IoT

MDPI

Article

A Client/Server Malware Detection Model Based on Machine Learning for Android Devices

Arthur Fournier, Franjieh El Khoury * and Samuel Pierre

Mobile Computing and Networking Research Laboratory (LARIM), Department of Computer and Software Engineering, Polytechnique Montreal, Montreal, QC H3T 1J4, Canada; arthur.fournier@polymtl.ca (A.F.); samuel.pierre@polymtl.ca (S.P.)

* Correspondence: franjieh.el-khoury@polymtl.ca

Abstract: The rapid adoption of Android devices comes with the growing prevalence of mobile malware, which leads to serious threats to mobile phone security and attacks private information on mobile devices. In this paper, we designed and implemented a model for malware detection on Android devices to protect private and financial information, for the mobile applications of the ATISCOM project. This model is based on client/server architecture, to reduce the heavy computations on a mobile device by sending data from the mobile device to the server for remote processing (i.e., offloading) of the predictions. We then gradually optimized our proposed model for better classification of the newly installed applications on Android devices. We at first adopted Naive Bayes to build the model with 92.4486% accuracy, then the classification method that gave the best accuracy of 93.85% for stochastic gradient descent (SGD) with binary class (i.e., malware and benign), and finally the regression method with numerical values ranging from −100 to 100 to manage the uncertainty predictions. Therefore, our proposed model with random forest regression gives a good accuracy in terms of performance, with a good correlation coefficient, minimum computation time and the smallest number of errors for malware detection.

Keywords: Android devices; mobile malware; mobile applications; malware detection; client/server architecture; offloading; prediction; classification; regression

Citation: Fournier, A.; El Khoury, F.; Pierre, S. A Client/Server Malware Detection Model Based on Machine Learning for Android Devices. *IoT* **2021**, *2*, 355–374. https://doi.org/10.3390/iot2030019

Academic Editors: Uday Tupakula and Hyun-Ho Choi

Received: 27 May 2021
Accepted: 23 June 2021
Published: 24 June 2021

1. Introduction

Nowadays, mobile devices are becoming an essential part of our daily life and are used even more than conventional computer systems such as personal computers. The information manipulated by smartphones is diverse and is very often private or even confidential. Therefore, smartphones are used as work tools, means of payment, or simply as means of communication. However, they are often more vulnerable than traditional computer systems due to the usage of all kinds of networks and protocols, such as wi-fi and mobile networks (e.g., third or fourth generation, etc.).

For the last few years, the rapid evolution of the usage of Android has made malware attacks on mobile devices an important challenge in research. Moreover, the existence of Android malware is increasing rapidly. This malware can access the information of users in the background without the users on Android devices being aware, send a short message service (SMS) to the username, and violate the privacy of users [1].

This growth was accompanied by several methods of analysis and malware detection, such as static analysis, dynamic analysis, or hybrid analysis, to keep Android devices secure from malware. Therefore, accurate malware detection analysis methods require a large number of hardware resources, which are important resource-constrained mobile devices.

Most of the defense mechanisms in Android Play Protect were already integrated before version 8 but are now visible to users. Moreover, the Android framework is based on Linux, and each application runs in a restricted environment due to the Android virtual

machine, formerly Dalvik, and now known as Android Runtime (ART) [2]. This latter guarantees a basic level of security.

Applications can be installed from the Google Play Store, which already integrates a malware scanner called Bouncer, based only on dynamic analysis, and is unfortunately imperfect [3]. Bouncer aims to check for malicious apps and known malware, but it is possible to install applications from unknown sources. A common practice is for malware authors to repackage known applications (often paid) with malware features before redistributing them for free to naive users through unofficial free stores, such as Aptoide [4]. This will highlight the limitations of Bouncer.

To overcome the shortcomings of Bouncer and Play Protect, as well as to protect themselves from applications of unknown sources, several vendors of antivirus for Android smartphones are distributed, as free or premium versions, for individuals (e.g., Lookout) or businesses (e.g., MI:RIAM by Wandera, z9Engine by Zimperium, Skycure by Symantec) [5–8]. However, these antiviruses do not always indicate the method used. Therefore, more complex methods require a modification of the Android framework or even the phone hardware, as for the CogSystems HTC D4 [9], and are therefore very uncommon.

In the context of mobile commerce (i.e., m-commerce), the previously introduced security issues of mobile devices are all the more important. Devices, applications, networks and servers will be required to manipulate privacy information and financial currencies, whether virtual or not. Popular mobile payment applications, such as Apple Pay [10], have many vulnerabilities that delay their adoption by the general public and put their users at risk. The threat of malware and malware software leads to a serious issue in mobile commerce.

Symantec reported over 30 million attack attempts on mobile devices in the market, which leads to a low level of protection that users afford their smartphones [1,11]. The number of malware programs continues to grow and target mobile platforms, such as The-ZeuS-in-the-Mobile (Zitmo) which damages the user's data on Android devices [12]. Malware activities on mobile devices include call and text-message recording, geolocation, and private data transmission [13]. Recently, high-profile attacks (e.g., SolarWinds, Microsoft Exchange Server ProxyLogon attacks, the vulnerabilities found in Pulse Secure VPN, etc.) help attackers avoid the hurdle of needing to overcome multi-factor authentication [11].

Some of the most promising methods rely on the use of cloud computing to analyze data [14], and on monitoring the communications carried out by the device [15]. This represents a multitude of approaches to be explored, in order to define the malware detection method in mobile environments that best addresses our problem.

In our previous publication [16], we presented our proposed model for malware detection on Android devices based on client/server architecture to reduce the heavy computation of data on the mobile device and doing the processing remotely on the server, but we focused on the mobile device part by directly performing the tests offline on the device. We gradually optimized the classification method of malware detection from Naive Bayes to binary classification, then to regression algorithms for better classification of the newly installed applications on Android devices. The random forest regression algorithm, with numerical values ranging from −100 (benign) to 100 (malware), gives good results for malware detection.

The main contributions of this paper are as follows:

(1) Describe the client/server architecture and the remote processing (offloading) on the server for prediction of the newly installed applications to reduce the computation time on the mobile device, and by using numerical values for classification (i.e., −100 for benign and 100 for malware) to manage the uncertainty predictions;
(2) Implement our proposed model for malware detection to validate our proposed methodology;
(3) Detail the Naive Bayes method and present the results of more classification and regression algorithms from the Waikato environment for knowledge analysis (Weka)

version 3.8.2 [17] to emphasize our choice of the most accurate classifier (i.e., the random forest algorithm) for prediction in terms of performance with a good correlation coefficient, a minimum computation time and the smallest number of errors for malware detection.

(4) Add more graphics and tables to clearly visualize our choice of the best regression algorithm, with high coefficient correlation and low computation time.

The rest of the paper is organized as follows. In Section 2, we present a brief background in malware detection and an overview of the existing methods for malware detection, then at the end, we perform a comparative study of these methods to highlight the strengths and the shortcomings of each. We describe the proposed model for malware detection on Android mobile devices in Section 3. In Section 4, we present the implementation of the proposed model, and we illustrate the different functionalities of our implemented application entitled "ATISCOM Malware Detection". We detail the different stages of the optimization of our proposed classification methodology for malware detection in Section 5. In Section 6, we discuss the evaluation performance of the proposed classification methodology. Finally, we provide a conclusion in Section 7.

2. Background and Related Work

In this section, we first introduce a brief background of malware detection; secondly, we present a general overview of the available malware detection methods; and thirdly, we highlight the strengths and the shortcomings of these methods.

2.1. Background

Hereafter we define the different terminologies and concepts related to Android apps, machine learning and evaluation metrics that we will use within this paper [1,10–13,18–24].

Malware is an Android package kit (APK), also known as an Android app, used to serve illegal purposes, such as espionage or extortion. Therefore, malware detection is a classification problem, which consists of determining the class of an application. The different classes presented in this paper are of two types: (1) malware; and (2) benign. An application is benign if it is legitimate and harmless.

There are three analysis methods for malware detection: (1) a static analysis method that is based on different data structures, represents the code in various ways, and has a variable sensitivity of analysis; (2) a dynamic analysis method that is separated according to the level of inspection (i.e., application, kernel, virtual machine) and the generation of application entries; and (3) a hybrid analysis method that combines static and dynamic analysis to benefit from both types of analysis, with the gaps of one minimized by the forces of the other.

Machine-learning technology consists of many different methods and objectives. The common denominator of these methods is the input of a very large amount of data, labeled for supervised learning, or unlabeled for unsupervised learning in a learning algorithm. In our case, the purpose of a learning algorithm is to take some known features of many applications identified as malware or not, then to determine whether new samples are malware or not based on the rules recently established by this algorithm.

Permissions are considered as features. We think about the presence or not of each of the 151 permissions of the Android system in the "Manifest" file of a specific application, which gives a Boolean vector, and the last feature defining whether the application is malware or a benign application. The latter can be a class that can only take the value of malware or benign, or a number that defines the probability of its being malware. In the first case, we face a classification problem, and in the second, a regression problem.

The classification problem is considered as a problem whose challenge is to manage only the certainty predictions and to determine the class of a given sample as malware or benign. However, we address in this paper the regression problem. Therefore, we associate numerical probabilistic values to manage the uncertainty predictions: (1) -100 if the algorithm is certain that the application is benign; (2) 100 if it is certain that the

application is malware; (3) 0 if the model is totally unable to recognize the class of the sample; and (4) any other value that gives an indication of the certainty of the model, when it applies the rules that it has previously built on this new sample. Note that many of these regression algorithms require heavier computations, and potentially require offloading if they are run on a mobile.

An offloading is referred to whenever calculations are uploaded to the server from the client to be handled in a performed and quick way, because of the limitation of Android phone processing. Moreover, the efficiency of this operation depends on the quality of the network linking the client and server. In this research work, we will focus on reducing the processing time.

Performance is distinguished from accuracy. It will be used to describe the speed of execution of the algorithms, taking into account the storage space and the capacity of the random access memory (RAM) used by the application. The performance can be expressed during the training phase of the algorithm, especially during the classification or prediction of a new sample, since this operation will generally be performed in real-time on the user device.

Evaluation metrics are quantifiable measures that determine if the detection model efficiently differentiates malware from benign applications. Among these metrics, let us quote:

(1) The accuracy, which designates in our case the ability of the system to correctly predict whether a new application is malware or benign. The accuracy criterion will be considered to evaluate a classification algorithm that indicates the percentage of correctly classified applications, and the correlation coefficient in the case of a regression algorithm to show the percentage of correlation between predicted and actual values. We will obviously seek to maximize accuracy in priority;
(2) The precision is the proportion of correct positive predictions. A detection model producing no false positives has an accuracy of 1;
(3) The recall is the proportion of actual positive results that have been correctly identified. In addition, the recall is called the true positive rate (TPR). A detection model producing no false negatives has a recall of 1;
(4) A false positive rate (FPR) represents the total percentage of wrongly classified applications.

2.2. Malware Detection Methods

The detection of malware is becoming an important challenge for researchers. In this section, we present a literature review based on three categories of malware detection analysis methods for applications installed on Android devices: (1) the static analysis method; (2) the dynamic analysis method; and (3) the hybrid analysis method. Different solutions have been developed to ensure the accuracy of detection of malware, and by taking into account the context (e.g., real device, emulator, etc.) in which the malware detection process takes place.

MADAM [21] presents itself as one of the most advanced recent solutions for real-time mobile malware detection by dynamic analysis and machine learning. This framework uses self-learning to recognize malware. It classifies them based on different malicious behaviors observed at different levels of Android: kernel, application, user, and package. In this paper, 2800 malware from 125 different families, from three databases, were tested on a real device, giving a detection rate of 96.9%. In addition, 9804 legitimate applications were tested to show that the false positive rate (FPR) is very low: 0.2%. This solution is not intended for the general public but seeks to prove the strength of such a dynamic multi-level approach on the device, and eventually encourage smartphone system builders to integrate MADAM into their OS. Therefore, this is the most important gap, apart from meeting all the requirements of such a system, and is perfectly in line with our work.

CrowDroid [22] presents a relatively simple detection method. However, its great innovation is to propose a client/server architecture for crowdsourcing the information retrieved from each end-user's phone to create the online database.

T2Droid [25] is a dynamic analysis method for malware detection, with an excellent detection rate ranging from 98% to 99% and an FPR of 2% tested on 160 applications. It runs partially in the secure zone of the ARM TrustZone, which is included in several latest-generation phones as a trusted execution system, in order to start the system and then the applications. Therefore, the system is secure, unlike conventional antivirus software, and requires full control over the hardware and access to the secure area to deploy the software. It is difficult or even impossible to include such a system in a lambda device.

IntelliAV [26] is an on-device malware detection system that uses static analysis coupled with machine learning. The application is already available on Google Play Store, allowing for possible accuracy comparisons. Based on a training and validation set of 19,722 VirusTotal applications [27], including 9664 malware ones, the authors obtained a true positive rate (TPR) of 92.5% and an FPR of 4.2%, with only 1000 attributes generated by the training process. This is much smaller than the static analysis methods based on machine learning off-device. Moreover, the authors evaluated their model on a set of 2898 benign applications and 2311 malware from VirusTotal, dated from February 2017. The accuracy is 71.96%.

Aonzo et al. [28] developed an application available on the Google Play Store called BAdDroIds. They proposed a static analysis method using on-device machine learning that is supposedly efficient. This method has an accuracy of 98.9% and an FPR of 0.6%. When installing a new application, BAdDroIds extracts the permissions declared in the Manifest file and calls to the AAPI (Android application program interface) from the DEX code (i.e., the bytecode that runs in the Android virtual machine). To reduce false results, when the classification algorithm has less than 70% confidence, the application lets the user choose whether to classify the installed program as malware or not. The model was built in the best experiment from 12,000 samples, outside the device and before experimentation with the 1000 applications installed on the LG Nexus 5 test device. The method of feature extraction does not change, but the analysis of an Android package (APK) takes an average of 64.474 s and the stored file weighs 5539 kibibytes (Kib).

ADroid offers lightweight monitoring and dynamic analysis methods directly on the device [29]. ADroid does not require root access but needs numerous permissions to monitor the Android application space. ADroid's assumptions are not valid in the case of a compromised device, and it does not detect privilege escalations. ADroid has on the one hand a blacklist of malware signatures in the manner of traditional antivirus, and on the other hand a user whitelist. For applications that do not belong to either list, behaviors are observed and recorded to create a behavioral vector, which will be compared to normal behavior. The testing results on 720 applications give an FPR of 5.9% and a detection rate of 97%. The processing takes 0.53s every 5s to generate the vector. However, the operation of ADroid gives some interesting ideas and can be improved: clustering the vector can be used to reduce its size and false positives, and adding an off-device detection as a complement could improve performance when the network allows it.

Monet [30] implements a client/server architecture using signatures representing the behavior of the running application. The signature is generated on the device and then sent to the server for comparison with the database. The client application intercepts the calls using hooks and kernel modules, which can be problematic to implement on any device. Root exploits appear to be detectable through suspicious system calls. This method does not seem to be able to detect completely new malware, but only variants of existing malware. The article presents very good performances, with a detection rate of 99% and an FPR of 0% tested on 4223 applications. However, we should note the limitations of the experiments performed only on the detection of DroidKungfu variants with legitimate applications in the set, and on original malware manually transformed into variants for the performance of the client application.

SAMADroid [31] is a hybrid method with three levels. It combines static and dynamic analysis, an on-device client and a remote server, and machine learning. Drebin's dataset [18] is used to train the algorithm, which contains 5560 malware from 179 different families, as well as a selection of 123,453 benign applications extracted from different stores. SAMADroid's accuracy is remarkable. The static analysis has a detection rate of 99.07% for an FPR of 0.03%, and the dynamic analysis has a detection rate of 82.76% for an FPR of 0.1%. Nevertheless, it should be noted that the static analysis can only be performed via the remote server. The performances are therefore not guaranteed without a stable internet connection. Finally, the used dataset contains many legitimate applications, but sometimes less malware than other newer datasets.

ServiceMonitor [32] is a method for dynamic analysis on a real device. The dataset used contains 4034 malware taken from Drebin's dataset and 10,370 legitimate applications from the Android store. Experiments with this database show a detection rate of 96% for a true false positive (TFP) of 4.4%. On the device, the overheads are, for the CPU, 0.8%, and for memory, 2%. However, ServiceMonitor requires root access to implement analysis on the phone. Its algorithm is trained on an emulator, then tested on a physical device.

Wang et al. [33] proposed a new method of malware detection based on two observations: usage and application anomalies. This method claims to detect new malware using zero-day (i.e., vulnerabilities unknown to date). This method works through a sandbox included in a framework called CuckooDroid, and not on a real device. For use on a real device, the phones will call the cloud computing service containing the emulator and retrieve the results afterward. It allows identifying features by static and dynamic analysis of the tested application. They used a one-class support vector machine (SVM) for classification. If the analyzed application deviates from the categorized behavior (benign applications) by more than the threshold, it is classified as new malware using a new vulnerability (zero-day). Indeed, the obtained accuracy is very good, with a detection rate as high as 98.76% for an FPR of only 2.24%, and a false negative rate (FNR) of 1.24%. This method is validated on a set of 5560 malware from Drebin's dataset and 12,000 benign applications from Chinese application markets, validated by VirusTotal.

De C. Souza et al. [34] developed a hybrid expert system in the identification of malware using the artificial intelligence and fuzzy system approach, as well as the linearly scaled hyperbolic tangent (LiSHT) activation function to increase the accuracy (97.82%) of fuzzy neural network outputs. The proposed model shows a significant improvement in runtime (27.28s for training and 0.01s for testing) compared to other hybrid models.

Almshari et al. [35] adopted a method of descriptive analysis of the power consumption data of the machines and their clusters, connected on a smart grid, to detect if they are infected without violating the privacy of the users. The authors used two-way analysis of variance (ANOVA) and autoregressive integrated moving average (ARIMA) methods. This approach proves that there is a correlation between electric power and which software application is running, and it is possible to create power consumption profiles for various software applications including normal and abnormal behavior like a virus. The results validate the good detection of what type of application is running, and if an individual machine or its cluster is infected. This method is considered a good management tool for administrators and could serve as a future extension of our proposed model in terms of monitoring the behavior of the running application.

2.3. Comparision of the Existing Malware Detection Methods

Table 1 represents a comparison study of the existing malware detection methods detailed in the previous section to highlight the strengths and the shortcomings of each, as well as to justify our choice of the malware detection method on Android devices [25,26,28–33].

Table 1 shows that most of the methods have a good, or even very good detection rate: the most vulnerable method already achieves 92.5%, and some methods are close to 100%.

Table 1. Comparison of the malware detection methods.

Method	Type	Total Number of Tested Apps	Detection Rate (%)	False Positive Rate (%)	Overhead
MADAM	Dynamic on device	12,604	96.9	0.2	1.4%
T2Droid	Dynamic on device	160	98 to 99	2	N/A [1]
IntelliAV	Static on device	19,722	92.5	4.2	4–16s
BAdDroIds	Static on device	14,988	98.9	0.6	64,67s 5539KiB
ADroid	Dynamic on device	720	97	5.9	N/A [1]
Monet	Static client/server	4223	99	0	7%
SAMADroid	Hybrid client/server	129,013	Static: 99.07 Dynamic: 82.76	Static: 0.03 Dynamic: 0.1	0.6%
ServiceMonitor	Dynamic on device	14,404	96	4.4	0.8-2%
CuckooDroid	Hybrid on emulator	17,560	98.76	2.24	N/A [1]

[1] Not provided in the literature.

It is important to point out the high false-positive rate (e.g., 5.9% for IntelliAV), which can quickly invalidate the method whatever the announced detection rate (97%) of accuracy. In addition, it is crucial to highlight in the case of BAdDroIds the high capacity of the storage (5539 KiB), and the enormous process time for malware detection (64.67 s) that invalidate the method despite its high detection rate (98.9%). Ideally, we will try to reach a detection rate of 100% with a false positive rate of 0% by taking into account the performance and the management of constraints. However, in reality, it will often be a compromise between these two values. The distinction in this fuzzy area for the method that requires this compromise can however be left to the user. In other words, the user will have the choice to classify the installed program as malware or not.

The 19,722 applications analyzed by IntelliAV have more validation weight than the 720 of ADroid. Therefore, we intend to maximize the number of applications tested by our system.

As for the context where the process of the existing method is performed, we can see that Monet and SAMADroid have a good detection rate close to 100% for a false positive rate that is close to 0% in client/server architecture, regardless of the quality of internet connection. This will be a good choice for our solution.

Moreover, Alzaylaee et al. [36] offered a detailed study of the differences between the execution environments on an emulator versus a real phone. This study recommended performing the detection on a real device, rather than an emulator such as with CuckooDroid.

3. Proposed Model

Our proposed model is based on a static analysis method for malware detection on Android devices. This model comprises four modules: (1) collection of permissions; (2) dataset collection; (3) training of the dataset; and (4) prediction of the newly installed applications based on client/server architecture.

3.1. Collection of Permissions

We collected 151 permissions of the Android system from the official site of Android development, which are categorized into three security levels [37,38]: (1) normal permissions to protect the API calls that could annoy but not harm the user and do not need user approval (e.g., SET_WALLPAPER, VIBRATE, WAKE_LOCK, READ_SYNC_SETTINGS, etc.); (2) dangerous permissions that let an application performs harmful actions (e.g., CLEAR_APP_CACHE, RECORD_AUDIO, SEND-SMS, RECEIVE_SMS, BIND_VISUAL_VOICEMAIL_SERVICE, READ_PHONE_NUMBERS, READ_SMS, READ_CONTACTS, etc.); and (3) signature permissions that regulate access to extremely dangerous privileges (e.g., CLEAR_APP_USER_DATA, BIND_NOTIFICATION_LISTENER_SERVICE, REQUEST_INSTALL_PACKAGES, READ_VOICEMAIL, BIND_DEVICE_ADMIN, etc.). Therefore, malware with dangerous or signature permissions can spy on users, delete data, and abuse their billing accounts.

These permissions will be declared in the "Manifest" file, in order to provide an overview of which parts of the system the analyzed application can access. For example, many malware applications ask for permission to send SMS messages, while few benign applications do and are generally identifiable. Moreover, Android uses INSTALL_TIME permissions where their requirements (e.g., location, network communication, personal information, storage, hardware controls and system tools) are declared in the "Manifest" file and need user approval during the installation of an application from the Android market.

To extract those permissions used for our dataset, we ran the multi-platform command line (aapt dump permissions <app.apk>) based on the Android asset packaging tool (aapt). This tool is composed of the application dump permissions that take a package as an argument, and lists the permissions given by its "Manifest" file.

We then compared these permissions with the list of official permissions of the Android system to obtain a Boolean vector indicating 1 (i.e., the presence of permissions) or 0 (i.e., absence of permissions). Furthermore, we indicated whether the application was malware or benign, as shown in Figure 1. On the phone itself, the PackageManager class allows finding the "Manifest" file and the permissions associated with the installed applications or readable archives on the file system.

```
@DATA
1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, benign
```

```
@DATA
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, malware
```

Figure 1. Boolean vectors with binary classes (i.e., benign and malware).

3.2. Dataset Collection

We collected applications from two sources that contain the most recent applications: theZoo [39] and contagio [40]. Then, we gathered 5560 malware applications from 179 different families using Drebin's dataset [18].

However, these databases do not provide benign applications to build a balanced database, in order to have as many benign applications as malware applications. Therefore, we download about 27,965 applications from the Google Play Store [41], taking the top 500 in each category with a high detection rate and a favorable public rating.

Furthermore, we conducted a double-check with concurrent antivirus programs, such as the VirusTotal database [27], to prepare our dataset for training and have good precision in our result. We then verify that each collected application is indeed a valid APK or Java Archive (JAR) package whose permissions can be determined via the "Manifest" file.

3.3. Training of the Dataset

For the training, we constructed two sets. The first was a test set consisting of the latest TheZoo and contagio malware, which make up a total of 251 valid malware, and the same number of benign applications randomly selected from the 27,965 downloaded. The second is the training set, consisting of 5304 of Drebin's malware and an equal number of benign Google Play Store's applications, randomly selected from the remaining 27,714.

We used at first the Naive Bayes algorithm included in Weka 3.8.2 [17] software to build our training model, as shown in Figure 2. This software is written in Java and has, in addition, a graphical interface and a command-line API to use from a Java program.

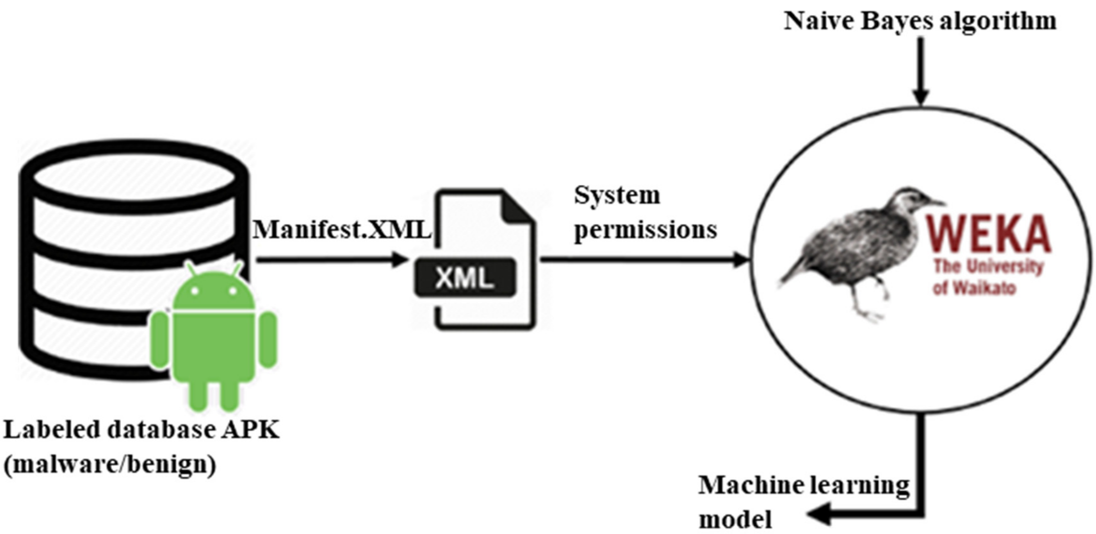


Figure 2. Training model.

Therefore, it is possible to remove the code from the graphical interface of the program and include the Java archive thus generated in an Android project, to make predictions from the model.

We built the training model on the server. For this reason, we wrote several batch scripts to extract the permissions of our 32,000 applications, compared them to the system permissions to get the Boolean vector composed of 151 attributes of values 0 or 1, and added at the end our knowledge of their classification (i.e., malware or benign), as shown in Figure 1. All this is written in an ARFF type file, which corresponds to a data file exploitable by Weka 3.8.2.

At a second stage, we optimized our training model by replacing the benign value with −100 and the malware value with 100, to take into account the uncertainty predictions, as shown in Figure 3. We used the random forest regression algorithm integrated into Weka 3.8.2, which gives a good accuracy for prediction in terms of performance with a good correlation coefficient, a minimum computation time and the smallest number of errors for malware detection.

```
@DATA
1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, -100
@DATA
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1
, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 100
```

Figure 3. Boolean vectors with numerical values (i.e., −100 for benign and 100 for malware).

3.4. Prediction of the New Installed Applications Based on Client/Server Architecture

Our prediction model is based on a client/server architecture, as shown in Figure 4. This architecture is composed of three components: (1) mobile device; (2) server; and (3) client/server communication.

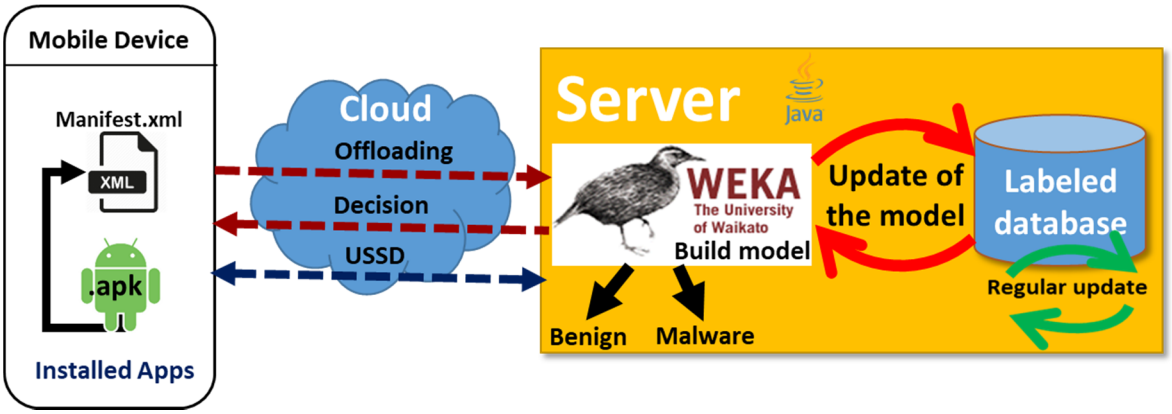


Figure 4. Proposed client/server architecture for malware detection on Android devices.

In this architecture, we present the six steps to perform the prediction: (1) labeling on the server by assigning a class to each application (i.e., benign or malware); (2) building the model by the training set on the server; (3) acquiring a new application by the client; (4) the client sends the analysis request to the server for processing remotely (i.e., offloading); (5) performing the prediction on the server by using the static analysis method and the machine learning algorithms for classification; and (6) sending the prediction result to the client.

3.4.1. Mobile Device

This represents the client's side. It has a hybrid operation to detect malware with varying degrees of efficiency, depending on whether it is offline, on mobile networks or online, to the best of its capacities. We focused on methods for offline mobile devices that will not require a lot of resources. However, we adopted the assumption that all computations and storage could be handled on the device itself and sending only instructions that slow down the execution to the server for processing remotely (i.e., offloading) and prediction.

3.4.2. Server

The server handles the remote processing (i.e., offloading) of the prediction using the pre-trained model to determine the result of the prediction, ranging from -100 to 100 and return it to the client (i.e., mobile device), in order to decide whether it is malware (100) or benign (-100), as justified in the next section.

3.4.3. Client/Server Communication

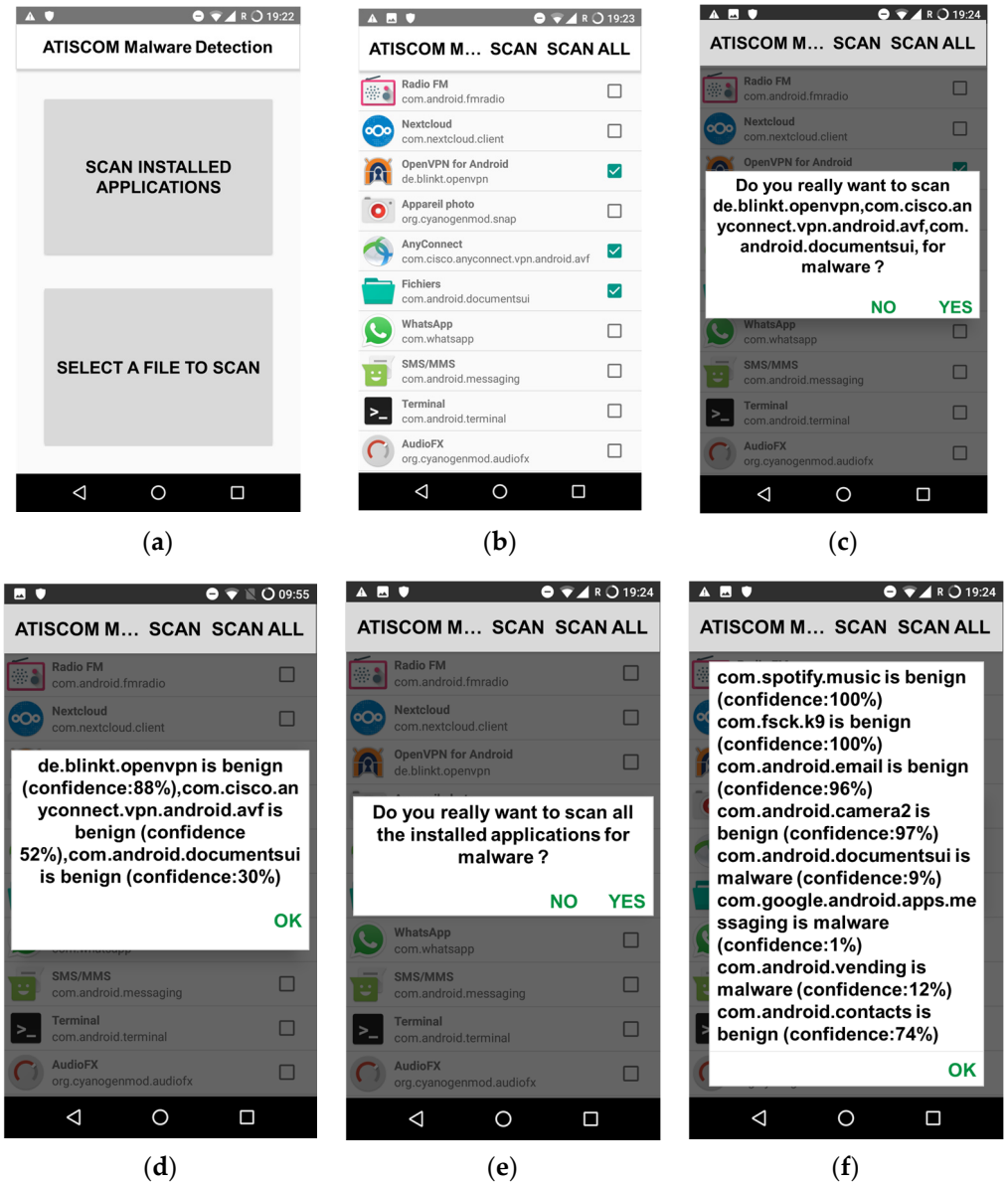
The communication between the client (i.e., mobile device) and the server is established by using USSD (unstructured supplementary services data). USSD is a user-interactive, menu-driven, cheaper, faster solution and is better than SMS regarding cost, security and channel usage [42]. It provided accuracy and good performance in our detection environment.

4. Implementation of the Proposed Model

In this section, we present the implementation of our proposed model for malware detection based on the static analysis method and the random forest regression algorithm for prediction on client/server architecture.

We used: (1) Android Studio and Java for development; (2) various types of Android devices (i.e., MotoG LTE (first generation) and Nexus 5 API 28) for the test; (3) a Linux server for remote processing (offloading) of the prediction; and (4) USSD technology for communication between the client (i.e., mobile device) and the server, by sending a remote request from the client to the server for remote processing of the prediction when a new application is installed and sending the prediction result (i.e., −100 for benign and 100 for malware) from the server to the client for decision-making.

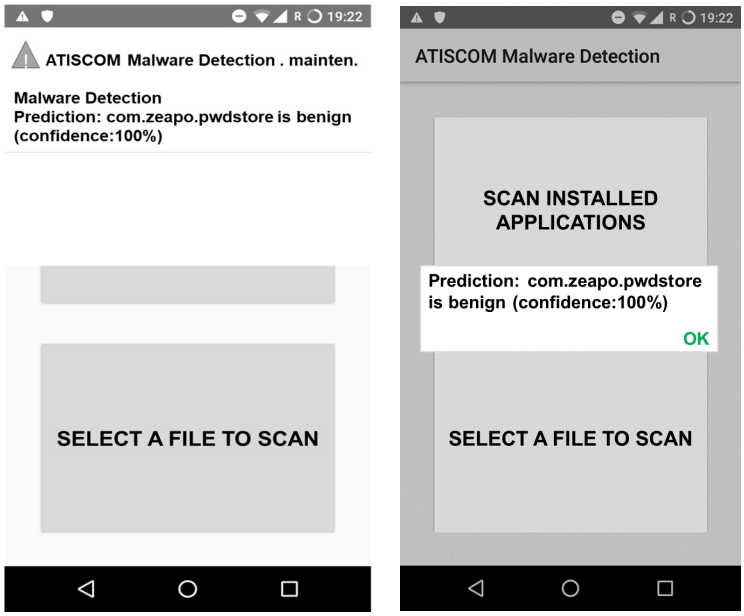
Our application is called "ATISCOM Malware Detection". It aims to protect the private information of mobile payment applications against malware attacks. Our application provides the ability to manually scan one or many of the existing installed applications and files on an Android mobile device (Figure 5), and automatically make predictions of the newly installed applications (Figure 6).



(**a**) (**b**) (**c**)

(**d**) (**e**) (**f**)

Figure 5. "ATISCOM Malware Detection Application" on Nexus 5 API 28: (**a**) home page; (**b**) list of the installed applications; (**c**) alert to scan manually one or multiple applications; (**d**) prediction results of the selected applications; (**e**) alert to scan all the installed applications at once; (**f**) prediction results of the installed applications all at once.



(**a**) (**b**)

Figure 6. Automatic malware detection on Nexus 5 API 28: (**a**) alert for prediction of a newly installed application; (**b**) notification of the prediction result.

Figure 5b depicts a list of installed applications on a Nexus 5 API 28, with the possibility to select manually one or multiple applications for scanning, by pressing the button SCAN (Figure 5c), or all at once by pressing the button SCAN ALL (Figure 5e). The results of the predictions are illustrated, respectively, for one or multiple installed scanned applications or files in Figure 5d, and for the installed applications scanned all at once, in Figure 5f.

Figure 6 represents the second functionality of the "ATISCOM Malware Detection" application, which aims to automatically scan the newly installed applications on the Android mobile device Nexus 5 API 28. In other words, the program detects any new installation through Android's BroadcastReceiver and scans the installed package. The prediction appears as a notification after a few seconds (Figure 6b).

5. Optimization of the Classification of the Applications

In this section, we performed measurements and tests of the collected applications on Android devices, in order to determine the best algorithm to choose for the classification of the applications (i.e., benign or malware). We conducted our tests by using: (1) the Naive Bayes algorithm to build the model; (2) the default classification algorithms included in Weka 3.8.2 to manage the certainty predictions and to determine the binary class (i.e., benign or malware); and (3) the default regression algorithms included in Weka 3.8.2, by associating numerical probabilistic values (i.e., −100 for benign and 100 for malware) to manage the uncertainty predictions.

5.1. Naive Bayes Method

We used the Naive Bayes algorithm to build the model on the collected data. To test the model, Weka offered cross-validation. The model was then trained on 90% of the data and tested on the remaining 10%. This was performed 10 times, in order to have tested the model on all the data by having trained it on all the others.

With our current dataset and the algorithm chosen, the accuracy of the model in cross-validation was as follows:

- Accuracy (correctly classified instances): 92.4486%;
- False positive rate (incorrectly classified benign applications): 0.217;

- False negative rate (malware not detected): 0.047.

In our case, it was more important to detect malware, which is the minority class, so we used the balanced sets. This was tested in cross-validation for Naive Bayes in Weka 3.8.2 on a representative set (i.e., 27,000 benign applications for 5000 malware) versus a balanced set (i.e., 5000 benign applications for 5000 malware), as illustrated in Tables 2 and 3.

Table 2. Global performance in cross-validation for Naive Bayes: representative set versus balanced set.

Set	Representative	Balanced
Correctly Classified Instances	31,016	9483
Correctly Classified Instances (%)	93.23	89.39
Kappa statistic	0.743	0.788
Mean absolute error	0.075	0.114
Root mean squared error	0.240	0.288
Relative absolute error (%)	28.01	22.98
Root relative squared error (%)	65.60	57.51
Total Number of Instances	33,269	10,608

Table 3. Detailed performance in cross-validation for Naive Bayes: a representative set versus a balanced set.

Set	Representative			Balanced		
Class	**Malware**	**Benign**	**Average**	**Malware**	**Benign**	**Average**
TPR	0.768	0.964	0.932	0.850	0.938	0.894
FPR	0.036	0.232	0.201	0.062	0.150	0.106
Precision	0.800	0.956	0.931	0.932	0.862	0.897
Recall	0.768	0.964	0.932	0.850	0.938	0.894
F-Measure	0.783	0.960	0.932	0.889	0.898	0.894
MCC [1]	0.743	0.743	0.743	0.791	0.791	0.791
ROC Area [2]	0.964	0.964	0.964	0.964	0.964	0.964
PRC Area [3]	0.836	0.993	0.968	0.960	0.967	0.963

[1] Matthews correlation coefficient. [2] Receiver operating characteristic. [3] Precision recall curve.

After testing in cross-validation for Naive Bayes, we constructed the confusion matrices based on representative and balanced sets, as shown in Figure 7.

a	b	<= classified as
4071	1233	a = malware
1020	26945	b = benign

(**a**)

a	b	<= classified as
4510	794	a = malware
331	4973	b = benign

(**b**)

Figure 7. Confusion matrices in cross-validation for Naive Bayes: (**a**) representative set; (**b**) balanced set.

It can be seen from the comparative Tables 2 and 3 and Figure 7 that, even if the overall accuracy (Table 2) seems better with the representative set, the detailed results tell a different story. The confusion matrices in Figure 7 clearly show that the classifier trained on the balanced set was better at detecting malware, even though, in proportion, more benign applications were misclassified. Therefore, we used the balanced set rather than the representative set for the remainder of the tests.

However, the detection of benign applications was our second criterion after malware, and the difference is not significant enough to justify taking this criterion into account ahead of the number of correctly classified malware.

5.2. Classification Method

We performed our tests on 37 complex classification algorithms included by default in Weka 3.8.2. We applied these tests on our dataset with two classes, benign, or malware.

Figure 8 depicts the overall performance (i.e., global accuracy and false-positive rates) of each of the 8 complex classification algorithms presented among the 37 tested ones. This allows us to define a numerical score showing the certainty of each of these classifiers.

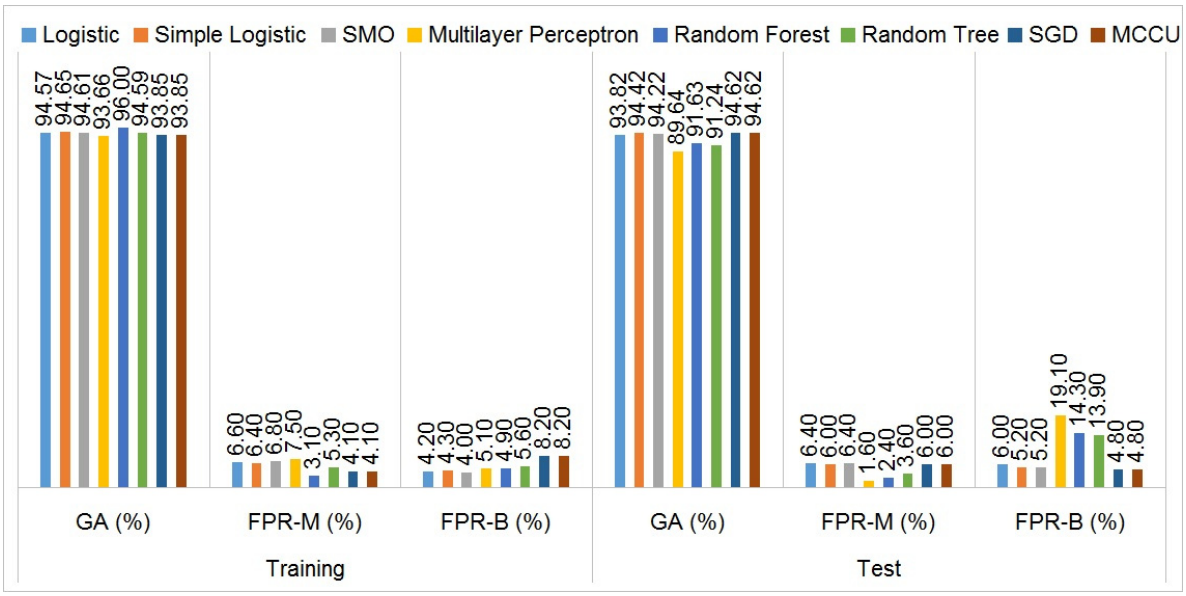


Figure 8. Comparison of the tested Weka classifiers.

We take into consideration for each classifier the following hyperparameters:

- The global accuracy (GA) corresponds to the percentage of correctly classified applications compared to the whole of the tested applications;
- The false-positive rate of malware (FPR-M) corresponds to the percentage of the number of benign applications classified as malware, divided by the number of benign applications in total;
- The false-positive rate of benign (FPR-B) applications represents the percentage of the number of malware applications that will not be detected, which corresponds to the number of malware applications classified as benign, divided by the number of malware applications in total.

Figure 8 highlights the superiority of some algorithms that we then studied in more detail. The classifiers in question with their exact configuration are as follows.

(1) Simple Logistic: weka.classifiers.functions.SimpleLogistic -I 0 -M 500 –H 50 -W 0.0
(2) SMO: weka.classifiers.functions.SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K ““weka.classifiers.functions.supportVector.PolyKernel –E 1.0 -C 250007”” -calibrator ““weka.classifiers.functions.Logistic –R 1.0E-8 -M -1 -num-decimal-places 4””
(3) SGD: weka.classifiers.functions.SGD -F 0 -L 0.01 -R 1.0E-4 -E 500 -C 0.001 -S 1
(4) MCCU: weka.classifiers.meta.MultiClassClassifierUpdateable -M 0 -R 2.0 -S 1 -W weka.classifiers.functions.SGD—-F 0 -L 0.01 -R 1.0E-4 -E 500 –C 0.001 -S 1

We then constructed confusion matrices, as shown in Figure 9, for each of these classifiers, and determined that the stochastic gradient descent (SGD) and multi-class

classifier updateable (MCCU) matrices are the most accurate classifiers among the above-presented classifiers.

a	b	<= classified as
238	13	a = malware
15	236	b = benign

(**a**)

a	b	<= classified as
238	13	a = malware
16	235	b = benign

(**b**)

a	b	<= classified as
239	12	a = malware
15	236	b = benign

(**c**)

a	b	<= classified as
239	12	a = malware
15	236	b = benign

(**d**)

Figure 9. Confusion matrices of the best classifiers on the test set: (**a**) simple logistic; (**b**) SMO; (**c**) SGD; (**d**) MCCU.

Moreover, these two classifiers are updateable, where data can be added to them after the construction of the model without having to rebuild everything. The only difference is that the second one is multiclass, which allows processing datasets with more than two classes. However, we can imagine cases where we would have, for example, three classes—malware, benign, and unsure. Hence, the drawback of this classification approach is that it does not manage the uncertainty predictions, and we must allow for the introduction of the user's judgment to decide about the uncertain cases. To manage the uncertainty predictions, we decided to use a probabilistic numerical attribute rather than increasing the number of classes. This will be discussed in the following subsection.

5.3. Regression Method

As noted above, one of the drawbacks of our classification approach is that it does not manage the uncertainty predictions or allow for the introduction of the user's judgment. This is therefore one of the motivations for the second approach using regression methods. Since these regression methods require numerical attributes to manage the uncertainty predictions, we prepared the data of our dataset by replacing our class dataset CLASS of type character (i.e., benign and malware) with a numerical value IS_MALWARE (i.e., −100 for benign and 100 for malware).

Table 4 represents the performance of the six selected regression algorithms among the 20 tested regression algorithms included by default in Weka 3.8.2: (1) random forest; (2) random committee; (3) random tree; (4) instance-based learner (IBK); (5) multilayer perceptron; and (6) linear regression.

The following algorithms, once trained, perform their prediction by assigning an IS_MALWARE score to the tested samples, and it was up to us to perform a second processing to decide which score was sufficient to guarantee our prediction. Unlike classifiers, we did not have a precise number of correct predictions or a confusion matrix, since the predictions can very well give −13, 0 or 56 and not 100 or −100 like labeled samples. Therefore, it is necessary to refer to the correlation coefficient, located between 0 and 100%, which indicates whether the predictions are close to reality, and the different error values are between reality and prediction. Moreover, it is crucial to take into account the time for the use of certain algorithms on the mobile phone.

Figure 10 depicts the correlation coefficient and the computation time of each of the 6 above selected regression algorithms.

In Figure 10a, we observe that the predictions of the random forest, random committee, random tree regression and instance-based learner algorithms give the best correlation coefficients on the validation set, compared to the multilayer perceptron and linear regression algorithms. On the other hand, the instance-based learner, multilayer perceptron and

linear regression algorithms indicate clearly that they require a significant computation time to do the predictions and they do not have a much better correlation coefficient than random forest, as shown in Figure 10b.

To be able to compare these above results (see Table 4 and Figure 10) to the classification methods, we further investigated the predictions of the algorithms with the best correlation coefficients on the validation set and the minimum computation time: (1) random forest; (2) random committee; and (3) random tree. We did not study the Multilayer Perceptron for the moment since it requires a great deal of resources to be computed and does not have a better performance than random forest, but we had to keep in mind that its study would be interesting, with the possibility of varying the hyperparameters to customize the neural network.

The simplest regression method consists of converting the predicted IS_MALWARE into a binary by adopting 0 as the value of the threshold, either in the case of the numerical prediction being strictly less than 0, where the prediction is considered to be benign, or in the case that the numerical prediction is greater than or equal to 0, where the prediction is considered to be malware. Therefore, the results are not ideal with this threshold, and we obtain important prediction errors for the regression algorithms (41 prediction errors for random forest, 42 for random committee, and 44 for random tree) even greater than the 27 (12 + 15) errors for 502 samples that were shown in Figure 9 for the classifiers.

We then conducted our simulations on different thresholds ranging from -90 to 90 in increments of 10. The results are much more promising on certain thresholds, the smallest numbers of errors, in this case, being: (1) 17 for random forest with a threshold of -30; (2) 18 for random committee with a threshold of -20; and (3) 19 for random tree with a threshold of -20 or -30. Consequently, the random forest algorithm is considered the best regression algorithm compared to the other tested ones.

For a more complex model that manages the uncertainty predictions, we could define not one but two thresholds for the algorithm. The idea would then be to have a margin between the two thresholds such that any prediction whose value is within the margin is considered uncertain.

Table 4. Comparison of the regression algorithms with the numerical class dataset.

Algorithm	Set	Correlation Coefficient	Mean Absolute Error	Root Mean Squared Error	Relative Absolute Error (%)	Root Relative Squared Error (%)	Time (s)
Random Forest	Training	0.9657	8.4154	26.040	8.415	26.040	7.13
	Test	0.9180	16.399	39.904	16.399	39.904	0.02
Random Committee	Training	0.9680	6.218	24.937	6.218	24.937	1.4
	Test	0.9170	15.312	39.984	15.312	39.984	0.01
Random Tree	Training	0.9680	6.218	24.937	6.218	24.937	0.15
	Test	0.9090	13.822	42.119	13.822	42.119	0.01
IBK	Training	0.9680	6.218	24937	6.218	24.937	43.04
	Test	0.9060	14.784	42.401	14.784	42.401	2.13
Multilayer Perceptron	Training	0.9550	11.0169	29.513	11.016	29.513	1259.67
	Test	0.9150	16.708	40.669	16.708	40.669	0.13
Linear Regression	Training	0.8730	36.578	48.772	36.578	48.772	6.73
	Test	0.8500	37.729	52.902	37.729	52.902	0.9

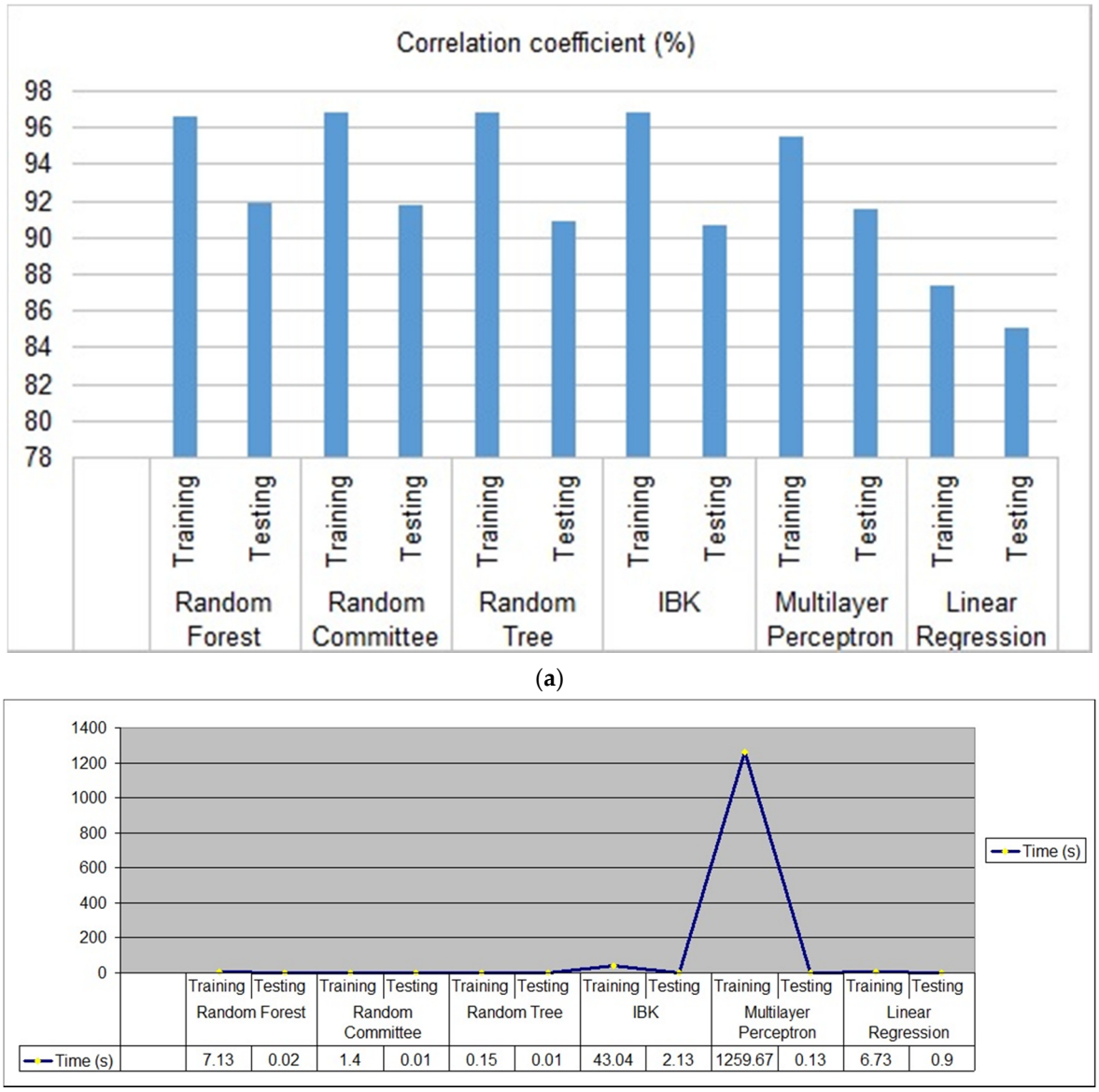


Figure 10. Comparison of the regression algorithms with the numerical class dataset: (**a**) on the correlation coefficient; (**b**) on the computation time.

One possible avenue to refine the proposed model would be to find an optimal margin for the uncertainties and to indicate to the user that our prediction was not precise enough for an application falling within this margin. However, for the moment, we limited our research work to the use of a simple threshold that gives very good results. Indeed, in our current proposed model, the addition of a third category has as yet no interesting uses.

6. Discussion

In this section, we evaluate the gradual improvement of the accuracy of the proposed malware detection method through three factors: (1) algorithm changes; (2) data additions; and (3) potential increase in computation time.

Indeed, in the first method using the Naive Bayes algorithm, the prediction of the class of an application is almost instantaneous. However, the accuracy of the model is less than 90%.

In the second method using both classification algorithms of stochastic gradient descent (SGD) and multi-class classifier updateable (MCCU), we obtained 93.85% as global

accuracy on the training set and 94.62% on the test set. On the other hand, these classifiers do not manage the uncertainty predictions.

To remedy the problem of uncertainty predictions, we adopted the third method of regression by converting the character values of the class (i.e., benign and malware) to numerical values (i.e., −100 for benign and 100 for malware). With the model of regression based on random forest, which is our best method, we obtained a good accuracy for prediction in terms of performance with a good correlation coefficient, a minimum computation time and the smallest number of errors for malware detection.

On the other hand, making a prediction on a mobile device (i.e., off-device) takes 10 to 15 s for each application on MotoG LTE (first generation), which is an important element to be taken into account. Moreover, it becomes very significant when we try to scan on a mobile device all the installed applications in one shot. This can therefore become a problem for the user and even seems unpractical, especially if the user thinks that the application has crashed and will close it down. To reduce the computation time, we adopt a client/server architecture that allows data to be sent to the server for remote processing (i.e., offloading) of prediction using the Random Forest regression algorithm, and returns to the client the result of prediction as a simple integer, between −100 and 100. This latter is considered a very low data weight for communication between the server and the client using USSD technology.

7. Conclusions

In this paper, we proposed a prediction model to secure mobile payment applications on Android devices based on client/server architecture to palliate the heavy computational load on mobile devices for malware detection. Furthermore, we optimized our malware detection methodology for better accuracy and minimum computation time. We adopted the random forest regression algorithm included by default in Weka 3.8.2 for remote processing (i.e., offloading) of the predictions on the server, with a numerical class ranging from −100 to 100. We obtained a good accuracy for prediction in terms of performance with a good correlation coefficient, a minimum computation time and the smallest number of errors for malware detection. We have limited our research work to the prediction of the newly installed applications by adopting a fixed threshold value for decision making (i.e., −100 for benign or 100 for malware). In addition, we implemented our proposed model to validate our methodology. Our "ATISCOM Malware Detection" application was tested on various Android devices (i.e., MotoG LTE (first generation) and Nexus 5 API 28) and we obtained good prediction results.

In future work, it will be crucial to make our model a hybrid that will be able to support both static and dynamic malware detection methods, as well as to take into account the automatic update of our dataset implemented on the server with data of the most recent applications. Moreover, to refine our proposed model it would be fundamental to find an optimal margin between two thresholds instead of one fixed threshold value, in order to manage the uncertainty predictions.

Author Contributions: Conceptualization, A.F., F.E.K. and S.P.; methodology, A.F., F.E.K. and S.P.; software, A.F. and F.E.K.; validation, A.F., F.E.K. and S.P.; formal analysis, A.F. and F.E.K.; data curation, A.F. and F.E.K.; writing—original draft preparation, F.E.K.; writing—review and editing, F.E.K.; visualization, F.E.K.; supervision, S.P.; project administration, S.P. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC), Prompt Quebec, Flex Groups and Q-Links.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Publicly available datasets for TheZoo [39] and Contagio [40] were analyzed in this study. This data can be found here: [https://thezoo.morirt.com/, accessed on 23

June 2021] and [https://contagiodump.blogspot.com/, accessed on 23 June 2021], respectively. The data presented in this study for Drebin [18] are available on request from the corresponding author on drebin@sectubs.de. The data are not publicly available due to [download policy].

Acknowledgments: We would like to express our gratitude to Flex Groups teams for their technical support.

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. Diogenes, Y.; Ozkaya, E. *Cybersecurity—Attack and Defense Strategies*, 2nd ed.; Packt Publishing Ltd.: Birmingham, UK, 2019.
2. Source. Android Runtime (ART) and Dalvik. Available online: https://source.android.com/devices/tech/dalvik (accessed on 10 March 2021).
3. Kumar, M. Dynamic Analysis Tools for Android Fail to Detect Malware with Heuristic Evasion Techniques. The Hackers News. 2014. Available online: https://thehackernews.com/2014/05/dynamic-analysis-tools-for-android-fail.html (accessed on 12 March 2021).
4. Aptoide. Available online: https://fr.aptoide.com/ (accessed on 25 February 2021).
5. Lookout. Available online: https://www.lookout.com/ (accessed on 6 March 2021).
6. Wandera. Available online: https://www.wandera.com/miriam/ (accessed on 7 March 2021).
7. Zimperium Advanced Mobile Security. Available online: https://www.zimperium.com/technology (accessed on 12 March 2021).
8. ZDNet. Available online: https://www.zdnet.com/article/symantec-buys-mobile-security-startup-skycure/ (accessed on 13 March 2021).
9. Cog™. Available online: https://cog.systems/collateral/datasheet-htc-a9-by-d4-lp/ (accessed on 13 March 2021).
10. Jawale, A.S.; Park, J.S. A Security Analysis on Apple Pay. In Proceedings of the 2016 European Intelligence and Security Informatics Conference (EISIC), Uppsala, Sweden, 17–19 August 2016; pp. 160–163. [CrossRef]
11. Broadcom. Multi-Factor Authentication: Headache for Cyber Actors Inspires New Attack Techniques. Available online: https://symantec-enterprise-blogs.security.com/blogs/threat-intelligence/multi-factor-authentication-new-attacks (accessed on 23 May 2021).
12. Etaher, N.; Weir, G.R.S.; Alazab, M. From ZeuS to Zitmo: Trends in Banking Malware. In Proceedings of the IEEE 14th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom), Helsinki, Finland, 20–22 August 2015; pp. 1386–1391. [CrossRef]
13. Wang, Y.; Hahn, C.; Sutrave, K. Mobile payment security, threats, and challenges. In Proceedings of the IEEE Second International Conference on Mobile and Secure Services (MobiSecServ), Gainesville, FL, USA, 26–27 February 2016; pp. 1–5. [CrossRef]
14. Xiao, L.; Li, Y.; Huang, X.; Du, X. Cloud-Based Malware Detection Game for Mobile Devices with Offloading. *IEEE Trans. Mob. Comput.* **2017**, *16*, 2742–2750. [CrossRef]
15. Cheng, Z.; Chen, X.; Zhang, Y.; Li, S.; Sang, Y. Detecting Information Theft Based on Mobile Network Flows for Android Users. In Proceedings of the IEEE 2017 International Conference on Networking, Architecture, and Storage (NAS), Shenzhen, China, 7–9 August 2017; pp. 1–10. [CrossRef]
16. Fournier, A.; El Khoury, F.; Pierre, S. Classification Method for Malware Detection on Android Devices. In *Advances in Intelligent Systems and Computing*; Springer: Cham, Switzerland, 2021; pp. 810–829.
17. Weka. The Workbench for Machine Learning. Available online: https://www.cs.waikato.ac.nz/ml/weka/ (accessed on 16 March 2021).
18. Arp, D.; Spreitzenbarth, M.; Hübner, M.; Gascon, H.; Rieck, K. Drebin: Effective and Explainable Detection of Android Malware in Your Pocket. NDSS Symposium. 2014. Available online: https://www.ndss-symposium.org/ndss2014/programme/drebin-effective-and-explainable-detection-android-malware-your-pocket/ (accessed on 14 March 2021).
19. Gharib, A.; Ghorbani, A. DNA-Droid: A Real-Time Android Ransomware Detection Framework. In *Network and System Security*; Springer: Cham, Switzerland, 2017; pp. 184–198.
20. Wang, Y.; Alshboul, Y. Mobile security testing approaches and challenges. In Proceedings of the IEEE First Conference on Mobile and Secure Services (MobiSecServ), Gainesville, FL, USA, 20–21 February 2015; pp. 1–5. [CrossRef]
21. Saracino, A.; Sgandurra, D.; Dini, G.; Martinelli, F. MADAM: Effective and Efficient Behavior-based Android Malware Detection and Prevention. *IEEE Trans. Dependable Secur. Comput.* **2018**, *15*, 83–97. [CrossRef]
22. Burguera, I.; Zurutuza, U.; Nadjm-Tehrani, S. Crowdroid: Behavior-based malware detection system for android. In Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, Chicago, IL, USA, 17 October 2011; pp. 15–26. [CrossRef]
23. G. Inc. Machine Learning Crash Course. 2020. Available online: https://developers.google.com/machine-learning/crash-course/classification/ (accessed on 11 February 2021).
24. Microsoft; Azure. Machine Learning—Evaluate. 2019. Available online: https://docs.microsoft.com/en-us/azure/machine-learning/studio-module-reference/machine-learning-evaluate (accessed on 12 February 2021).

25. Yalew, S.D.; Maguire, G.Q.; Haridi, S.; Correia, M. T2droid: A TrustZone-Based Dynamic Analyser for Android Applications. In Proceedings of the 16th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (Trustcom/BigDataSE/ICESS), Sydney, Australia, 1–4 August 2017; pp. 240–247. [CrossRef]
26. Ahmadi, M.; Sotgiu, A.; Giacinto, G. IntelliAV: Building an Effective On-Device Android Malware Detector. 2018. Available online: http://arxiv.org/abs/1802.01185 (accessed on 17 February 2021).
27. Virustotal. Available online: https://www.virustotal.com/gui/home/upload (accessed on 19 February 2021).
28. Aonzo, S.; Merlo, A.; Migliardi, M.; Oneto, L.; Palmieri, F. Low-Resource Footprint, Data-Driven Malware Detection on Android. *IEEE Trans. Sustain. Comput.* **2020**, *5*, 213–222. [CrossRef]
29. Ruiz-Heras, A.; García-Teodoro, P.; Sánchez-Casado, L. ADroid: Anomaly-based detection of malicious events in Android platforms. *Int. J. Inf. Secur.* **2017**, *16*, 371–384. [CrossRef]
30. Sun, M.; Li, X.; Lui, J.C.S.; Ma, R.T.B.; Liang, Z. Monet: A User-Oriented Behavior-Based Malware Variants Detection System for Android. *IEEE Trans. Inf. Forensics Secur.* **2017**, *12*, 1103–1112. [CrossRef]
31. Arshad, S.; Shah, M.A.; Wahid, A.; Mehmood, A.; Song, H.; Yu, H. SAMADroid: A Novel 3-Level Hybrid Malware Detection Model for Android Operating System. *IEEE Access* **2018**, *6*, 4321–4339. [CrossRef]
32. Salehi, M.; Amini, M. Android Malware Detection Using Markov Chain Model of Application Behaviors in Requesting System Services. 2017. Available online: http://arxiv.org/abs/1711.05731 (accessed on 13 January 2021).
33. Wang, X.; Yang, Y.; Zeng, Y.; Tang, C.; Shi, J.; Xu, K. A Novel Hybrid Mobile Malware Detection System Integrating Anomaly Detection with Misuse Detection. In Proceedings of the 6th International Workshop on Mobile Cloud Computing and Services (MCS'15), Paris, France, 11 September 2015; pp. 15–22. [CrossRef]
34. De Souza, C.; Guimarães, A.J.; Rezende, T.S.; Souza Araujo, V.; Do Nascimento, L.A.F.; Oliveira Batista, L. An Intelligent Hybrid Model for the Construction of Expert Systems in Malware Detection. In Proceedings of the IEEE Workshop on Evolving and Adaptive Intelligent Systems (EAIS), Bari, Italy, 27–29 May 2020; pp. 1–8. [CrossRef]
35. Almshari, M.; Tsaramirsis, G.; Khadidos, A.O.; Buhari, S.M.; Khan, F.Q.; Khadidos, A.O. Detection of Potentially Compromised Computer Nodes and Clusters Connected on a Smart Grid, Using Power Consumption Data. *Sensors* **2020**, *20*, 5075. [CrossRef] [PubMed]
36. Alzaylaee, M.K.; Yerima, S.Y.; Sezer, S. Emulator vs. real phone: Android malware detection using machine learning. In Proceedings of the 3rd ACM on International Workshop on Security and Privacy Analytics, Scottsdale, AZ, USA, 22–24 March 2017; pp. 65–72. [CrossRef]
37. Felt, A.P.; Wang, H.J.; Moshchuck, A.; Hanna, S.; Chin, E. Permission Re-Delegation: Attacks and Defenses. In Proceedings of the 20th USENIX Security Symposium, San Francisco, CA, USA, 10–12 August 2011; pp. 1–16. Available online: https://www.usenix.org/legacy/event/sec11/tech/full_papers/Felt.pdf?hl=JA (accessed on 22 May 2021).
38. Developers. Documentation: Manifest. Permissions. Available online: https://developer.android.com/reference/android/Manifest.permission (accessed on 22 May 2021).
39. TheZoo Aka Malware DB. Available online: https://thezoo.morirt.com/ (accessed on 13 March 2021).
40. Contagio Malware Dump. Available online: https://contagiodump.blogspot.com/ (accessed on 13 March 2021).
41. Google Play. Available online: https://play.google.com/store (accessed on 12 March 2021).
42. Lakshmi, K.K.; Gupta, H.; Ranjan, J. USSD—Architecture Analysis, Security threats, Issues and Enhancements. In Proceedings of the IEEE International Conference on Infocom Technologies and Unmanned Systems (ICTUS'2017), Dubai, United Arab Emirates, 18–20 December 2017; pp. 798–802. [CrossRef]