On the use of Machine Learning techniques to detect Malware in Android Operative System: *A survey*

María Fernanda Mora Alba
Department of Computer Science
Instituto Tecnológico Autónomo de México
México, Distrito Federal
Email: maria.mora@itam.mx

***Abstract*—In this survey we explore the most novel and relevant approaches for malware detection in Android Operative System using Machine Learning techniques. Specifically, we reviewed the most cited published works from 2012 to 2016. The target audience is machine learning researchers, therefore we first provide a substantial general overview on mobile device security, as well as on Android Operative System and security. The survey is organized by the type of analysis that is performed: static, dynamic, permission-based or hybrid. Along with a description of each work, we analyze Machine Learning relevant elements: the features, the dataset, the models and the performance results. We additionally provide a comparative table with all the surveyed works with additional elements (such as the scope, type of malware, monitoring environment and type of learning), which can be consulted *here*. We conclude the survey with some opportunity areas regarding a Machine Learning approach for Android malware detection, together with interesting venues of work regarding a general approach to detect Android malware.**

***Index Terms*—Malware detection, Android security, Android Operative System, Security in mobile devices, Machine Learning, Classification, Clustering, Dimensionality Reduction, Static Analysis, Dynamic Analysis, Permissions-based analysis.**

I. Introduction

Mobile Devices are the fastest growing consumer technology, with 1.9 billion worldwide reported users in 2014, surpassing for the first time PC users, according to a report from Morgan Stanley Research.

In addition to the impressive number of users, people is spending more time on the mobile devices: in 2011, for the first time ever, people spent on average more time using mobile applications than browsing the web: 81 minutes vs 74 minutes [Rashidi and Fung2015].

In contrast to conventional personal computers, applications are essential for the *mobile device experience*, as they provide entertainment, productivity enhancement, healthcare, online dating, home security and business management. Hence, “mobile device applications are becoming increasingly sophisticated, robust, life-engaging and also privacy-intrusive” [Rashidi and Fung2015]. So, while once limited to simple voice communication, the mobile device now allows multiple and very different functionalities. This implies that very sensitive information is stored: personal, financial or laboral such as contacts, messages, photos and passwords. Usually, users are careless, they think of the mobile device as the simple communication device it was years ago. This landscape provides a very attractive field for security attacks [Rashidi and Fung2015].

For example, malicious software, for example Malware, can not only steal private information such as contacts list, text messages or location, but it also can cause financial loss by making camouflaged premium-rate phone calls and text messages, or even stealing credit card information [Rashidi and Fung2015]. While malware was initially created to attain notoriety or simply for fun, nowadays it is empowered by financial incentives [Demme et al.2013].

Android is currently the most popular smartphone operating system: 1.1 billion devices running on Google's Android Operating System in 2014, marking its >80% of mobile market share. Additional to the number of users, the number of Android appplications reached 1.6 million in early 2015, beating its major competitor, Apple Apps Store [Rashidi and Fung2015].

[Rashidi and Fung2015] says that “it is prohibitive for app marketplaces such as Google App Store, to thoroughly verify if an app is legitimate or maliciuos. Mobile users are left to decide for themselves whether an app is safe to use”. The problem is that users are neither informed nor attentive: in a survey of 308 Android users they showed low attention and comprehension rates, as only 17% of participants paid attention to permissions during installation, and even worse, 3% of respondents could correctly answer all three permission comprehension questions [Felt et al.2012]. These results also showed that Android's permission system, that was supposed to notify users about the risks of installing applications, actually was not helping the user make adequate security decisions.

The fact that, unlike iPhone Operative System users, Android Operative System users do not have to root or jailbreak their devices to install applications coming from from unknown sources, puts more pressure on the importance of the security of Android.

On the one hand, even though important advances have been made on malware detection in traditional personal computers during the last decades, adopting and adapting those techniques to smart devices is a challenging problem. For example, power consumption is one major constraint that makes unaffordable to run traditional detection systems on the device, while externalized (i.e. cloud-based) techniques rise many privacy concerns [Suarez-Tangil et al.2014b].

On the other hand, relying on currently developed approaches is not enough because intelligent malware keeps modifying and adapting very rapidly, so detecting it becomes more challenging and difficult [Narudin et al.2016]. So, traditional approaches such as the antivirus is not very effective, since it requires continuous signature database updating [Sohr et al.2011]. Additionally, antivirus solutions tend to require important resource consumption and software complexity [Narudin et al.2016], hence unviable for mobile devices.

Therefore we need automatic, reliable and scalable mechanisms. Historically, machine learning classifiers have played a part in the development of intelligent systems for many years. The classifiers are provided with a labelled dataset, trained on it and then they produce a model, allowing the labelling of new data. [Narudin et al.2016] claims that "adopting machine learning classifiers has proven to enhance detection accuracy".

In response to these needs, in this survey we explore the most novel and relevant approaches for malware detection in Android Operative System using Machine Learning techniques from 2012 to 2016. The target audience is machine learning researchers, therefore we first provide in *Section II* a substantial general overview on mobile device security, as well as on Android Operative System and security. The user that is acquainted with such overview can skip this section. The surveyed techniques come in *Section III*; they are organized by the type of analysis that is performed: static, dynamic, permission-based or hybrid. Along with a description of each work, we analyze Machine Learning relevant elements: the features, the dataset, the models and the performance results. We additionally provide a comparative table with all the surveyed works with additional elements (such as the scope, type of malware, monitoring environment and type of learning), which can be consulted *here*. We conclude in *Section IV* with some opportunity areas regarding a Machine Learning approach for Android malware detection, mainly regarding the following: the small size of the datasets, the imbalance class problem, the extremely high number of features, the lack of consistency of the evaluation metrics and error analysis and the need of integrating clustering, dimensionality reduction and classficiation methods into one single model. Also we suggest interesting venues of work regarding a general approach to detect Android malware including: bigger malware datasets, a comprehensive understanding of the currently available datasets, more approaches than integrate the three types of analysis, more hardware-based approchaes, the additional classification of types of malware (trojans, worms rootkits, etc), discussion on the usability, scalability and compatibility with other Operative Systems, consideration of computational complexity, overhead and efficiency, which are specially critical for mobile devices.

II. Mobile device security and Android Operative System and security overview

A. Mobile device security

According to NIST Computer Security Handbook, *Computer Security* is defined as "the protection afforded to an automated information system in order to attain the applicable objectives of preserving the *integrity, availability, and confidentiality* of information system resources" [Stallings2007].

The *OSI security architecture* allows to organize the task of providing security using the following concepts [Stallings2007]:

- Security attack/threat: defined as any action that compromises the security of information.
- Security mechanism: defined a process that is designed to detect, prevent, or recover from a security attack.

According to [Stallings2007], maybe the most sophisticated threats to computer systems are programs that are built to exploit vulnerabilities in computing systems. Such threats are referred to as malicious software, or **malware**. In this context, we consider threats to application programs, utility programs and kernel-level programs [Stallings2007].

The most important examples of malware in mobile devices are virus, worms, Trojans, rootkits and botnets. Virus injects malicious code into existing programs that is replicated to other programs when the code is executed. Worms are spread over the network and exploit vulnerabilites on the computers that are connected to the network. Trojans appear to provide functionality to hide its malicious content. Rootkit directly infects the Operative System, so they can operate freely and for longer periods; usually they hide malicious user-space processes/files, installing Trojans or disabling firewalls[1] and antiviruses. Finally, botnet is a network of infected devices under command and control of an attacker [Stallings2007]

[1]A firewall is a barrier that dictates which traffic is authorized to pass in each direction. It can operate at the level of IP packets, or at a higher protocol layer [Stallings2007].

[La Polla et al.2013]; usually they are used to send spam or perform Denial of Service attacks (DoS)[2].

Mobile malware can disseminate through various paths, such as a spam SMS with a link to a website where a user is able to download the malicious code, or a spam MMS with infected attachments, or even infected programs that are sent and received via Bluetooth [La Polla et al.2013].

Some malware fill devices with unwanted advertisements to gain revenue for the malware creator. Others can dial and text so-called premium services resulting in extra phone bill charges. Some other malware is even more insidious, hiding itself (via rootkits or background processes) and collecting private data like GPS location or confidential documents. [Demme et al.2013].

Malware detection techniques can be classified in *Static Analysis, Dynamic Analysis and Permission-based Analysis*. Is it also possible to have an hybrid of them [Amos et al.2013].

- *Static Analysis:* inspects software properties and source code without executing the application, so it is performed using parameters such as code analysis, taint tracing and control flow dependencies [Dua and Bansal2014]. It is **inexpensive**, thus amicable to memory-limited mobile devices, but obfuscation, polymorphism and encryption techniques embedded in software makes it difficult. For example, software typically use static characteristics of malware such as suspicious strings of instructions in the binary to detect threats. Unfortunately, it is quite easy for malware writers to produce many diferent code variants that are functionally equivalent, both manually and automatically- For instance, one malware family in the AnserverBot dataset of [Demme et al.2013] had 187 code variations. It also produces less information, thus limiting the extraction of possible features that can be used by machine learning algorithms [Narudin et al.2016]. This type of analysis can be classified according to the way of detection [Dua and Bansal2014]:
  - Misuse detection: uses a signature for detection of malware based on security policies and rule-sets by matching of signatures.
  - Anomaly detection: uses algorithms to identify malware based on certain features, so signatures are not needed. Surprisingly, Machine Learning algorithms commonly appear in this class because they are useful to *learn* the features or characteristics of known malwares and predict unknown malware based on this learning [Dua and Bansal2014].
- *Dynamic Analysis:* it monitors malware behaviour when the application is executed. It can be performed using parameters such as native code, system calls [3], network traffic[4], used memory and user interaction [Dua and Bansal2014]. It also produces more information than static analysis, which can be used to build or enrich features [Narudin et al.2016]. Obviously this comes with the associated overhead [Demme et al.2013], which is an important shortcoming for mobile devices resource-constrained nature. Hence, dynamic analysis commonly is **expensive** and more difficult to scale. Additionally, some malwares even perform transformation attacks that changes not only their code but also its behavior during runtime [Wu and Hung2014].
- *Permission-based analysis* can be done with the help of permissions specified in the manifest file.
  Permission-based: Permissions are listed in some file and control the access to several types of resources in the device. Users have the right to allow or deny the installation of applications but the individual permissions are already defined. It is possible to detect malware using the permissions in the Manifest.xml.

It is important to point out that security tools and mechanisms used in personal computers are not feasible for applying on mobile devices due to its enormous resource consumption and battery depletion [Burguera et al.2011], which puts more pressure on the energy demands and constraints.

B. Android Operative System

Just like all operating systems, Android Operative System enables applications to make use of the hardware features through abstraction and provide a defined environment for applications [Brahler2010].

Android Operating System can be visualized as a stack of software components. Its source code is released by Google under open source licenses [Brahler2010]. Its architecture is formed by 4 main layers placed one on top of each other that can be visualized in Figure 1 (taken from [Rashidi and Fung2015]):

1) *Linux Kernel*: placed at the base, it is the base Operating System. This means that all requests made from upper layers should pass through the kernel using a system call interface before they're executed in hardware [Burguera et al.2011].
2) *Libraries and runtime environment:* Android runtime combines the core libraries of the Java Virtual Machine and Dalvik Virtual Machine, which is responsible for running Android applications in the operating system.

[2]In a DoS, the attacker seeks to make a network resource unavailable to its intended users -for example an online store) by disrupting the services of a host connected to the Internet

[3]A system call is how a program requests a service from the operating system's kernel, so, analyzing this may help to identify unauthorized requests.

[4]The idea is that almost every every application, malign or benign must connect no the network. So network traffic may help to identify malign behaviour

3) *Application Framework:* it puts a particular structure on developers, for example, it doesn't have a *main()* function or single entry point for execution. The developers must design applications in terms of components, which can be seen in Figure 1 [Enck et al.2009].
4) *Applications:* they are written in Java and executed in its own Dalvik Virtual Machine. They involve both the pre-installed applications provided with a particular Android implementation and third-party applications developed by individuals -hence unofficial- app developers [Rashidi and Fung2015].

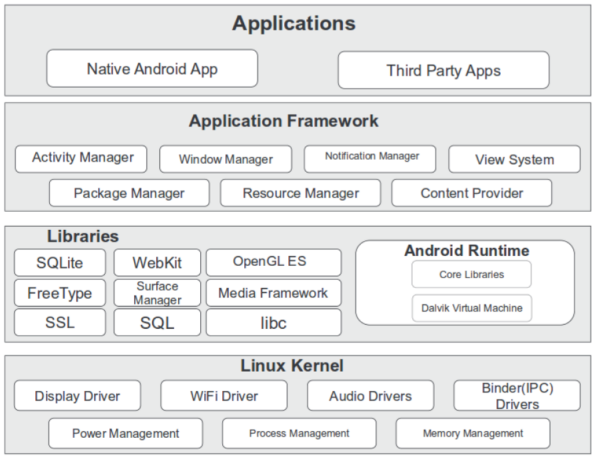

Fig. 1. Android operating system architecture (taken from [Rashidi and Fung2015])

We briefly describe Android's application structure with focus on the important files, taken from [Peiravian and Zhu2013]:

- Android Application Package file (APK). Each Android application is compiled and packaged in a single file that includes all the application code (.dex files), resources, assets, and manifest file. It uses the .apk extension, so it is easily found.
- Android Manifest file: Commonly refered as AndroidManifest.xml, is one of the most important files. It contains all essential information about the Application. Once an application is run, the first file the Android system looks at is the Manifest file. It can be visualized as a roadmap to ensure that the application can work properly in the Android system. It is worth noting that the Android Operative System will not let an application to access permissions, resources and features that are not specified in the AndroidManiefest file. So, the AndroidManifest.xml provides the first hand information to understand the security settings and characteristics of the Apps. To see a more detailed explanation on Android Manifest file, please refer to [Enck et al.2009].

We will discuss now how the security is organized, managed and deployed in Android.

C. Security in Android

Android applications make use of advanced hardware, innovative software and a large amount of data with the final objective of bringing value to the consumers. So, the platform must offer an environment that guarantees the security of the user, the data, the device, the network and the applications themselves. Despite the fact that Android is based on Linux, it is not straightforward to follow the same desktop analysis approach for Android security [Yan and Yin2012].

There are several reasons of why Android security is specially critical:

- Number of users of Android OS: 1.1 billion devices running on Google's Android Operating System were shipped in 2014, marking its >80% of mobile market share.
- Number of Android Applications: 1.6 million in early 2015, surpassing its major competitor, Apple Apps Store [Rashidi and Fung2015].
- Number of Android-targeted malware: In 2013 F-secure reported 827 new families or variants of mobile malware; most of them were based on Android platform [Kang et al.2015].
- Number of Android threats and vulnerabilities: [Felt et al.2011] found that one of the main threats posed by malicious Android applications are privacy violations which leak sensitive information such as location information, contact data, pictures, SMS messages, etc. to the attacker. But even applications that are not malicious and were carefully programmed may suffer from such leaks, for instance when they contain advertisement libraries. Additionallt, threats can occur in any of the above-mentioned layers of Android OS stack, such as application or framework layer.
- Third-party market applications: since Android OS is an open source platform, it allows the installation of third-party applications, with regional and international app-stores.
- Developers decision power: few of them fully understand their privacy implications [Arzt et al.2014].
- Users decision power: on whether an app is safe to use. This puts too much responsibility on the end users.
- Android specific characteristics: Android Operative System is different in several ways: application framework, touch-screen based graphical user interface, and management of personal data [Wu and Hung2014].

Android protects applications and data through a combination of two enforcement mechanisms, one at the system level placed on the Linux Kernel base of Figure 1 and the other as a middleware placed between the Kernel and the Application level. This middleware defines the differentiated and core Android security framework but it is built on the fundamental guarantees granted by the underlying Linux system [Enck et al.2009].

Android security model can be considered system centric [Shabtai et al.2012] as it is highly based on a permission-based mechanism. There are about 130 permissions that govern access to different resources. An Android application requires several permissions to work. Applications statically identify the permissions that define their right to resources and information, and interfaces at installation time [Peiravian and Zhu2013]. However, the application/developer has limited ability thereafter to decide to whom those rights are given or how they are used. Consequently, an essential step to successfully install an Android application into a mobile device is to allow the installation of all the permissions requested by the application. In fact, before an application is installed, a list of permissions requested by the application is prompted onto the screen and it asks the user to confirm them for installation.

Although permission requests are useful for users to prevent possible misuse of resources by applications, users often have rare knowledge to determine if permissions might be harmful or not. The application may request network access, including Wifi and SMS service, which are normal requests, whereas some malware steal bandwidth or other useful information. So it's very difficult for users to determine, at the first place, whether an App is a malware by using the permission request exclusively [Peiravian and Zhu2013].

At the system level, Google publicly announced that a security checking mechanism is applied to each application that is uploaded to their market. The open design of the Android operating system allows a user to install any applications he want no matter that they are downloaded from an untrusted source. Yet, the permission list is still the minimal (and almost only) defense for a user to detect whether an application could be potentially harmful. For example, a careful user can choose not to install an application if he notices that it unnecessarily requests permission to his personal contact or other private resources such as its location [Peiravian and Zhu2013].

Google further classifies adds permission protection levels and categorize the application into normal, dangerous, signature, and signature/system [Huang et al.2013]. However, this baseline security framework has not prevented the proliferation of malicious applications in Android Operative System.

For a more exhaustive explanation of additional security refinements taken by Android, check the following sections from [Enck et al.2009]: public vs private components, broadcast intent permissions, content provider permissions, service hooks, protected APIs, pending intents and URI permissions.

III. Survey

In addition to the mechanism to detect malware, a critical challenge is the need for the collection and experimentation with a large dataset for training malware classifiers [Amos et al.2013]. Is difficult to collect *real* Android malware mainly for two reasons: 1. Android malware is a novel research area and 2. Due to the malign nature of this type of data, labelled examples are not commonly released or revealed. So, before 2011 there have been no tangible datasets available to the whole research community. Researchers used to build their own artificial malware or used a crawler to collect apps from the internet. [Burguera et al.2011] presented a crowd-sourcing system to collect real samples of application execution traces. Also, in 2012 it was published a comprehensive dataset called MalGenome that comprises of 49 different Android malware families with a total of 1,260 malware samples [Narudin et al.2016].

Moreover, [Amamra et al.2012] points out the importance of an algorithmic and replicable approach: "malware classifiers must be trained and evaluated in a repeatable and consistent manner with large-scale experimentation and automation infrastructure" [Amamra et al.2012].

Due to the relevance of the problem, several surveys on mobile security threats and mechanisms have been published. Such mechanisms are different in nature and in its methods and can be classified into different categories. We can mention some of the most recent works: [Sujithra and Padmavathi2012], [Amamra et al.2012], [La Polla et al.2013], [Dua and Bansal2014], [Rashidi and Fung2015], [Faruki et al.2015], [Khan et al.2015]. Some are focused specifically on Android, for example [Faruki et al.2015] and [Rashidi and Fung2015], and some others are broader, such as [La Polla et al.2013] and [Suarez-Tangil et al.2014b]. In these works, authors have organized and classified their work using: feature misuse, attack goals, distribution, infection, privilege acquisition, type of analysis, type of detection [Faruki et al.2015], etc. As of our knowledge, there is no published survey that is focused on a **machine learning analysis** of the defense mechanisms against Android threats. We choose to classify the survey using the type of analysis that is performed, i.e. static, dynamic, hybrid or permission-based, instead of a machine-learning type classification because we realized that this analysis approach is widely used by researchers in the malware detection literature, so it will give us more insights on how and where the machine learning techniques are really being used. We ordered the reviewed works by year of appearance.

To detect malware, static analysis commonly utilizes the manifest file of the Android applications discussed above and retrieves information such as permissions and API calls. Dynamic analysis focuses on the run-time behavior and the system metrics of the applications while running. Additionally, most of the dynamic analysis methodologies depend upon executing the applications in an emulator to collect the run-time information [Wu and Hung2014].

A. Machine Learning for Static Analysis

[Hanna et al.2012] presented Juxtapp, a scalable system to perform automatic code similarity analysis on Android applications. The similarity approach helps to detect code reuse, hence, it can determine if an app contains copies of buggy code that may indicate piracy, or is an occurrence of known malware. They did not addressed a classification problem as most of the works we will discuss later, instead they addressed a clustering and dimensionality-reduction problem. First, they divided the code in k-grams sequences to create many features, then they used feature hashing[5] to reduce the dimensionality of the feature set. Later, they computed Jaccard similarity between the feature sets, together with agglomerative hierarchical clustering to group the applications. The system was evaluated using more than 58,000 Android applications and demonstrated that it is able to scale. Juxtapp detected 463 applications with buggy code reuse of Google-provided sample code (that may lead to serious vulnerabilities in real-world apps). It also detected 34 instances of known malware (specifically trojans) and 13 variants of the GoldDream malware, as well as pirated variants of a popular paid game with notable code variation from the original. While this approach is interesting because it performs not-supervised learning, it doesn't perform a supervised phase, hence it is difficult to compare its performance with other methods.

[Wu et al.2012] is a system aimed to label and classify the applications, so they address both a clustering and classification problem. They used the following features: requested permissions, intent messages from each application's manifest file, and regards components such as activity, service collected from permissions and system calls. They applied k-means algorithm to cluster the observations. The number of clusters is chosen using Singular Value Decomposition (SVD) method on the low rank approximation. Finally, they used k-nn for the classification task. The dataset consists of 1,500 benign and 238 malign from Contagio Mobile platform. The approach obtained 97.8% of accuracy but only 87% of recall, which means that from 100 real-malware samples, DroidMat is able to detect 87 samples and the other 13 are classified as benign, which could have serious consequences. So, DroidMat should be used -if used at all because there are other static techniques that have better performance- as a pre-filter, not as a unique classification method.

[Huang et al.2013] followed an approach that monitors the application permissions, using features from the corresponding application package file (APK) and the Android Manifest file discussed above. They addressed a clustering problem, followed by a classification problem. Rule-based was used for clustering and labeling. They applied several machine learning classifiers: AdaBoost, Naive Bayes, Decision Tree and Support Vector Machine. The dataset consisted of 124,769 benign and 480 malign applications. The results were 81% in the recall for naive bayes classifier, that is, from each 100 malware samples, the method is unable to detect 19 of them for the naive bayes classifier. This result in accuracy is low, so it must be used as a quick filter to identify malicious applications. The method still requires a second pass to achieve reasonable results. Also, as labeling is rule based, so it is difficult to scale and generalize.

[Aafer et al.2013] propose DroidAPIMiner, which is claimed to be a robust and lightweight detection mechanism. To detect malware they address a classification problem. The selected features are gathered using API level information that is within the bytecode. The authors claim that it conveys substantial semantics about the apps behavior. The features include critical API calls (choosing the most frequent), their package level information, as well as some dangerous parameters. The dataset has 20,000 benign applications, 3,987 malware apps obtained from McAfee and Android Malware Genome Project. They used Decision trees, k-nn and Support Vector Machine. The results showed that K-nn achieves an accuracy (i.e a detection of correct cases, both malware and non-malware) of 99%, and a True positive rate of 97.8%, meaning that the miss rate is just 2.2 %, so they are able to detect almost all the malware. A potential shortcoming of this proposal is that new malware might easily include more benign API calls into their code with the objective of camouflaging its malign behaviour.

[Peiravian and Zhu2013] tried a combined use of permissions and API calls of Android applications to build high dimension feature vectors. They addressed a classification problem for malware detection. The permission is obtained from each Application's profile information and the APIs are obtained from the packed App file by using packages and classes to be able to represent API calls. The dataset consists of 1200 real-word benign apps and 1200 malware apps. They used Support Vector Machine, decision trees (specifically J48) and Bagging[6]. Bagging has the best performance in classifying all created data sets with respect to AUC (>96%). Because permission settings and APIs are always available for each application, this system can be generalized to other mobile devices in addition to Android.

[Arp et al.2014] was published in response to heavy applications that were prohibitive for mobile devices. The objective is to predict malware, hence it addresses a classification problem. It collects permissions, hardware access, API calls, network address, etc, generating 545,000 features. Dataset consists of 123,453 benign applications and 5,560 malware

[5]Feature hashing applies a hash function to features, providing a fast way of vectorizing features.

[6]Bootstrap-based ensemble method that creates base classifiers with the objective of ensembling them by training each base classifier using a random redistribution of the training set

samples were collected. They used a Support Vector Machine, achieving 94% of accuracy and a very low False Positive Rate (i.e. a false alarm) of 1%. On five popular smartphones, the method required 10 seconds for an analysis on average, so it suitable to run on applications directly on the device. This work tackles the overhead and efficiency problem, which is loosely tackled in many works. Unlike most of the static methods we've discussed, it can identify signs of obfuscation, but the code is not fully available to be analyzed. The number of features is overwhelmingly high, almost five times larger than the number of observations, so this dataset potentially suffers from the curse of dimensionality[7]. In fact, this is the reviewed dataset with the largest number of features.

[Arzt et al.2014] introduced FLOWDROID, a static taint-analysis mechanism for Android applications. The authors addressed a classification problem to be able to predict malware. FLOWDROID is capable of modelling Android-specific challenges like the application cycle or callback methods, hence reducing missed leaks or false positives (i.e. false alarms). Novel on-demand algorithms helped FLOWDROID maintain high efficiency. The authors also proposed DROIDBENCH, an Android-specific platform to evaluate the effectiveness and accuracy of taint-analysis tools for applications. FLOWDROID was tested using DROIDBENCH, achieving 93% recall and 86% precision. This means that of each 100 real malware samples, it fails to detect 7, and that of each 100 samples that were predicted as malware by FLOWDROID, 86 really were malware (14 were a false discovery). Additionally, FLOWDROID also found leaks in a subset of 500 apps from Google Play and about 1,000 malware apps from the VirusShare project.

[Yerima et al.2014a] proposed a static analysis of Android malware using Bayesian classification models built from mining data generated by automated reverse engineering of the Android application packages employing a Java implemented custom package analyzer. They addressed a dimensionality reduction problem followed by a classification problem. The dimensionality reduction was performed using an Information Theory approach, specifically, the Information Gain method, which calculates the entropy generated by each feature; they retrieved only the 25 most frequent features. The classification problem was approached using three models using the following combination of features: standard Android permissions in the Manifest files, code properties indicative of potential malicious payload and both standard permissions and code properties. The models were built by extracting these properties from a set of 1,000 samples of 49 Android different malware families together with another 1,000 benign applications across a variety of categories. So, unlike other datasets, this dataset is not unbalanced, as 50% of the applications are malware. But this approach of building a balanced set is artificial, because actually malware is scarce compared to the non-malware. Evaluation was performed using the confusion matrix usual metrics. Results showed that mixed-based and code property-based models are a better choice than the permissions-only model. Specifically, the mixed-based approach reported an accuracy of 93% and an area under the ROC curve of 0.977; this metric can be used to easily compare methods.

[7]When the dimensionality increases, the volume of the space increases so fast that the available data become sparse, generalizing poorly to new data

[Suarez-Tangil et al.2014a] proposed a text mining approach to classify malware samples into families based on the code structures. So they addressed both a classification and clustering problem. But this work differs to the previously discussed works in that it classify into families of malware (namely 49), not into malware/not-malware. They performed a statistical analysis of the distribution of such structures over a large dataset of real examples. They found strong closeness to some questions that are common in automated text classification and other information retrieval tasks, so they adapted the standard Vector Space Model commonly used in these areas (i.e. vector embeddings). They measured similarity between malware samples, applied hierarchical clustering and then classified unknown samples into known families using k-nn. Although the technique is claimed to be fast and potentially scalable, it was only tested using only 1247 examples. One of the main shortcomings of this approach is that it has too many features, exactly 84,854 and too few observations. This potentially lead to the curse of dimensionality mentioned above. Regardless of this, the method reached a classification error of just 5.74%, which is very high considering there were 49 possible classes. Another disadvantage is that obfuscation techniques could change the code syntactic structure of a malware sample but maintain its malign purpose. So, a semantic approach may be worthwhile trying.

[Kang et al.2015] proposed a detection system that uses serial number information from the certificate as a feature. So this work addresses a classification problem. The classification is performed two times: first to detect malware and then to identify the type of malware family. It checks a serial number and looks for suspicious behavior of SMS, system commands in the code and permission requests from the Manifest file. Hence, the features include serial number, information of the certificate, the application name, requested permission, component, and intent. 51,179 applications were downloaded from Google Play and 4,554 malicious applications were downloaded from Share, Contagio Mobile, and Malware.lu. They used a Bayesian classification model based on a similarity scoring, detecting the malware with 98% of accuracy and the family type with 90% of accuracy. Additionally, the system can help analysts to react efficiently from Android malware's threats by detecting and classifying with high accuracy in a reasonable time. One potential problem of this approach is that they they assume that $P(c_i = malicious) = P(c_i = benign)$, i.e. is equally likely that the the category of the application is malicious than benign, regardless of the inherent imbalance status of the dataset (much more benign apps than malign).

While some of the work on static analysis uses traditional machine learning algorithms for classification problems (support vector machine, decision trees, k-nn), many of the uses other approaches such as Bayesian classification models, and text-mining algorithms ([Yerima et al.2014b] and [Suarez-Tangil et al.2014a]). As of the clustering approaches we can mention the Agglomerative Code similarity and the hierarchical clustering ([Hanna et al.2012] and [Suarez-Tangil et al.2014a]), respectively). All of the proposed approaches address at least a classification problem, except [Hanna et al.2012] that addressed a feature learning, clustering and dimensionality-reduction problem. [Suarez-Tangil et al.2014a] and [Wu et al.2012] followed a sequential approach of clustering to label the observations and then classification to detect the malware; this is an interesting approach because as we mentioned, it is hard to get malware samples. We detect too many features in most of the works and very small datasets, which is a serious problem in Machine Learning and many strategies exist to tackle them, none of them definite. Only [Hanna et al.2012] and [Yerima et al.2014b] performed a dimensionality reduction to reduce the number of features, mainly using the Information Gain method, but many other methods exist, such as Principal Component Analysis and t-SNE.

We now present the surveyed work on Dynamic Analysis.

B. Machine Learning for Dynamic Analysis

[Burguera et al.2011] proposed Crowdroid, a dynamic analysis system that examines application behaviour to detect malware (specifically trojans) in Android. It achieves this by monitoring Linux Kernel system calls and report them to a centralized server. They address a clustering problem to obtain labels for malware samples. This framework is also able to detect self-written malware. They clustered each sample using k-means to differentiate between benign and malicious applications. They collected system call traces coming from an unlimited number of real users based on crowdsourcing; this yielded 20 feature vectors. The authors recognize that if applying this system in a mobile device, it might have an extra overhead in the processor, hence this approach may lack of usability and generality. As the authors only address a clustering problem, it remains difficult to compare this method with others and therefore to assess how good it really is.

[Dini et al.2012] presented MADAM, a Multilevel Anomaly Detector for Android Malware. MADAM simultaneaously monitors Android at the kernel-level (that is, system calls, running processes, free RAM, CPU usage) and user-level (idle/active, key-stroke, called numbers, sent/received SMS, Bluetooth/WI-FI analysis) to detect real malware infections (trojans and rootkits). Therefore they address a classification problem. The training dataset had 900 standard vectors and 100 malicious ones, which were defined manually based on predefined characteristics. MADAM detected 93% of the malware (100% of rootkits). The performance overhead was assessed as acceptable. One shortcoming of MADAM is that the malware samples are defined based on rules, which calls into question its applicability: first, rule-based approaches are difficult to generalize and scale (as opposed to machine learning ones), second, malware tends to change its characteristics over time, so the the rules that once were valid, then may not be.

[Shabtai et al.2012] proposed Andromaly, a dynamic detection system that monitors both the smartphone and user behaviors. They take into consideration several parameters, from sensors activities to CPU usage. Hence, 88 features were used to describe these behaviors: CPU consumption, number of sent packets through the Wi-Fi, number of running processes, battery level, system and usage parameters. 10 datasets were formed from two different devices by activating 44 applications for 10 minutes. They reduced the dimensionality of the dataset using Chi-Square, Fisher Score and Information Gain methods. For the classification they tried k-nn, Logistic Regression, Decision Tree, Bayesian Networks and Naïve Bayes. None of these models performed best in all three experiments. The best results for the experiment 1 are achieved with the decision Tree: area under the ROC curve, true positive rate and accuracy > 0.999 which means that Andromaly correcly classifies almost all the samples. For experiment 2 the results were not so good: Decision tree 0.8-0.9 in area under the ROC curve, true positive rate and accuracy. For experiment 3: Naive Bayes area under the ROC curve > 0.84. For experiment 4: Naive Bayes >0.88 in AUC. One interesting characteristic of Andromaly is that it is open and modular, hence, can easily accommodate multiple malware detection techniques. It is also light in terms of CPU, memory and battery consumption. One shortcoming is that given the different results in each experiments, it remains difficult to compare Andromaly with other methods.

[Ham and Choi2013] defined novel features examining the structural features of Android architecture (defined above in Figure 1) and selecting the optimal to detect mobile malware. They addressed both dimensionality reduction and classification problems. The features include diverse categories regarding: network, SMS, CPU, power monitor, and process category and virtual Memory[8]. Dimensionality reduction was applied using the Gain Information algorithm, so 20 features were finally considered. 11,628 normal instances and 3,876 infected instances (76%, 24%) were collected. The performance was tested using four machine learning algorithms: Random Forest, Support Vector Machine, Logistic Regression and Naive Bayes. Random forest achieved the highest recall (99%) and an area under of ROC curve of 99.8 %.

[8]In Android, when an app is started, only a part of the program is arranged in the memory and a part of the hard disk is used by making it into virtual memory. so, peak memory size placed into virtual memory and shared memory size are also considered as monitoring features

[Amos et al.2013] introduced a system that collects a number of features such as battery, memory, network and permission yielding 6,832 feature vectors. The addressed a classification problem. The authors proposed the STREAM framework, which was developed to facilitate a rapid large-scale validation of mobile malware classifiers. They used Random forest, Naive Bayes, multilayer perceptron, Bayes net, logistic regression and J48 into a dataset with 1,330 malicious and 408 benign applications. However, the authors used Android emulator to collect selected features, which was proved not as accurate as a real device and in fact the authors claimed that using a real device was impossible. The approach also suffers from a high number of features and a very small dataset, which may imply poor generalization.

[Demme et al.2013] examined the feasibility of building a malware detector based in hardware using already existing performance counters. They addressed a classification problem to detect malware. The underlying assumption the authors made is that run-time behavior that is captured using performance counters, can be further utilized to detect malware and that the minor variations in malware that are typically used to cheat signature software do not significantly interfere with the proposed method. They built a multi-dimensional time series data with the count events together with k-nn, Decision Trees, Random Forest and artificial neural networks. The dataset uses 503 malware and 210 non-malware programs from both Android ARM and Intel X86 platforms, achieving a high area under the ROC curve result. One shortcomming of this work is that they don't present quantitative comparable metrics. For example, they don't numerically report the true positive rate, accuracy, etc. Additionally, the ROC curve measure is only computed below 10% false positive rates. Hence, it is difficult to compare this proposed approach with others.

[Yuan et al.2014] proposed a Deep Learning method[9] that utilizes more than 200 features extracted from both static and dynamic analysis of Android applications. The features fall into three types: required permissions, sensitive API and dynamic behavior. This apporach addresses a classification problem. The required permissions and the sensitive API are analyzed using the .apk file of an Android app yielding 184 features. The dynamic behavior is tested running the .apk file in a sandbox named DroidBox[10], yielding 18 features. The deep learning algorithm has 2 phases: the unsupervised pretraining phase and the supervised back-propagation phase and a Deep Belief Network was used for both tasks [11] The model achieved a 96% accuracy with real-world Android application sets, above traditional machine learning models (Support Vector Machine, Naive Bayes, Linear Regression, Multilayer Perceptron). One shortcomming of this proposal is that they don not discuss in detail the size of the used dataset, which is key to deep learning models. They only claim that a malware set of 250 samples was downloaded from contagio mobile and 250 benign apps were downloaded from Google Apps Storem but we don not know how many observations for training the models this dataset generated.

[Wu and Hung2014] proposed a dynamic and partly static malware analysis using application instrumentation, emulation, GUI testing. Automatic tools were used to extract static and dynamic features from a training dataset composed of 32,000 benign and 32,000 malicious applications and a testing set of 1,000 and 1,000 respectively. There are 56,354 features obtained from event combinations of logged data. The results showed that the prediction accuracy reaches 86.1% and F-score reaches 85.7 %. The authors claimed that the accuracy increases significantly with the dataset size, but, as it is, the method only correctly classifies 84% of the samples. The framework should be used in conjunction with other existing works to improve the detection rate, questioning its usefulness.

[Yerima et al.2015] proposes a novel approach using ensemble learning for the detection of malware. The authors address a classification problem. The machine learning models are trained with a large dataset of malign and benign observations from an important antivirus vendor. Unlike Machine Learning, Ensemble learning performance is not negatively impacted with very high number of features. So, a dimensionality reduction is not necessary. The model achieved 97.3% to 99% detection accuracy with very low false positive rates.

[Narudin et al.2016] evaluates the effectiveness of mobile bots detection using network traffic. The address a classification problem. They assess five classifiers, namely a decision tree (J48), Bayes network, multi-layer perceptron, k-nn and random forest using 1,200 malware samples from Android Malware Genome Project and 1,200 benign samples. For this task they used 11 features from 4 groups: basic information, content-based, time-based and connection-based. The results showed that Bayes network and random forest classifiers produced more accurate readings, with a 99.97% true positive rate as opposed to the multi-layer perceptron with only 93.03%. A shortcoming of this approach is that the detection process must be run using cloud services, through which network traffic is analyzed remotely. However this also helps to reduce the overhead. Another shortcoming is that the dataset is artificially balanced, hence questioning the high performance metrics obtained.

Dynamic approaches they tend to run heavy, sometimes requiring to run some processes and analysis in external servers. However, there was not much discussion on this trend

[9]Deep Learning is a subfield of Machine Learning that aims to model high level abstractions in data using deep neural networks. It has been successfully applied to AI-hard problems such as speech and image recognition.

[10]DroidBox collects the runtime activities such as network data, file read and write operations, started services and loaded classes, information leaks using the network, SMS, etc.

[11]This Deep Learning model is composed of multiple layers of hidden units with connections between the layers but not between units within each layer. When used for pretraining, it can learn to probabilistically reconstruct its inputs.

or quantitative and consistent measures in this regard. The revised approaches also tend to have too many features, with the associated overfitting and curse of dimensionality disadvantage we already mention before for the static analysis. The proposals mainly focus on a classification problem, they rarely performed clustering; many more static analysis approaches combined classification with clustering and dimensionality reduction.

The first neural network approach were used by [Amos et al.2013] and [Demme et al.2013], using a multilayer perceptron; the problem is that the datasets are very small and neural networks usually need large datasets to learn the high number of parameters they have. [Yuan et al.2014] proposed the first Deep Learning approch but little discussion on the dataset size was found. [Yerima et al.2015] explored an alternative type of learning, namely Ensemble learning, which can be promising for the presented datasets, with very high number of features. Finally, we only found one hardware-based approach, namely [Demme et al.2013].

We describe next the permission-based approaches.

C. Machine Learning for Permission Analysis

[Sanz et al.2013] proposed PUMA, a method for detecting malicious Android applications by analysing the permissions. 1,811 benign Android Application samples and 239 malware samples were collected, so the dataset is unbalanced. They used Random Forest, Naive Bayes, SVM, Logistic. The features of the dataset were the collected permissions from the Android Manifest file. The results using cross-validation showed a 0.92 area under the ROC curve with the random forest classifier, 86.37% of accuracy but 19% true positive rate. This means that if this scheme is used, it should be used as a pre-filter.

[Peng et al.2012] introduce the notion of risk scoring and ranking for Android applications. They propose to use probabilistic generative models for risk scoring schemes based on permissions[12], ranging from the simple Naive Bayes (PNB), to advanced hierarchical mixture models (HMNB). The Benign Dataset is composed of 2 datasets from Google Play 157,856 apps for training and testing, 324,658 apps for validation. The Malware dataset of 378 unique .apk files that are known to be malicious. The features of the dataset are the permissions from the Android Manifest file. The results showed that all three generative models achieve an area under the ROC curve above 0.94, HMNB achieving the highest score. However, PNB is more suitable because it has the monotonicity property of the ranking scheme (i.e. removing a permission always reduces the risk value of an app).

[12]So they assume that some parametrized random process generates the application datasets and learn the parameter value θ that best explain the data. Next, for each application they compute $p(a_i|\theta)$, the probability that the app's data is generated by the model.

[Aung and Zaw2013] implement a framework that extracts several permission features from several downloaded applications from android markets. The features were collected from the corresponding APK file. For each application, the authors identified real permissions required by the application using a binary label. The used two datasets of 500 and 200 applications with 160 features each. They applied dimensionality reduction of the space of variables using Information Gain method followed by k-means clustering and finally, classification using three tree-based approaches. The results showed that the Random Forest achieves 92% accuracy and TPR. The paper doesn't provide details on the results of the clustering algorithm. For example, we would like to know how balanced were the resulting classes in order to be able to interpret the confusion matrix results. Also, the False Negative Rate is high: 8%, which implies that 8% of the time, the algorithm classifies a malign application as benign.

[Frank et al.2012] used a probabilistic model to mine permission request patterns from Android and Facebook applications. They used a method for Boolean matrix factorization to find intersecting clusters of permissions. They used a dataset of 188,389 Android applications. They found that the permission requests of applications with low reputation differ from the permission request patterns of applications with high reputation. This may suggest that permission request patterns can be utilized to construct a risk metric of the quality, and hence, security, of new applications. For Android, the authors found that there is indeed a relationship between permission request patterns and categories. One shortcoming of this approach is that it outputs a risk metric, not a classification, thus it serves much more as way to identify user satisfaction rather than application maliciousness.

In addition to the already mentioned shortcomings of each one of the discussed proposals, the permission-based approaches have the following disadvantages, pointed out by [Aafer et al.2013]:

- The existence of a some permissions in the application manifest file does not necessarily mean that it is actually used within the code. This means that many Android applications have more privileges than they should.
- Many requested permissions -specially the critical- are actually not written in the application's code itself, instead they are asked by advertisement programs.
- Malware is in fact capable of performing malicious behavior without any permission.

D. Machine Learning for hybrid analysis

In this subsection we present a work with an hybrid approach to detect malware.

[Spreitzenbarth et al.2015] present Mobile-Sandbox, a static and dynamic analyzer. In the static analysis a parse of the application's Manifest file and decompilation of the application is done. In the static phase, the application determines if the application is suspicious looking permissions or intents. Then the sandbox performs the dynamic analysis executing the application in order to log all performed actions including those stemming from native API calls. Finally they combine all of these results and try to detect malicious applications with the help of machine-learning techniques. The dataset consists of 69,223 apps from the most important Asian markets and 6,162 malicious samples from different malware families. The features were transformed into a bag-of-words representation. The classification algorithm used was Support Vector Machine achieving an accuracy of 94% of the malware an a FPR of 1%. An interesting characteristic of Mobile-Sandbox is that it can track native API calls, and is easily accessible through a web interface.

IV. Conclusions and future work

Along this survey, we have identified some opportunity areas regarding a machine learning approach to detect malware in Android mobile devices:

- As we have seen through this work, malicious individual data is is very hard to sample. Also, malicious activities can last very little. So there is insufficient data to learn from or detect.
- As there are not enough malicious applications to use in the training phase, most of the presented methods suffer from class imbalance. Many papers reported Accuracy as a means to evaluate their models. The problem is that a dataset suffering from class imbalance will be uninformative in terms of Accuracy. These observations were very rarely discussed in the papers.
- Many papers collected a very high number of features (i.e. dimensions) combined with very few observations. This yields to the well-known problem of the curse of dimensionality: when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. The amount of data needed to achieve significance grows exponentially with the dimensionality. The curse of dimensionality also implies that the model will generalize poorly, i.e. often over-fitting on the training set and achieving poor performance in unseen observations.
- More exploration of techniques for dimensionality reduction. Most of the works that performed dimensionality reduction used methods based on Information Theory (i.e. Information Gain), but there exist many other methods, such as Principal Component Analysis and t-distributed stochastic neighbor embedding (see [Maaten and Hinton2008]).
- None of the revised works included an error analysis, i.e. an analysis that helps to understand why the algorithm performed successfully in some observations and unsuccessfully in others. This also includes understanding which are the features that contribute more to the performance (for example using an incremental analysis). This endeavour is key to really understand the models and to produce better ones.
- The need of more approaches that integrate clustering methods, dimensionality reduction and classification methods. First, clustering can help to obtain labeled datasets (i.e. benign and malware) if the underlying dataset do have both types of observations (regardless of the researcher ignoring the true labels). Once the dataset is labelled, a dimensionality reduction and classification schemes may follow.
- The malicious behavior may vary between attacks, resulting in many types of malicious behaviors. The learnt algorithms are not definite and final, so a continuously adapting machine learning scheme should be implemented. This implies much more overhead than initially considered, potentially making the scheme unaffordable due to the mobile device particular resource constriction.
- There is not an standardized way to measure the performance of the models. While some papers present many classification metrics such as Accuracy, TPR, FPR, TNR, area under the ROC curve, etc, some others only present the accuracy, which, as we have seen, es misleading.
- There is not a clear description of the dataset used to evaluate the performance of the models. While some papers report using Cross validation techniques and others splitting the dataset into training and test, some others omit this explanation.

With the objective of improving the malware detection models for Android devices (not necessarily with a machine learning perspective), we can identify the following interesting venue of work:

- The need of bigger malware datasets. This will condition the advancement of the models themselves, as they are not only evaluated, but also trained using the available data.
- A comprehensive understanding of the currently available datasets. This will guide the work towards the needs in this area.
- The need of more approaches that integrate the three types of analysis (dynamic, static and permissions-base) in a modular approach.
- Most of the machine learning methods will fail to detect instantaneous and abrupt attacks, so it may be worthwhile to combine machine learning detectors with misuse-based detectors, which are rule-based or knowledge-based.
- The need of more hardware-based approaches, as they tend to be more efficient than software-based. We only found one such type of work, e.g. [Demme et al.2013].
- The need of models that, in addition to identify malware, are also able to classify the malware by type/family.
- Discussion on the usability, scalability and compatibility

with other Operative Systems.

- While there are consistently reported classification metrics i.e. effectiveness of the detection, we did not find this for the computational and mobile device efficiency side, for example, a classic computational complexity measure, mobile device resource-consumption metrics (for example CPU, memory and battery consumption), overhead metrics, etc. Many proposals claim to be "very efficient" but they rely on external applications to perform certain tasks and analysis (mostly dynamic-oriented analysis, for example sandboxes or emulators). Hence, currently this is a very vague notion as it is really hard to compare the proposals in this dimension. This will impact the real scalability of the systems and will guide the needs in this path.

REFERENCES

[Aafer et al.2013] Yousra Aafer, Wenliang Du, and Heng Yin. 2013. Droidapiminer: Mining api-level features for robust malware detection in android. In *International Conference on Security and Privacy in Communication Systems*, pages 86–103. Springer.

[Amamra et al.2012] Abdelfattah Amamra, Chamseddine Talhi, and Jean-Marc Robert. 2012. Smartphone malware detection: From a survey towards taxonomy. In *Malicious and Unwanted Software (MALWARE), 2012 7th International Conference on*, pages 79–86. IEEE.

[Amos et al.2013] Brandon Amos, Hamilton Turner, and Jules White. 2013. Applying machine learning classifiers to dynamic android malware detection at scale. In *2013 9th international wireless communications and mobile computing conference (IWCMC)*, pages 1666–1671. IEEE.

[Arp et al.2014] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck. 2014. Drebin: Effective and explainable detection of android malware in your pocket. In *NDSS*.

[Arzt et al.2014] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *ACM SIGPLAN Notices*, 49(6):259–269.

[Aung and Zaw2013] Zarni Aung and Win Zaw. 2013. Permission-based android malware detection. *International Journal of Scientific and Technology Research*, 2(3):228–234.

[Brahler2010] Stefan Brahler. 2010. Analysis of the android architecture. *Karlsruhe institute for technology*, 7:8.

[Burguera et al.2011] Iker Burguera, Urko Zurutuza, and Simin Nadjm-Tehrani. 2011. 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.

[Demme et al.2013] John Demme, Matthew Maycock, Jared Schmitz, Adrian Tang, Adam Waksman, Simha Sethumadhavan, and Salvatore Stolfo. 2013. On the feasibility of online malware detection with performance counters. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 559–570. ACM.

[Dini et al.2012] Gianluca Dini, Fabio Martinelli, Andrea Saracino, and Daniele Sgandurra. 2012. Madam: a multi-level anomaly detector for android malware. In *International Conference on Mathematical Methods, Models, and Architectures for Computer Network Security*, pages 240–253. Springer.

[Dua and Bansal2014] Lovi Dua and Divya Bansal. 2014. Taxonomy: Mobile malware threats and detection techniques. *International Journal of Computer Science & Information Technology*, 6.

[Enck et al.2009] William Enck, Machigar Ongtang, Patrick Drew McDaniel, et al. 2009. Understanding android security. *IEEE security & privacy*, 7(1):50–57.

[Faruki et al.2015] Parvez Faruki, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. 2015. Android security: a survey of issues, malware penetration, and defenses. *IEEE Communications Surveys & Tutorials*, 17(2):998–1022.

[Felt et al.2011] Adrienne Porter Felt, Matthew Finifter, Erika Chin, Steve Hanna, and David Wagner. 2011. A survey of mobile malware in the wild. In *Proceedings of the 1st ACM workshop on Security and privacy in smartphones and mobile devices*, pages 3–14. ACM.

[Felt et al.2012] Adrienne Porter Felt, Elizabeth Ha, Serge Egelman, Ariel Haney, Erika Chin, and David Wagner. 2012. Android permissions: User attention, comprehension, and behavior. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 3. ACM.

[Frank et al.2012] Mario Frank, Ben Dong, Adrienne Porter Felt, and Dawn Song. 2012. Mining permission request patterns from android and facebook applications. In *2012 IEEE 12th International Conference on Data Mining*, pages 870–875. IEEE.

[Ham and Choi2013] Hyo-Sik Ham and Mi-Jung Choi. 2013. Analysis of android malware detection performance using machine learning classifiers. In *2013 International Conference on ICT Convergence (ICTC)*, pages 490–495. IEEE.

[Hanna et al.2012] Steve Hanna, Ling Huang, Edward Wu, Saung Li, Charles Chen, and Dawn Song. 2012. Juxtapp: A scalable system for detecting code reuse among android applications. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, pages 62–81. Springer.

[Huang et al.2013] Chun-Ying Huang, Yi-Ting Tsai, and Chung-Han Hsu. 2013. Performance evaluation on permission-based detection for android malware. In *Advances in Intelligent Systems and Applications-Volume 2*, pages 111–120. Springer.

[Kang et al.2015] Hyunjae Kang, Jae-wook Jang, Aziz Mohaisen, and Huy Kang Kim. 2015. Detecting and classifying android malware using static analysis along with creator information. *International Journal of Distributed Sensor Networks*, 2015:7.

[Khan et al.2015] Jalaluddin Khan, Haider Abbas, and Jalal Al-Muhtadi. 2015. Survey on mobile user's data privacy threats and defense mechanisms. *Procedia Computer Science*, 56:376–383.

[La Polla et al.2013] Mariantonietta La Polla, Fabio Martinelli, and Daniele Sgandurra. 2013. A survey on security for mobile devices. *IEEE communications surveys & tutorials*, 15(1):446–471.

[Maaten and Hinton2008] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605.

[Narudin et al.2016] Fairuz Amalina Narudin, Ali Feizollah, Nor Badrul Anuar, and Abdullah Gani. 2016. Evaluation of machine learning classifiers for mobile malware detection. *Soft Computing*, 20(1):343–357.

[Peiravian and Zhu2013] Naser Peiravian and Xingquan Zhu. 2013. Machine learning for android malware detection using permission and api calls. In *2013 IEEE 25th International Conference on Tools with Artificial Intelligence*, pages 300–305. IEEE.

[Peng et al.2012] Hao Peng, Chris Gates, Bhaskar Sarma, Ninghui Li, Yuan Qi, Rahul Potharaju, Cristina Nita-Rotaru, and Ian Molloy. 2012. Using probabilistic generative models for ranking risks of android apps. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 241–252. ACM.

[Rashidi and Fung2015] Bahman Rashidi and Carol Fung. 2015. A survey of android security threats and defenses. *Journal of Wireless Mobile Networks, Ubiquitous Computing, and Dependable Applications*, 6.

[Sanz et al.2013] Borja Sanz, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo Álvarez. 2013. Puma: Permission usage to detect malware in android. In *International Joint Conference CISIS'12-ICEUTE´ 12-SOCO´ 12 Special Sessions*, pages 289–298. Springer.

[Shabtai et al.2012] Asaf Shabtai, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. 2012. "andromaly": a behavioral malware detection framework for android devices. *Journal of Intelligent Information Systems*, 38(1):161–190.

[Sohr et al.2011] Karsten Sohr, Tanveer Mustafa, and Adrian Nowak. 2011. Software security aspects of java-based mobile phones. In *Proceedings of the 2011 ACM Symposium on Applied Computing*, pages 1494–1501. ACM.

[Spreitzenbarth et al.2015] Michael Spreitzenbarth, Thomas Schreck, Florian Echtler, Daniel Arp, and Johannes Hoffmann. 2015. Mobile-sandbox: combining static and dynamic analysis with machine-learning techniques. *International Journal of Information Security*, 14(2):141–153.

[Stallings2007] William Stallings. 2007. *Network security essentials: applications and standards*. Pearson Education India.

[Suarez-Tangil et al.2014a] Guillermo Suarez-Tangil, Juan E Tapiador, Pedro Peris-Lopez, and Jorge Blasco. 2014a. Dendroid: A text mining approach to analyzing and classifying code structures in android malware families. *Expert Systems with Applications*, 41(4):1104–1117.

[Suarez-Tangil et al.2014b] Guillermo Suarez-Tangil, Juan E Tapiador, Pedro Peris-Lopez, and Arturo Ribagorda. 2014b. Evolution, detection and analysis of malware for smart devices. *IEEE Communications Surveys & Tutorials*, 16(2):961–987.

[Sujithra and Padmavathi2012] M Sujithra and G Padmavathi. 2012. Mobile device security: A survey on mobile device threats, vulnerabilities and their defensive mechanism. *International Journal of Computer Applications*, 56(14).

[Wu and Hung2014] Wen-Chieh Wu and Shih-Hao Hung. 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*, pages 247–252. ACM.

[Wu et al.2012] Dong-Jie Wu, Ching-Hao Mao, Te-En Wei, Hahn-Ming Lee, and Kuo-Ping Wu. 2012. Droidmat: Android malware detection through manifest and api calls tracing. In *Information Security (Asia JCIS), 2012 Seventh Asia Joint Conference on*, pages 62–69. IEEE.

[Yan and Yin2012] Lok Kwong Yan and Heng Yin. 2012. Droidscope: seamlessly reconstructing the os and dalvik semantic views for dynamic android malware analysis. In *Presented as part of the 21st USENIX Security Symposium (USENIX Security 12)*, pages 569–584.

[Yerima et al.2014a] Suleiman Y Yerima, Sakir Sezer, and Gavin McWilliams. 2014a. Analysis of bayesian classification-based approaches for android malware detection. *IET Information Security*, 8(1):25–36.

[Yerima et al.2014b] Suleiman Y Yerima, Sakir Sezer, and Igor Muttik. 2014b. Android malware detection using parallel machine learning classifiers. In *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*, pages 37–42. IEEE.

[Yerima et al.2015] Suleiman Y Yerima, Sakir Sezer, and Igor Muttik. 2015. High accuracy android malware detection using ensemble learning. *IET Information Security*, 9(6):313–320.

[Yuan et al.2014] Zhenlong Yuan, Yongqiang Lu, Zhaoguo Wang, and Yibo Xue. 2014. Droid-sec: Deep learning in android malware detection. In *ACM SIGCOMM Computer Communication Review*, volume 44, pages 371–372. ACM.