the development, they used the Androguard tool to extract the information of the APK and the Scikit-learn *framework* (Documentation..., 2014). From the AndroidManifest.xml, they developed a binary vector that contains the information of each permission used

No todos los algoritmos de aprendizaje de máquina funcionan de manera eficiente para un mismo problema, es por ello que es necesario hacer un estudio de los más adecuados. Adicionalmente, estas herramientas requieren de un conjunto de datos para aprendizaje y pruebas; en el análisis de software malicioso para Android es necesario contar con una cantidad representativa de aplicaciones, tanto maliciosas, como benignas. A través del análisis estático o dinámico se pueden obtener varios datos de una aplicación, pero el reto es abstraer la información más representativa, aquella que permita a un algoritmo clasificarla como maliciosa o benigna.

A continuación se enuncian algunos estudios del uso de algoritmos de aprendizaje de máquina con análisis estático y dinámico, y algunos trabajos sobre desarrollos de datasets de aplicaciones Android.

A. Aprendizaje de máquina con análisis dinámico

DroidDolphin (Wu & Hung, 2014) es un *framework* de análisis dinámico que hace uso de la GUI, del big data y el aprendizaje de máquina para la detección de aplicaciones maliciosas en Android. Su proceso de análisis consiste en extraer la información de las llamadas a la API y de trece actividades, mientras la aplicación se encuentra en ejecución sobre ambientes virtuales. Durante este estudio se utilizó el algoritmo de aprendizaje de máquina SVM con la librería publica LIBSVM (Chang & Lin, 2011), un conjunto de entrenamiento de treinta y dos mil aplicaciones benignas e igual número de aplicaciones maliciosas, un conjunto de test de mil aplicaciones sanas y mil maliciosas. Como resultados preliminares se obtuvo una precisión de 86.1% y un F-score de 0.875.

De sus resultados y conclusiones se derivan las siguientes recomendaciones, tener en cuenta: un equilibrio entre las aplicaciones sanas y las maliciosas; que la calidad de la predicción depende de la cantidad de aplicaciones en el dataset, el alto consumo de recursos al usar entornos virtuales; y los códigos maliciosos con técnicas que detectan entornos virtuales.

El trabajo de Feizollah et al., (2014) consiste en la evaluación de cinco algoritmos de clasificación de aprendizaje de máquina (i.e., NB, KNN, DT, MLP y SVM) con cien muestras de malware del Android Genome Project y doce muestras de aplicaciones sanas. El estudio parte del análisis dinámico para extraer distinta información del tráfico de la red y así detectar las actividades maliciosas desde los dispositivos móviles. En el trabajo se utilizó el software de aprendizaje de máquina WEKA (Hall, 2009) y se concluyó que el mejor algoritmo fue KNN con un índice de TPR de 99.94% contra un resultado de FPR de 0.06%.

B. Aprendizaje de máquina con análisis estático

Sahs y Khan (2012) proponen un sistema de aprendizaje de máquina para detección de malware en dispositivos Android con el algoritmo de clasificación One - Class SVM y el análisis estático como técnica para obtener la

información de las aplicaciones. Durante el desarrollo usaron la herramienta Androguard para extraer la información de las APK y el *framework* Scikit-learn (Documentation..., 2014). A partir del archivo AndroidManifest.xml se desarrolló un vector binario que contiene la información de cada permiso que es utilizado por cada aplicación y un diagrama de flujo de control [CFG] que es una representación abstracta de un programa. Por otra parte, el algoritmo fue implementado con un conjunto de test de 2081 aplicaciones benignas y 91 maliciosas, con la información de cinco kernel –sobre vectores binarios, cadenas, diagramas, conjuntos y permisos no comunes– y otro para cada aplicación. Como resultado consiguieron una baja tasa en falsos negativos pero un alta tasa de falsos positivos.

Existen estudios que exploran otros algoritmos de aprendizaje de máquina, como las redes neuronales [NN], como el realizado por Ghorbanzadeh, Chen, Ma, Clancy, & McGwier (2013), quienes proponen evaluar las vulnerabilidades de seguridad que se presentan en una estructura de permisos de una aplicación Android. Durante su estudio utilizaron un dataset de 1.700 aplicaciones benignas y maliciosas junto con la herramienta Apktool que permite descompilar un archivo APK y obtener archivos .xml como el AndroidManifest.xml. Por otra parte, la fase de entrenamiento, validación y testeo tomó un 70%, 10% y 20%, respectivamente de los datos del dataset; La NN cuenta con una estructura de dos capas con diez neuronas y una capa de salida de 34 neuronas; al algoritmo se le suministró la información de un vector binario que representa los permisos solicitados por cada aplicación. Finalmente, a través de diez experimentos, se consiguió una precisión de 65%.

Yerima, Sezer, McWilliams, y Muttik (2013) proponen un modelo de aprendizaje de máquina que integra un clasificador Bayesiano; su diferencial se encuentra en la información que será suministrada al sistema de aprendizaje a partir de detectores: de llamadas a la API, de comandos, de permisos, de código encriptado y presencia de aplicaciones secundarias o archivos .jar.

Cada detector genera variables binarias sobre las APK, analizadas. A partir de estos datos se construye una clase que indica si el aplicativo es sospechoso o benigno. Este resultado será la información suministrada al clasificador Bayesiano. El estudio contó con un total de dos mil aplicaciones, tanto de tiendas oficiales, como de terceros, mil software maliciosos y mil aplicaciones benignas. Adicionalmente, el trabajo empleó un conjunto de 1.600 aplicaciones, 800 para benignas y 800 para códigos maliciosos. Para la fase de entrenamiento se usaron 400 (200 para benignas y 200 para códigos maliciosos). Los resultados más significativos obtenidos para un total de veinte características y 1600 aplicaciones, fueron: TPR 90.6%, FNR 0.094%, precisión de 93.5% y AUC de 97.22%.

for every application and a Control Flow Graph [CFG], which corresponds to an abstract representation of a program. Conversely, the algorithm was implemented with a test set of 2.081 clean apps and 91 malicious, with the information of 5 kernels – about binary vectors, strings, diagrams, sets, and uncommon permissions – and another one for each application. As a result, they were able to obtain a low rate of false negatives but a high rate of false positives.

There are also studies that explore other machine learning algorithms such as Neural Networks [NN], e.g. Ghorbanzadeh, Chen, Ma, Clancy, and McGwier (2013), where they propose the assessment of the security vulnerabilities presented in a permission structure of an Android application. These workers used a dataset of 1.700 benign and malicious applications together with the Apktool tool, which allows the de-compiling of an APK file and obtain .xml files like the AndroidManifest.xml. The training, validation, and testing phases took 70%, 10%, and 20% of the dataset data, respectively. The NN has a two-layered structure with 10 neurons and an exit layer with 34 neurons. The algorithm received the information from a binary vector that represents the requested permissions for each application. Finally, through 10 experiments, they achieved a precision of 65%.

Yerima, Sezer, McWilliams, and Muttik (2013) propose a machine learning model that integrates a Bayesian classifier; its difference relies on the information to be provided to the learning system through detectors of API calls, permission commands, encrypted code, and presence of secondary applications or .jar files.

Each detector generates binary variables over the analyzed APK. Through these data, a class that indicates whether the application is suspect or benign is generated. This result will be the information provided to the Bayesian classifier. The study used 2.000 applications, 1.000 malicious and 1.000 healthy, both from official stores and third-party ones. Furthermore, they employed a set of 1.600 applications, 800 healthy and 800 malicious. For the training phase, 400 applications were used, 200 healthy, 200 malicious. The most significant results obtained for a total of 20 features and 1.600 applications were a TPR of 90.6%, FNR of 0.094%, precision of 93.5%, and AUC of 97.22%.

C. Dataset

Earlier, a malicious dataset software known as Android Genome Project – MalGenome – was published (Zhou & Jiang, 2012). The development has a sample of 1.260 malicious applications with 49 different families. The features description of its

dataset was also published and, as a part of the results, 1.083 malicious codes (86%) were re-packaged versions of legitimate applications, 36.7% had the capacity to raise privileges, and 45.3% had as their end the subscription of premium message systems. They assessed four security software that in the best case could detect 79.6% of the malicious applications; in the worst case, 20.2%.

Krutz et al. (2015) analyzed a free-access dataset containing more than 1.000 Android applications, more than 4.000 of their versions, and a total of 430,000 commits of online repositories. In addition of their results, it is possible to obtain information regarding the applications under and over privileges per commit category in the repositories and versions.

III. Methodology

We now describe the proposed *framework* for the analysis of malicious software in Android. The *framework* is based on the results of previous studies (Peiravian & Zhu, 2013; Yerima et al., 2013) and it uses two databases resulting from another two studies (Zhou & Jiang, 2012; Krutz et al., 2015). We also employ static analysis for the extraction of the features that identify each application, and take advantage of some machine learning free tools for the analysis of the applications.

A. Static analysis framework

The proposed *framework* (Figure 1) is composed of four phases: gathering, extraction, features generation, and training and testing.

- Gathering: we collected a total of 558 APK. The collection is composed of 279 applications with low privileges of the free access dataset mentioned in Section II (Krutz et al., 2015) and a random selection of 279 malwares of the MalGenome.
- Extraction: we used the ApkTool 2.0.3 tool to obtain the AndroidManifest.xml file of the gathered APK.
- Features generator: we developed some tools in Python that create the binary vectors that correspond to the granted permissions for each application; also, we added a layer regarding the classification of the application (either benign or malign). With these results, we created a new dataset containing the granted permissions for each application in binary.
- Training and testing: we created two partitions, one with 71% and the other with 29% of the data for the training and testing phases, respectively. Finally, we trained and verified each machine learning algorithm.

C. Dataset

En 2012 un grupo de investigadores publicó un dataset de software maliciosos conocido como Android Genome Project –MalGenome– (Zhou & Jiang, 2012). El desarrollo cuenta con una muestra de 1.260 aplicaciones maliciosas con 49 familias distintas. El grupo publicó la descripción de las características de su dataset; como parte de sus resultados se encuentran: 1.083 códigos maliciosos (86%) eran versiones re-empaquetadas de aplicaciones legítimas; 36.7% tenía la capacidad de elevar sus privilegios; y 45.3% tenía como finalidad la suscripción de servicios premium de mensajería. Se evaluaron cuatro software de seguridad. En el mejor de los casos se detectó 79.6% de aplicaciones maliciosas, en el peor, 20.2%

Una investigación de Krutz et al., (2015) provee a las personas interesadas el análisis y la publicación de un dataset de libre acceso que contiene una colección de más de mil aplicaciones Android, más de cuatro mil de sus versiones y un total de 430.000 commits de repositorios en línea. Adicionalmente, de sus resultados se puede obtener información respecto de las aplicaciones con sobre y bajo privilegios por categoría de los commits en los repositorios, y versiones, entre otros.

III. Metodología

A continuación se describe la propuesta del *framework* para análisis de software malicioso en Android. El *framework* se basa de los resultados de estudios previos (Peiravian & Zhu, 2013; Yerima et al., 2013), utiliza dos bases de datos que son resultados de trabajos publicados (Krutz et al., 2015; Zhou & Jiang, 2012), emplea el análisis estático para la extracción de las características que identifican a cada aplicación, y aprovecha herramientas libres de aprendizaje de máquina para el análisis de las aplicaciones.

A. Framework de análisis estático

El *framework* propuesto (Figura I) se compone de cuatro fases: recolección, extracción, generación de características y entrenamiento y pruebas.

- Recolección: se recolectó un total de 558 APK. La colección se encuentra compuesta por 279 aplicaciones de bajos privilegios del dataset de libre acceso mencionado en la sección II (Krutz et al., 2015) y una selección aleatoria de 279 malware del MalGenome.
- Extracción: se utilizó la herramienta ApkTool 2.0.3 para obtener los archivos AndroidManifest.xml de los APK recolectados.
- Generador de características: se desarrollaron herramientas en Python que tienen como finalidad crear los vectores binarios que corresponden a los permisos accedidos por cada aplicación, y se adicionó una etiqueta que concierne a la clasificación del aplicativo (benigno o maligno). Con los resultados

anteriores se creó un nuevo dataset que contiene los permisos accedidos por cada aplicación en binario.

- Entrenamiento y pruebas: se crearon dos particiones, una con el 71%, otra con el 29% de los datos para la fase de entrenamiento y de pruebas, respectivamente. Finalmente, se entrenó y verificó cada algoritmo de aprendizaje de máquina.

B. Generador de características

A través del desarrollo de un analizador de permisos se evaluaron los 558 AndroidManifest.xml contra una lista de 330 permisos. El analizador se encarga de crear una variable binaria (1) por cada permiso de la lista, el valor que le corresponde depende de si el permiso evaluado en la lista se encuentra o no en el archivo AndroidManifest.xml analizado.

R_i=1 El analizador detectó un permiso accedido
R_i=0 En otro caso (1)

Luego de terminar de analizar los permisos y antes de pasar a evaluar otro archivo AndroidManifest.xml, se crea una etiqueta que corresponde a la clasificación del aplicativo:

C_i=1 Si el aplicativo es malicioso
C_i=0 En otro caso (2)

A cada aplicación se le asigna a un vector definido por V = (R1, R2,…, R330, C) que contiene la información de los permisos y la etiqueta de clasificación.

C. Medidas de evaluación

Para evaluar el desempeño de los clasificadores en el contexto de nuestro problema, se utilizaron las siguientes medidas de evaluación:

Accuracy = # datos correctamente clasificados
Accuracy = # datos clasificados (3)

El Area Under ROC [AUC] es una estimación del área bajo la curva de ROC, que indica la capacidad de predicción del clasificador.

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

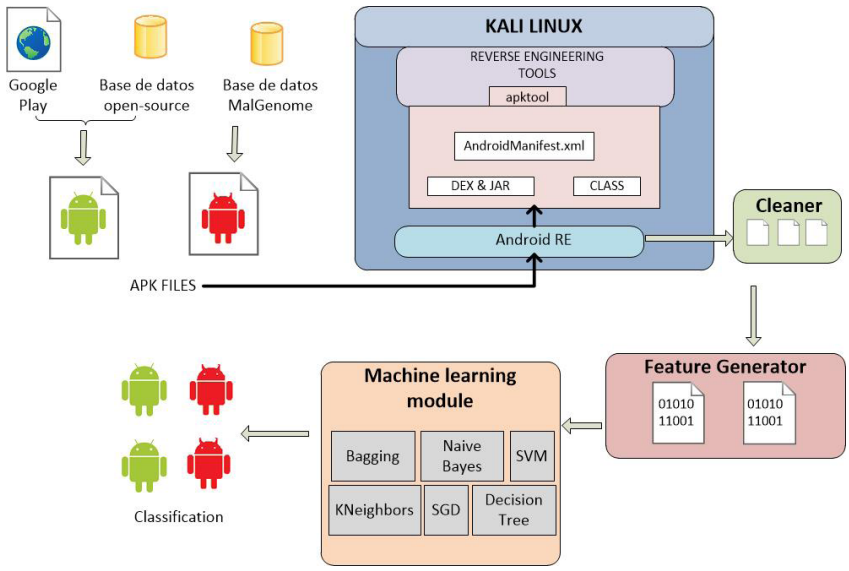

Figure 1. Framework for the analysis of malware in Android / Framework para analisis de software malicioso en Android

B. Features generator

Through the development of a permission analyzer, we assessed the 558 AndroidManifest.xml files with a list of 330 permissions. The analyzer is handled to create a binary variable (1); for each permission of the list, the value that corresponds depends on whether or not the assessed file in the list is in the analyzed AndroidManifest.xml file.

R_i=1 The analyzer detected a granted permission
R_i=0 Any other case (1)

After finishing the analysis of the permissions and before continuing to evaluate other AndroidManifest.xml files, a label corresponding to the classification of the application is created:

C_i=1 If the application is malware
C_i=0 Any other case (2)

For each application, we assign a vector defined by V = (R1, R2,…, R330, C), that contains the information of the permissions and the classification label.

C. Assessment measurements

In order to assess the performance of the classifiers in the context of our problem, we used the following assessment measurements:

Accuracy = number of correctly classified data
Accuracy = number of classifed data (3)

$$Precision = \frac{TP}{TP + FP} \quad (4)$$

$$Recall = \frac{TP}{TP + FN} \quad (5)$$

The area Under ROC [AUC] is an estimation of the area under the ROC curve, which indicates the prediction capacity of the classifier.

True Positive (TP) / *Verdadero Positivo*	False Positive (FP) / *Falso Positivo*
False Negative (FN) / *Falso Negativo*	True Negativa (TN) / *Verdadero Negativo*

Table 1. Confusión Matrix / Matriz de confusión

IV. Experiment

A. Employed technologies and results

We used Python as a development language, and the machine learning tools of Scikit-Learn 0.17. The selected classification algorithms for the experiment were: Naive Bayes, Bagging, KNeighbors, Support Vector Machines (SVM), Stochastic Gradient Descent (SGD), and Decision Tree. During the application of the methodology described in Section III, we assessed some algorithms with different configurations. The results of each test are described in the following sections.

Naive Bayes

As part of the experiment, we used the Naive Bayes algorithm for both Gaussian and Bernoulli distributions. Ultimately, with a performance in the classification of 90%, the algorithm with the highest results was for Bernoulli distributions (**Table 2**).

Bagging

We performed four bagging tests with several configurations. In these, we used the Decision Tree and KNeighbors classifiers. Two configurations presented samples of 30% of the data and 20% of the features; the other hand, samples of 90% data and 90% features. In conclusion, the bagging configuration with the best results in the performance of the classification was Decision Tree for samples of 90%, both from the data and the features (**Table 3**).

IV. Experimento

A. Tecnologías empleadas y sus resultados

Durante el trabajo se utilizó, como lenguaje de desarrollo, Python, y las herramientas de machine learning de Scikit-Learn 0.17. Los algoritmos de clasificación seleccionados para el experimento fueron los siguientes: Naive Bayes, Bagging, KNeighbors, Support Vector Machines (SVM), Stochastic Gradient Descent (SGD) y Decision Tree. Durante la aplicación de la metodología descrita en la sección III se evaluaron algunos algoritmos con distintas configuraciones. Los resultados de cada prueba se describen a continuación.

Naive Bayes

Como parte del experimento se utilizó el algoritmo de Naive Bayes tanto para distribuciones gaussianas, como para las de Bernoulli. Finalmente, con un desempeño en la clasificación del 90% el mejor algoritmo fue para distribuciones de Bernoulli (ver **Tabla 2**).

Bagging

Se realizaron cuatro pruebas de Bagging con distintas configuraciones, en estas se utilizaron los clasificadores de Decision Tree y KNeighbors. Dos configuraciones tomaron muestras del 30% de los datos y 20% de las características; las otras tienen muestras del 90% los datos y el 90% de las características. Finalmente, la configuración de Bagging con los mejores resultados en el desempeño de la clasificación fue Decision Tree para muestras del 90%, tanto de los datos, como de las características (ver **Tabla 3**).

KNeighbors

Para probar el algoritmo de clasificación KNeighbors se emplearon los siguientes valores 2, 3, 4 y 6, para k. Como parte de los resultados, el desempeño de la clasificación aumentó hasta $k = 4$ y empezó a disminuir con $k > 6$ (**Tabla 4**).

	Performance metrics / *métricas de desempeño*						
Algorithm	Precision		Recall		f1 - score		Accuracy
	0	1	0	1	0	1	
Gaussian	0,90	0,76	0,79	0,88	0,84	0,82	0,84
Bernoulli	0,93	0,88	0,88	0,92	0,90	0,90	0,90

Table 2. Performance of Naive Bayes classifieres / Desempeño de los clasificadores de Naive Bayes

			Performance metrics / *métricas de desempeño*						
Algorithm	Examples (%)	Features (%)	Precision		Recall		f1 - score		Accuracy
			0	1	0	1	0	1	
Kneighbors	30	20	0.94	0.88	0.88	0.93	0.91	0.90	0.91
	90	90	0.94	0.88	0.88	0.93	0.91	0.90	0.91
Decision Tree	30	20	0,93	0,80	0,82	0,91	0,87	0,85	0,87
	90	90	0,90	0,95	0,95	0,90	0,92	0,93	0,93

Table 3. Performance of Bagging classifieres / Desempeño de los clasificadores de Bagging

	Performance metrics / *métricas de desempeño*						
Algorithm	Precision		Recall		f1 - score		Accuracy
	0	1	0	1	0	1	
K = 2	0,93	0,94	0,94	0,93	0,93	0,93	0,93
K = 3	0,90	0,95	0,95	0,90	0,92	0,93	0,93
K = 4	0,94	0,95	0,95	0,94	0,94	0,94	0,94
K = 6	0,95	0,89	0,89	0,95	0,92	0,92	0,92

Tabla 4. Performance of KNeighbors classifieres / Desempeño de los clasificadores de KNeighbors

	Performance metrics / *métricas de desempeño*						
Algorithm	Precision		Recall		f1 - score		Accuracy
	0	1	0	1	0	1	
SGD	0,91	0,93	0,92	0,91	0,92	0,92	0,92
DT	0.93	0,95	0,95	0,91	0,94	0,94	0,94

Tabla 6. Performance of SGD and DT classifieres / Desempeño de los clasificadores de SGD y DT

Support Vector Machines [SVM]

Para los algoritmos de SVM se realizaron dos pruebas, una con una función lineal, otra con una función de base radial. Los resultados permiten concluir que el algoritmo con el mejor desempeño fue el que utilizo la función lineal (ver **Tabla 5**).

Stochastic Gradient Descent [SGD] y Decision Tree [DT]

Se realizó una prueba para cada clasificador con los siguientes parámetros: el algoritmo de SGD fue entrenado con una función de perdida hinge loss con una penalidad de 12; para Decision Tree se trabajó con la configuración por defecto de sckit-learn (ver **Tabla 6**).

B. Comparación de los resultados de las pruebas

Revisando los mejores resultados de cada algoritmo (**Tabla 7**), tanto el algoritmo de Naive Bayes y SGD presentan los menores resultados en el desempeño de la clasificación, con un 90% y 92% respectivamente. Posteriormente, Bagging continúa en el listado con una buena detección de software malicioso del 95% y con un desempeño de clasificación del 93%. Finalmente, los algoritmos de KNeighbors, SVM y Decision Tree cuentan con un desempeño de clasificación del 94% y se puede comprobar que presentan un comportamiento similar en los resultados de los VP contra los FN en la curva ROC (**Figura 2**).

C. Diferencial

Gran parte de los estudios revisados en la sección II, hacen uso de un crawler (Narudin, Feizollah, Anuar, & Gani, 2014) para la recolección de aplicaciones tanto sanas como maliciosas, y otros emplean el proyecto MalGenome. Pero, ninguno de los anteriores presenta entre sus resultados el dataset para que otras personas puedan hacer de su uso. Por lo anterior, es una motivación para este traba-

	Performance metrics / *métricas de desempeño*						
Algorithm	Precision		Recall		f1 - score		Accuracy
	0	1	0	1	0	1	
Lineal	0,93	0,95	0,95	0,93	0,94	0,94	0,94
RBF	0.93	0,88	0,88	0,92	0,90	0,90	0,90

Tabla 5. Performance of SVM classifieres / Desempeño de los clasificadores de SVM

	Performance metrics / *métricas de desempeño*						
Algorithm	Precision		Recall		f1 - score		Accuracy
	0	1	0	1	0	1	
Naive Bayes	0,93	0,88	0,88	0,92	0,90	0,90	0,90
Bagging	0,90	0,95	0,95	0,90	0,92	0,93	0,93
KNeighbors	0,94	0,95	0,95	0,94	0,94	0,94	0,94
SVM	0,93	0,95	0,95	0,93	0,94	0,94	0,94
SGD	0,91	0,93	0,92	0,91	0,92	0,92	0,92
Decision Tree	0,93	0,95	0,95	0,91	0,94	0,94	0,94

Tabla 7. Individual performance of classifiers / Desempeño individual de los clasificadores

KNeighbors

In order to test the KNeighbors algorithm, we employed the values 2, 3, 4, and 6 for *k*. As part of the results, the performance of the classification increased until k=4 and started to reduce for $k \geq 6$ (**Table 4**).

Support Vector Machines [SVM]

For the SVM algorithms, we carried out two tests, one with a lineal function and the other with a radial basis function. The results allow us to conclude that the algorithm with the best performance was that using the linear function (**Table 5**).

Stochastic Gradient Descent [SGD] and Decision Tree [DT]

We performed a test for each classifier with the following parameters: the SGD algorithm was trained with a hinge loss function with a penalty of 12; for Decision Tree, we worked with the sckit-learn default configuration (**Table 6**).

B. Comparison of the test results

Checking the best results for each algorithm (**Table 7**), both the Naive Bayes and the SGD results had the lowest results in the classification performance: 90% and 92%, respectively. In addition, Bagging continues in the list with a good detection of malicious software of 95% and with a classification performance of 93%. Finally, the KNeighbors, SVM, and Decision Tree have a clas-

sification performance of 94%. It is possible, through comparison, to see that these latter two present a similar behavior in the VP results versus the FN in the ROC curve (**Figure 2**).

C. Differential

A large number of revised studies in Section II use a crawler (Narudin, Feizollah, Anuar, & Gani, 2014) for the application gathering, either healthy or malicious; whilst other projects use the MalGenome project. However, none of these works presents in their results the dataset for use by other people. For this reason, the presentation of a *framework* that allows the analysis of malicious software in Android from applications that might be accessed and used in other studies is one motivation for this work. The obtained results will help as a comparison measure for future studies.

V. Conclusions and future work

We have proposed the assessment of a *framework* for the analysis of malicious software in Android. Our proposal uses six machine learning algorithms. From the obtained results in this experiment, the assessment of our dataset uses as better classification algorithms KNeighbors, SVM, and Decision Tree. Our results were obtained from applications used in other studies; nevertheless, as our contribution, we offer a *framework* that uses static analysis for the assessment of malicious applications for Android by using machine learning techniques. Consequently, some future work proposals could be to:

- evaluate the trained classifiers against another set of healthy applications with over-privileges;
- assess the models against new malicious codes;
- study other artificial intelligence techniques; and
- implement in our *framework* a method that allows the algorithm to learn bad classifications and new Android features for the training and testing of the classifiers.

Acknowledgments

We thank the *Universidad Icesi* and the professors of the Master in Informatics and Telecommunications for the development of this work; a special mention to José Ignacio Claros for his comments. We also thank everyone else for supporting this project. ST

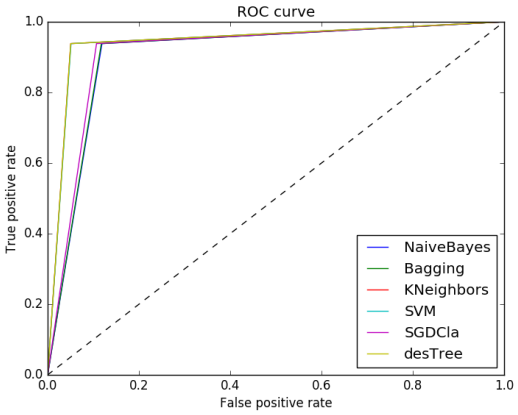

Figure 2. ROC Curv / Curva ROC

jo presentar un *framework* que permita el análisis de software malicioso en Android partir de aplicaciones que pudieran ser accedidas y que fueran utilizadas en otros artículos, debido a que los resultados obtenidos servirán como medidas de comparación para futuros estudios.

V. Conclusiones y trabajo futuro

En este artículo se propuso la evaluación de un *framework* para análisis de software malicioso en Android que utiliza seis algoritmos de aprendizaje de máquina. De los resultados obtenidos en el experimento, la evaluación de nuestro dataset presenta como mejores algoritmos de clasificación KNeighbors, SVM y Decision Tree. Los resultados presentados se obtienen a partir de aplicaciones utilizadas en otros artículos, sin embargo, como aporte, ofrecemos un marco de trabajo que utiliza el análisis estático para la evaluación de aplicaciones maliciosas para Android haciendo uso de técnicas de aprendizaje de máquina. Surgen, como propuestas de trabajo a futuro, las siguientes:

- evaluar los clasificadores entrenados contra otro conjunto de aplicaciones sanas con sobre privilegios;
- evaluar los modelos contra nuevos códigos maliciosos;
- estudiar otras técnicas de inteligencia artificial; e
- implementar en el *framework* un método que permita al algoritmo aprender de las malas clasificaciones y de nuevas características de Android para el entrenamiento y testeo de los clasificadores.

Agradecimientos

Gracias a la *Universidad Icesi* y a los profesores de la Maestría en Informática y Telecomunicaciones por sus conocimientos para el desarrollo de este trabajo, a José Ignacio Claros por sus comentarios, y para todos aquellos que han apoyado en este proyecto. ST

References / Referencias

Batyuk, L., Herpich, M., Camtepe, S. A., Raddatz, K., Schmidt, A., & Albayrak, S. (2011). Using static analysis for automatic assessment and mitigation of unwanted and malicious activities within Android applications. *Malicious and Unwanted Software (MALWARE), 2011 6th International Conference on*. Piscataway, NJ: IEEE.

Chan, P. K. & Lippmann, R. P. (2006). Machine learning for computer security. *The Journal of Machine Learning Research, 7, 2669-2672.*

Chang, C. C. & Lin, C. J. (2011). LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST), 2(3), 27.*

Documentation of scikit-learn 0.16.1. (2014). [blog: Scikit-learn]. Retrieved from: http://scikit-learn.org/0.16/documentation.html

Drake, J. J., Lanier, Z., Mulliner, C., Fora, P. O., Ridley, S. A., & Wicherski, G. (2014, March 26). *Android Hacker's Handbook.* John Wiley & Sons.

Elenkov, N. (2014). Android security internals: *An in-depth guide to Android's security architecture.* San Francisco, CA: No Starch Press.

Feizollah, A., Anuar, N. B., Salleh, R., Amalina, F., Ma'arof, R. U. R., & Shamshirband, S. (2014). A study of machine learning classifiers for anomaly-based mobile botnet detection. *Malaysian Journal of Computer Science*, 26(4), 251-265.

Fuentes, M. & Gómez, J. (2014). Valoración de la plataforma ASEF como base para detección de malware en aplicaciones Android. *Ingenium*, 8(21), 11-23.

Ghorbanzadeh, M., Chen, Y., Ma, Z., Clancy, T. C., & McGwier, R. (2013, January). A neural network approach to category validation of android applications. *In Computing, Networking and Communications (ICNC), 2013 International Conference on* (pp. 740-744). Piscataway, NJ: IEEE.

Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., & Witten, I. H. (2009). The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1), 10-18.

Krutz, D. E., Mirakhorli, M., Malachowsky, S. A., Ruiz, A., Peterson, J., Filipski, A., & Smith, J. (2015, May). A dataset of open-source Android applications. *In Mining Software Repositories (MSR), 2015 IEEE/ACM 12th Working Conference on* (pp. 522-525). Los Alamitos, CA: IEEE Computer Society.

Londoño, S., Urcuqui, C., Amaya, M., Gómez, J., & Cadavid, A. (2015). SafeCandy: System for security, analysis and validation in Android. *Sistemas & Telemática*, 13(35), 89-102.

Metz, C. (2016, junio 2). Google's training its ai to be Android's security guard. *Wired*. Retrieved from: https://www.wired.com/2016/06/googles-android-security-team-turns-machine-learning?utm_content=buffere407d

Narudin, F. A., Feizollah, A., Anuar, N. B., & Gani, A. (2014). Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing,* 20(1), 343-357. 2014.

Peiravian, N., & Zhu, X. Machine learning for android malware detection using permission and api calls. In *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on* (pp. 300-305). Los Alamitos, CA: IEEE Computer Society.

Petsas, T., Voyatzis, G., Athanasopoulos, E., Polychronakis, M., & Ioannidis, S. (2014, April). Rage against the virtual machine: hindering dynamic analysis of android malware. *In Proceedings of the Seventh European Workshop on System Security* (p. 5). New York, NY: ACM.

Pichai, S. (2014). Google I/O 2014 - Keynote [video. 6:43m]. Retrieved from https://www.google.com/events/io

Sahs, J., & Khan, L. (2012). A machine learning approach to android malware detection. *In Intelligence and Security Informatics Conference (EISIC), 2012 European* (pp. 141-147). Los Alamitos, CA: IEEE Computer Society.

Sharif, M. I., Lanzi, A., Giffin, J. T., & Lee, W. (2008). Impeding malware analysis using conditional code obfuscation. In NDSS Symposium 2008 (paper 19). Reston, VA: Internet Society. Retrieved from: http://www.isoc.org/isoc/conferences/ndss/08/papers/19_impeding_malware_analysis.pdf

Urcuqui, C. & Cadavid, A. Machine learning classifiers for Android malware analysis. *Proceedings of the IEEE Colombian Conference on Communications and Computing 2016* [in press].

Wu, W. C. & Hung, S. H. (2014). DroidDolphin: a dynamic Android malware detection framework using big data and machine learning. In: *Proceedings of the 2014 Conference on Research in Adaptive and Convergent Systems* (pp. 247-252). New York, NY: ACM. October 2014.

Yerima, S. Y., Sezer, S., McWilliams, G., & Muttik, I. (2013). A new android malware detection approach using bayesian classification. *In Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on* (pp. 121-128). Los Alamitos, CA: IEEE Computer Society.

Zhou, Y., & Jiang, X. (2012, May). Dissecting android malware: Characterization and evolution. In Proceedings 2012 IEEE Symposium on Security and Privacy: S&P 2012 (pp. 95-109). Los Alamitos, CA: IEEE Computer Society.

CURRICULUM VITAE

Christian Camilo Urcuqui Systems Engineer and Master in Informatics and Telecommunications from Universidad Icesi (Cali-Colombia). Member of Informatics and Telecommunications research group [i2t]. His areas of interest include: artificial intelligence, machine learning and security applied to informatics. Ingeniero de Sistemas y Máster en Informática y Telecomunicaciones de la Universidad Icesi (Cali-Colombia). Miembro del grupo de investigación en Informática y Telecomunicaciones [i2t] de la Universidad Icesi. Sus áreas de interés incluyen: inteligencia artificial, aprendizaje de máquina, y seguridad informática.

Andrés Navarro Cadavid, Ph.D. Electronic Engineer and Magister in Technology Management of the Universidad Pontificia Bolivariana (Medellín, Colombia) and Doctor of Engineering in Telecommunications of the Universidad Politécnica de Valencia (Spain). Full time professor and leader of the Informatics and Telecommunications research group (i2T) attached to the Information and Communications Department at the Universidad Icesi (Cali-Colombia). Counselor at the National Program of Electronics, Telecommunications and Informatics [ETI]. His areas of interest include Spectrum Management, Cognitive Radio, and Telematics solutions for health / Ingeniero Electrónico y Máster en Gestión Tecnológica de la Universidad Pontificia Bolivariana y Doctor Ingeniero en Telecomunicaciones de la Universidad Politécnica de Valencia (España). Profesor de planta y líder del grupo de investigación en Informática y Telecomunicaciones (i2t), adscrito al Departamento de Tecnologías de la Información y las Comunicaciones de la Universidad Icesi; es Consejero del Programa Nacional de Electrónica, Telecomunicaciones e Informática [ETI]. Sus áreas de interés incluyen: gestión del espectro, radio cognitiva y aplicaciones de la ingeniería telemática en salud.