Chapter Three

Machine Learning Classification Algorithms in Security:

A Literature Survey

3.1 Problems of the open-source nature of Android

3.1.1 Openness

Android system is an open-source and popular platform due to the flexibility and convenience of its usage, which facilitate the rapid innovation, and how almost every need is covered by an Android application

But at the same time, this openness has led to many information security issues, risks, and wide deployment of diverse types of malware. Those vulnerabilities continue to increase every year in number and in the ways of how malicious activity is conducted, rendering traditional defense mechanisms ineffective.

The problem with Android is that unlike other platforms, it is possible to install Android applications from unverified third-party sources, which ease the distribution of malware for attackers. And thus, determine the trustworthiness of these applications becomes a challenging task.

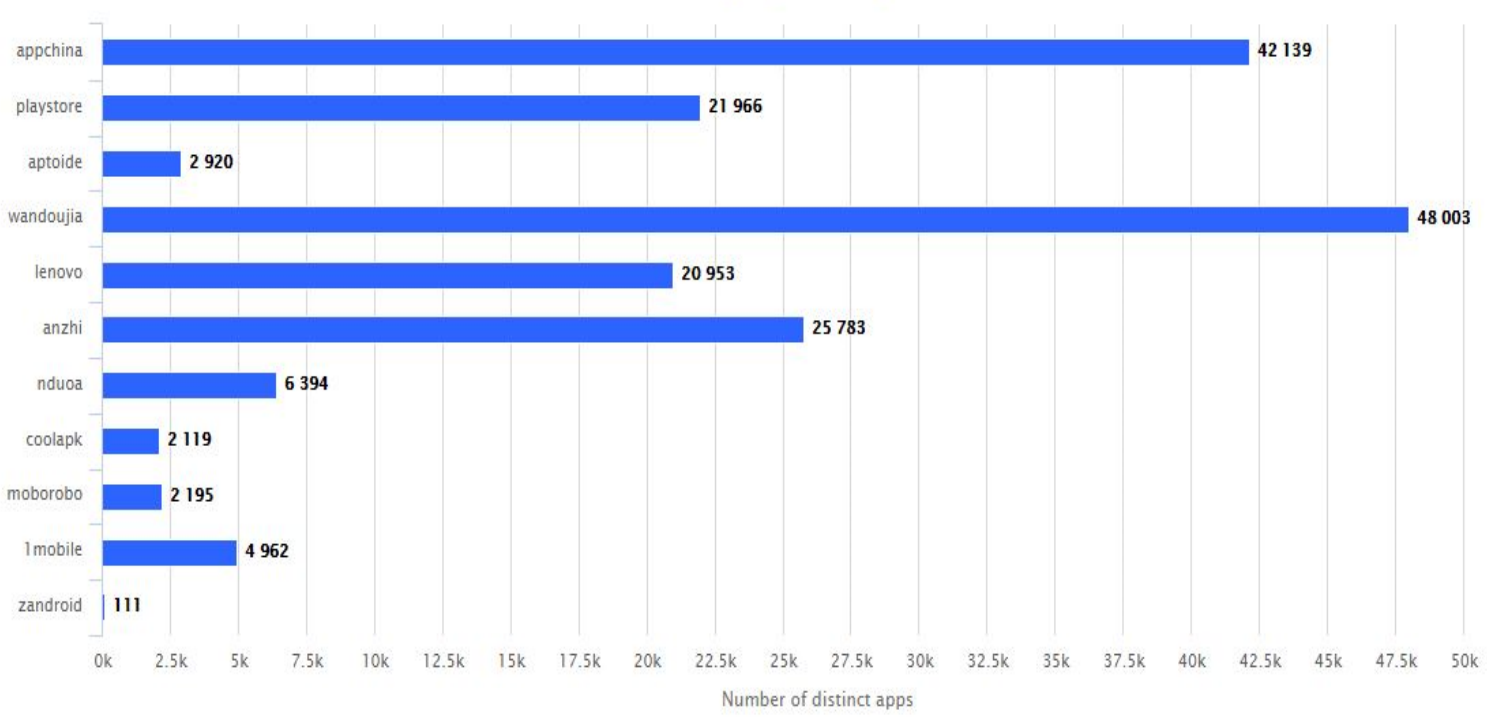

Figure 3.1: Malicious apps per marketplace.

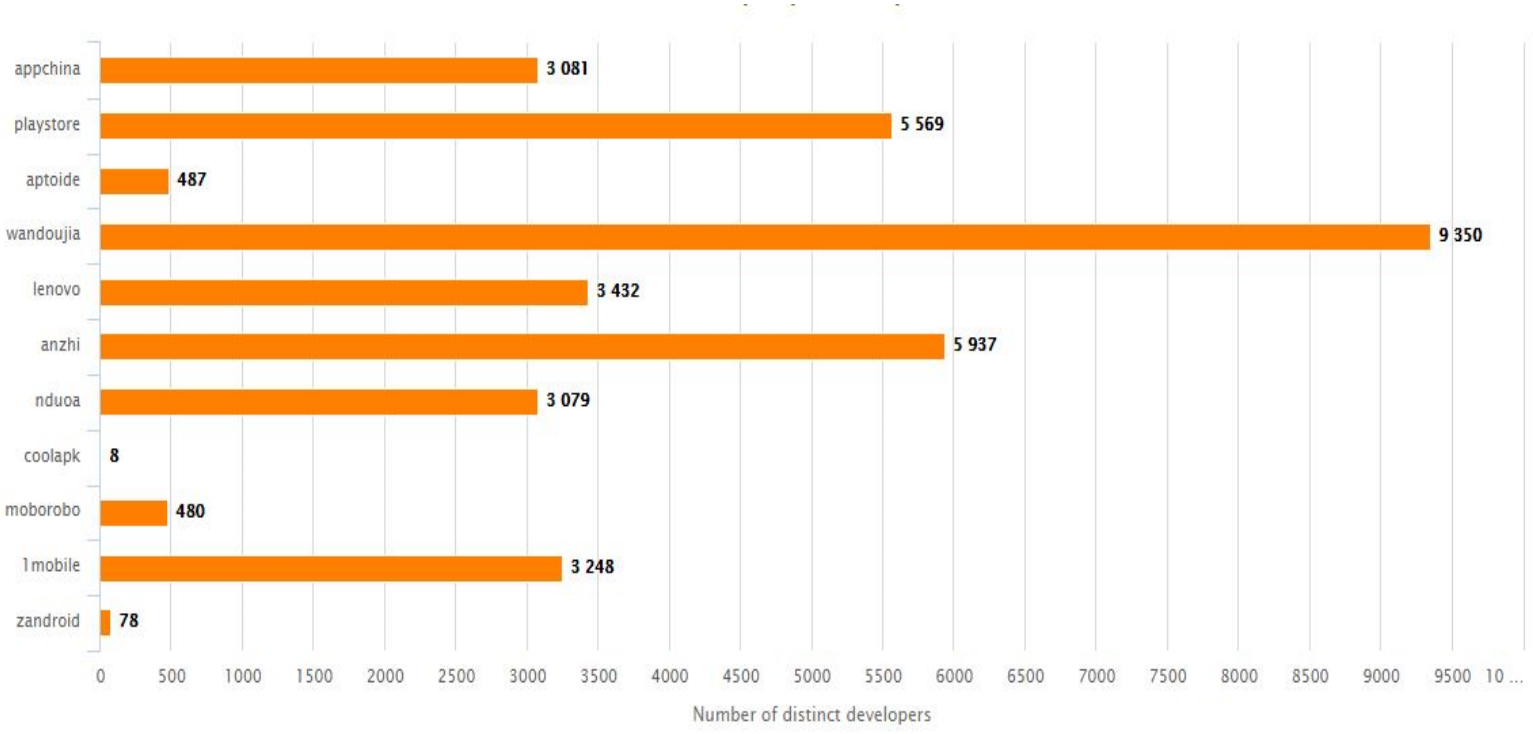

Figure 3.2: Malicious developers per marketplace.

3.1.2 The different forms of Android attacks

Attacks and malicious activities associated with Android mobile applications take different forms, and each malware takes advantages of specific Android component or mechanism to launch their attacks, whether it was the permissions, Intents, or functions within the source code itself. Fighting such wide range of malware is a tough battle for security researchers. What makes it even more difficult is how malware developers are employing highly sophisticated detection avoidance methods that render the state-of-the-art approaches ineffective.

The following section mentions examples of attack mechanisms, and the doors through which hackers are conducting their exploitations. Other malicious activities include but are not limited to: information leakage, kernel monitoring, sensitive data monitoring, and dynamic code loading, remote code execution, denial of service attacks, sensitive information & money stealing, authority bypassing, system crashing.

3.1.3 Taking advantage of Android components

For instance, when malicious activities are conducted through Android's Intents, they can be used maliciously to bypass authority and thus allowing the malware to get privileges of accessing and controlling sensitive information and functions and possibly stealing the users' money. And second, the denial of service caused by Intents, through the disabling of certain protection function and launching attacks that can cause the application to crash.

It was found that some particular intents are more frequently used in Android malware than benign apps. The same thing goes with requested and used permission, where investigations of analysts have gone into identifying the combination of permission, or the frequency of usage of any other Android components or services that can possibly indicate a malicious behavior.

3.1.4 Stealing of Sensitive Data with Adware

The collecting of sensitive information can be carried out through Adware as well. Mobile Adware behave in a tricky way in order to steal information. They don't show directly [30], and they wait for a number of days before launching for what they are designed for. In most of the cases, they look like a typical legitimate Android application, but in the background, they collect the following information: The model of the mobile device, the SDK version of the OS, availability of root access, and login credentials for the user's Google Play account.

The complexity and the multiple forms of malicious activities, the too many channels through which the malware can perform its work, and the tricky techniques that hackers follow to obfuscate their behavior have all contributed to the difficulty of detection with traditional approaches, and stat-of-the-art methodologies that don't consider adding the intelligence layer in their detection systems.

3.2 Traditional Detection Approaches

Traditional detection approaches -whether static or dynamic- follow one of the two styles: anomaly detection or misuse detection. A combination of both is also possible. But each approach has its limitations and weaknesses as well. The following section differentiate between the two detection approaches. A good understanding of the traditional state-of-the-art explains

better to the machine learning researcher the points in classical approaches that intelligence can help with.

3.2.1 Misuse Detection

The misuse detection is based on a list of specific signatures and features that the antivirus use to determine whether the encountered entity is malicious or benign. But the problem with the misuse detection is its inability to detect the malware that are not included in the list that was fed to it, namely the zero-day malwares. In addition to the significant overhead of the regular updating required for those lists. But no matter how many times the list was updated, it remain infeasible to design one that include each and every known malware.

3.2.2 Anomaly Detection

The anomaly detection on the other hand, does not work based on a specific list. It rather bases its malware identification by defining a normal behavior. Any deviation from this normal behavior would be detected as malicious action. With anomaly detection, there is no need to design exhaustive lists of signatures that needs to be periodically updated. By this, anomaly detection has the ability to detect zero-day attacks.

Nevertheless, the problem with anomaly detection is that it can detect a positive behavior that deviates from the priory defined normal behavior as negative, resulting in a high number of false alarms.

3.3 Issues of traditional approaches

The manual configuration of lists and behaviors fails to provide a generic protection. No matter how good the tool is, it can only provide protection to known and defined threats. [9]

The sophisticated malicious behaviors and detection-avoidance therefore require sophisticated security detection mechanisms that learn. Adding the artificial intelligence provided by machine learning will overcome the limitations addressed in the previous sections, resulting in more effective and reliable Android security apps.

3.4 Machine Learning with Static Analysis

Limited resources obstruct monitoring mobile applications at runtime, due to the significant overhead that comes along with it. And thus, static analysis approaches were advised. All static approaches are similar in that they work on offline features, which are the mobile application features that can be extracted without the need to run the application.

Among the static analysis methods being followed, each approach is distinguished by:

1- Initial inspirations
2- Which features they are focusing on.
3- The source of the extracted features.
4- The representation of the dataset.
5- The preprocessing practices to efficiently prepare the extracted data for the Machine-Learning model.
6- The classification/clustering approach.

3.4.1 The Initial Inspirations

The initial inspirations that lead to the learning-based approaches being adopted in Android security problems are many. Each researcher or analyst looks at the Android security problem from a different angle and thus, the exact same issue is being approached differently each time, which makes the diversity of the offered solutions a great advantage.

One example of which the ML helped in tackling the Android security problem indirectly is the SUSI framework [1], where researchers find a way to automate the process of preparing lists that antiviruses work on. Normally, a list with some classified API methods is given to the antivirus to help in detecting Android apps that use one of the classified suspicious methods. The problem with this approach is that it is infeasible to manually annotate and classify each and every method (more than 100,000 methods) in the Android API, and with every version of Android comes new functionalities and possibilities. That means that hackers can take advantage of some methods that go unclassified, and manipulate them for their harmful purposes.

Machine learning researchers here had a different way of looking into this problem, and they developed a model that can classify each and every method in the given Android API

version, so that antiviruses and detection software can rely on lists that are intelligently generated. So here, the contribution of the Machine-Learning was indirect.

On other researching attempts where Machine-Learning helped directly in solving the problem, they were almost all similar in how they follow the same steps: determining which Android application features to focus on (of dynamic or static nature), the best representations of those features, and then deciding of whether a supervised or unsupervised approach best suit the problem solving. The following sections illustrate more how different methods are distinguished.

3.4.2 Representations of the Datasets

A lightweight and robust representation of applications that enable determining typical indications of malicious activity is a crucial task. In addition, malicious activity is usually reflected in specific patterns and specific combinations of features.

1- Numeric vector spaces, or multidimensional vectors:
   At most learning approaches, numeric vectors are the option they operate well at. It is firstly needed to map the extracted features sets into vectors.
   In the DREBIN approach for instance, the extracted features were mapped to a vector space; because most learning methods operate on numerical vectors.
   Such that behavioral patterns are efficiently accessible to means of Machine-Learning.
2- Graph Theory: On other researches, the focus was on the structural composition of the underlying code. The use of the Graph Theory methods helps in structuring the Android source code into a better representation that enables the Machine-Learning models to work better. This method was inspired by two main observations: malicious functionality often concentrates on only a small number of its functions, and hackers often reuse existing codes to infect different applications. Graphs here help in measuring and quantifying the similarities in order to identify close candidates to a test sample. The steps that are included in this approach are:
   a. Extracting and labeling a call graph for each app sample. Some tools help in disassembling applications and extracting the call graph, e.g. Androguard framework. A typical function call graph contains a node for each function in the application code, while the edges represent the calls between functions.

b. A node is labeled according to its properties, or the type of instructions contained in the function that a node is representing. If two functions share properties, they will have the same label.
c. Now each node is labeled, and each function is characterized by the instructions it contains. Researches are also concerned about labeling the composition of functions, and thus, neighborhood of a function must be taken into account. For each node, a neighborhood hash is computed over all its direct neighbors in the function call graph – a procedure that is known as NHGK (Neighborhood Hash Graph Kernal)
d. Then comes the step of calculating the shared node labels in two graphs. Features are embedded in a space whose inner product is equivalent to the graph kernel.
e. And finally, the learning stage, where a linear SVM classifies a function call graph as malicious or benign. [9]

3.4.3 Features

The main target of feature-engineering is Features occur frequently in malicious apps but occur rarely in benign apps. A good engineering for the Android security detection problem requires an understanding of the domain-knowledge of Android, and awareness of the statistical approaches that are used to identify which features contribute the most to decide the final results, whether they were for classifications or clusters.

But before applying statistic, it is also important to identify –with Android experience- which features are employed to characterize the application better. At the same time, other researchers went into applying mathematical and statistical methodologies for the feature selection as information gain algorithm [14], PCA or Chi-square in order to eliminate the high computational overhead that high-dimensional datasets would bring. The focus is always on the most comprehensive features that depict the targeted behavior in the most efficient way. While in other approaches that adopt ensemble learners and deep learning, the orientation was more biased toward including almost each and every extracted feature, without the need to rank or analyze. [31][30]

Note that both a benign and a malicious app may require the same permissions and are thus indistinguishable via this permission-based mechanism. A point that is discussed more in

the fifth chapter, of whether the context is really represented in the dataset for the model to recognize it or not.

Most of the researches have represented the features in the datasets following a binary style: 0 means that the feature has not occurred, while 1 means that it does. A discrete representation of features can also be used. For example, when a specific feature is being used more than once, a value of 2 is assigned to it. [30]

Association-Rule mining [31] has a role to play here as well by presenting a way of a deeper delving into features, to exploit what better characterizes the malware. It helps in identifying the correlations among the features, producing 2-itemset association rules that shows better which features are most likely to be used by a malware than by a benign app.

If a malware cannot be identified correctly, its malicious characteristics must not have been properly learned by the machine learning model. This explains why some researchers went with a hybrid approach were both dynamic and static analysis were used to extract the data that depicts the Android application behavior.

3.4.4 Which Features Are Usually Excluded?

The eliminated features were those with no occurrences in either class. Out of the 130 permissions features, 5 had no occurrence in either class i.e.: ADD_VOICEMAIL, SET_POINTER_SPEED, USE_SIP, WRITE_PROFILE, WRITE_SOCIAL_STREAM.

Another point that should be taken care of is how some features are common in both the benign and the malicious Android app samples, and the existence of such features is not important and may possibly downgrade the overall quality of the classification model. Examples of this features: WRITE EXTERNAL STORAGE, a permission used to obtain privilege to write data on SD card.

In order to investigate the effect of feature diversity, three separate feature sets were created for training models and comparative analysis: (a) Feature set consisting of vectors from the 54 application attribute features only (b) Feature set of vectors from the 125 permissions only (c) Feature set consisting of vectors from a mix of all the (diverse) 179 property vectors. So the

point is, trying different ways of depicting the behavior and characteristic of Android app may help in deciding which way (of which features) should have the most attention. [27]

3.4.5 Algorithms and Models

Some classification problems are linearly separable, and others are not. The following section mentions the most popular Machine-Learning algorithms that were used in the Android security problems, along with justifications of their usage. Of course there is no one right way or one correct model. Nevertheless, results can differ depending on the choice of the classifier.

3.4.5.1 Bayesian Models

The use of probabilistic learners like Naïve Bayes algorithms consider the nature of the detection problem we have at hand. At some situations, it performs well with high accuracy, but in other security detection problems of different nature, the classification with probabilistic models was not helpful.

Naive Bayes produces very imprecise results when the detection classification-problem is the rule-based kind, or has almost fixed semantic with a variance that is not large enough to justify the imprecision introduced by probabilistic approaches which are rather susceptible to outliers. Same as in the SUSI research [1]

While in a different research, were the concept of probabilistic learners were very efficient in quantifying uncertainty, as it was viewed as the only method that take this ignored certainty into consideration and its calculations when performing the prediction tasks.

The result of the prediction is always ‘benign’ or ‘malicious’ with no level of uncertainty associated with it, while the researchers have argued that uncertainty is an important aspect of any prediction.

Their justification was how the ignoring of this uncertainty leads to incorrect classification of both benign and malicious apps. A utilized Bayesian machine learning that preserve the concept the uncertainty in all its calculation was helpful in many ways: in identifying the effect that different features have on the classification of the Android app, reducing incorrect decisions, and resulting in an improved accuracy. [32]

3.4.5.2 Support Vector Machines.

Support Vector Machine is the most frequent model of choice with Android security detection problems across the papers that were covered for this literature survey It was described as being concrete and precise, especially when algorithms as SMO (Sequential Minimal Optimization) is implemented to train an SVM.

The basic principle of an SVM is to represent training examples of two using vectors in a vector space. The algorithm then tries to find a hyper-plane separating the examples. For a new, previously unseen test example, to determine its estimated class, it checks on which side of the hyper-plane it belongs. In general, problems can be transformed into higher-dimensional spaces if the data is not linearly separable, but this did not prove necessary for any one of our classification problems.

SMO is only capable of separating two classes. However, when three classes or more are presented in the Android security detection problem, a one-against-all solution can be applied. In the one-against-all SVM classification, every class is tested against all the remaining classes together to find out whether the instance corresponds to the current single class or whether the classification must proceed recursively to decide between the remaining classes.[1]

3.4.5.3 Deep Learning.

Deep learning methods outperformed the classical machine learning that operate with a shallow architectures of less than three computational units. And they proved to be effective especially with the availability of more training data. In addition, deep learning techniques get better the more data you feed to it.

In Droiddetector proposed research [30] deep learning was compared with the traditional machine learning approaches, and the architecture was built with Deep Believe Networks (DBN), but other alternatives as Convolutional Neural Networks (CNN) can be used as well.

To characterize the Android apps with the designed deep learning architecture, 192 features that are collected from both dynamic and static analysis were used, and were representing following a binary style instead of discrete.

Association-rule mining is usually used with deep learning to identify the features of interest. Furthermore, it was suggested that a dataset of a malware/benign apps ratio of 1:1 leads to a better accuracy than other ratios as 1:5 or 1:20.

3.4.5.4 Ensemble Classifiers

The application of Random Forest, an ensemble learning method, is proposed to learn the app characteristics based on features from both source code and manifest xml file. Ensemble learning builds a prediction model by combining the strengths of a collection of simpler base models [33]. Random Forest for instance, uses Bagging (bootstrap aggregation) to produce a diverse ensemble of Random Trees independently trained on distinct bootstrap samples obtained by random sampling.

For security detection problems, ensemble classifiers are utilized in order to enable zero-day Android malware detection with high accuracy. A large feature space approach is developed, employing hundreds of features for classification decisions. The diversity and extent of the features employed is actually advantageous to ensemble learning as it provides greater degree of freedom to introduce randomness in feature selection.

Diverse categories of features are included in the case of ensemble learner: API calls, commands and permissions, thus inherently resilient to obfuscation within app code as majority of the features will still be accurately extractable through the Manifest XML file.

With this approach there is no requirement for feature selection step to eliminate less relevant features, the way we are experimenting with ranking and feature selection in chapter 4.

3.4.5.5 Parallel Machine Learning

Other researchers went into applying Machine Learning following a parallel style of running and execution. They observed how such an approach, if combined with a comprehensive list of feature, would produces good detection results.

The limitations that inspired such a combination (multiple learners in parallel joined with many features) were not the traditional approaches limitations alone, but also how Machine Learning itself is applied (or classical ML).

First, the traditional Machine Learning that was applied in Android security problems is usually based on one or two algorithms. Utilizing a single classifier can't cope with the wide variety of Android malware, as the researchers of this work [14] declare. And they think that by specifying one single algorithm, it is difficult to prove which one was most suitable to the Android malware.

Second, running the multiple models like in ensemble models as Random Forests for example consumes time and computational resources, as they require multiple iterations. Thus, in parallel Machine Learning and information fusion approaches, those resources are saved.

Another example of traditional Machine Learning approach that was resource-consuming is Drebin, which spent too much time to build the classifier because of the high-dimensional feature vector.

Third, the features used is a challenging part. Researchers are trapped between using too much features that depict the Android app behavior better-but which would consume more resources, and using less features to save running time execution consumption, but resulting in less accurate findings.

The use of parallel Machine Learning and information fusion enabled experimenting with various categories of features, at the same time saving resources and speeding up the classification process.

Another important aspect that should be considered if parallel Machine Learning was applied is the method by which diverse classification outputs are combined. In Mlifdect example of parallel ML, the outputs of the different models were in this case probabilities instead of classifications. Each probability value is combined with local credibility, which is localized as confidence measure of a classifier, a Dempster-Shafer theory and probability analysis were used to integrate results.