IntelliAV: Toward the Feasibility of Building Intelligent Anti-malware on Android Devices

Mansour Ahmadi, Angelo Sotgiu, Giorgio Giacinto

IntelliAV: Toward the Feasibility of Building Intelligent Anti-Malware on Android Devices

Mansour Ahmadi, Angelo Sotgiu, and Giorgio Giacinto

University of Cagliari, Italy

Abstract. Android is targeted the most by malware coders as the number of Android users is increasing. Although there are many Android anti-malware solutions available in the market, almost all of them are based on malware signatures, and more advanced solutions based on machine learning techniques are not deemed to be practical for the limited computational resources of mobile devices. In this paper we aim to show not only that the computational resources of consumer mobile devices allow deploying an efficient anti-malware solution based on machine learning techniques, but also that such a tool provides an effective defense against novel malware, for which signatures are not yet available. To this end, we first propose the extraction of a set of lightweight yet effective features from Android applications. Then, we embed these features in a vector space, and use a pre-trained machine learning model on the device for detecting malicious applications. We show that without resorting to any signatures, and relying only on a training phase involving a reasonable set of samples, the proposed system outperforms many commercial anti-malware products, as well as providing slightly better performances than the most effective commercial products.

Keywords: Android, Malware Detection, Machine Learning, On-Device, TensorFlow, Mobile Security, Classification

1 Introduction

Nowadays, mobile devices are ubiquitous tools for everyday life. Among them, Android devices dominated the global smartphone market, with nearly 90% of the market share in the second quarter of 2016 [25]. The majority of the security issues affecting Android systems can be attributed to third party applications (app) rather than to the Android OS itself. Based on F-secure reports on mobile threats [35], researchers found 277 new malware families, among which 275 specifically targeting Android devices. Also other recent reports clearly show that the malware infection rate of Android mobile devices is soaring. In particular, a report from McAfee [30] reported a significant growth of mobile malware in the wild. We believe that this huge amount of mobile malware needs to be timely detected, possibly by smart tools running on the device, because it has been shown that malware can bypass offline security checks, and live in the wild

for a while. As a matter of fact, to the best of our knowledge, even the most recent versions of Android anti-malware products are still not intelligent enough to catch most of the novel malware.

The success of machine learning approaches for malware detection and classification [5, 8, 41, 36, 26], as well as the advance in machine learning software for the execution in mobile environments, motivated us to empower Android devices with a machine-learning anti-malware engine. Although modern mobile devices come to the market with a huge amount of computational power, the development of any Android anti-malware product should consider its efficiency on the device to avoid battery drain, in particular when machine learning techniques are employed, as they are known to be computational demanding. On the other hand, we observe that an intelligent Android anti-malware product doesn't need to be unnecessarily complex, as it has been shown that Android malware executes simpler tasks than the desktop counterparts [7]. All the aforementioned reasons motivate the proposal for a machine learning solution to be deployed on mobile devices to detect potential malicious software.

1.1 On-Device Advanced Security

Although there are many offline systems proposed for mobile malware detection, mostly based on machine learning approaches (see Section 5), there are many reasons for a user to have an intelligent security tool capable of identifying potential malware on the device.

(*i*) The Google Play store is not totally free of malware. There has been many reports that have shown that malware could pass the Google security checks, and remain accessible to users for sometime on the Play store until someone flags it as inappropriate. For instance, the Check Point security firm reported a zero-day mobile ransomware found in Google Play in January 2017, which was dubbed as a *Charger* application, and was downloaded by more than a million users [31]. Another report from the same vendor cites the case of new variants of the famous Android malware family HummingBad [33]. We vet these samples in Section 3.2.

(*ii*) Third-party app stores are popular among mobile users, because they usually offer applications at great discounts. Moreover, the Google Play store has restricted access in some countries, so people have to download their required applications from third-party app stores. Nevertheless, security checks on the third-party stores are not as effective as those available on the Google Play store. Therefore, third-party markets are a good source of propagation for mobile malware. Many malware samples have been found on these stores during the past years, that were downloaded by millions of users. In addition, quite often users can be dodged by fake tempting titles like free games when browsing the web, so that applications are downloaded and installed directly on devices from untrusted websites. Another source of infection is phishing SMS messages that contain links to malicious applications. Recent reports by Lookout and Google [27, 24] show how a targeted attack malware, namely *Pegasus*, which is suspected to infect devices via a phishing attack, could remain undetected for a few years. We vet these samples in Section 3.2.

(*iii*) One of the main concerns for any 'computing' device in the industry, is to make sure that the device a user buys is free of malware. Mobile devices make no exception, and securing the 'supply chain' is paramount difficult, for the number of people and companies involved in the supply chain of the components. There is a recent report that shows how some malware were added to Android devices somewhere along the supply chain, before the user received the phone [32]. We vet these samples in Section 3.2.

(*iv*) To the best of our knowledge, almost all of the Android anti-malware products are mostly signature-based, which lets both malware variants of known families, and zero-day threats to devices. There are claims by a few Android anti-malware vendors that they use machine learning approaches, even if no detail is available on the mechanisms that are actually implemented on the device. We analyze this issue in more details in Section 3.2.

All of the above observations show that an anti-malware solution based on machine-learning approaches, either completely, or as a complement to signatures, can reduce the vulnerability of Android devices against novel malware.

1.2 Contribution

Accordingly, in this paper we introduce `IntelliAV`[1], which is a practical intelligent anti-malware solution for Android devices based on the open-source and multi-platform TensorFlow library. It is worth to mention that this paper does not aim to propose yet another learning-based system for Android malware detection, but by leveraging on the existing literature, and on previous works by the authors, we would like to test the feasibility of having an on-device intelligent anti-malware tool to tackle the deficiencies of existing Android anti-malware products, mainly based on pattern matching techniques. To the best of our knowledge, the performances of learning-based malware detection systems for Android have been only tested off-device, i.e., with computational power and memory space well beyond the capabilities of mobile devices. More specifically, the two main contributions of `IntelliAV` are as follows:

(*i*) We propose a machine-learning model based on lightweight and effective features extracted on a substantial set of applications. The model is carefully constructed to be both effective and efficient by wisely selecting the features, the model, and by tuning the parameters as well as being precisely validated to be practical for the capabilities of Android devices.

(*ii*) We show how the proposed model can be embedded in the `IntelliAV` application, and easily deployed on Android devices to detect new and unseen malware. Performance of `IntelliAV` has been evaluated by cross-validation, and achieved 92% detection rate that is comparable to other off-device learning-based Android malware detection relying on a relatively small set of features. Moreover, `IntelliAV` has been tested on a set of unseen malware, and achieved 72% detection rate that is higher than the top 5 commercial Android anti-malware products.

[1] http://www.intelliav.com

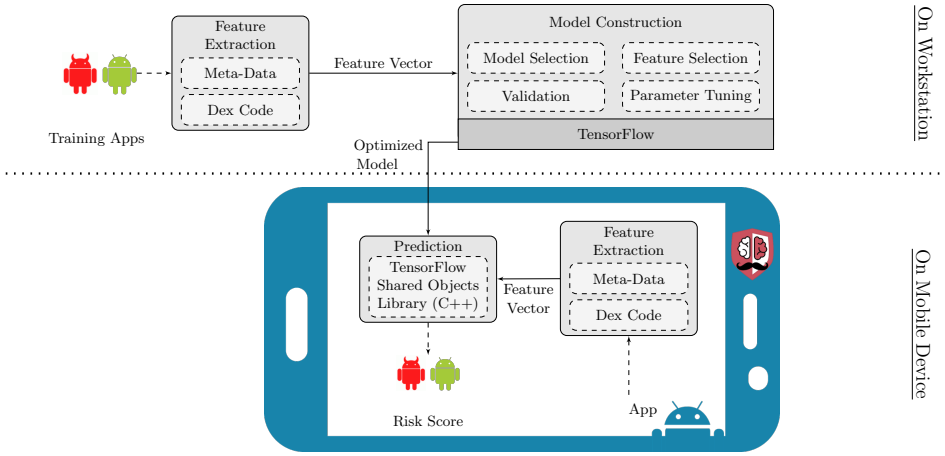


Fig. 1: Overview of `IntelliAV`.

The rest of the paper is organized as follows:
First, we reveal the detail of `IntelliAV` by motivating the choice of features and the procedure followed to construct the model (§2). We then present the experimental setup and results (§3). After that, we briefly mention the limitations of IntelliAV (§4) and review the related works on Android malware detection (§5). Finally, we conclude our paper discussing future directions of `IntelliAV` (§6).

2 System Design

The architecture of the proposed `IntelliAV` system is depicted in Figure 1, and its design consists of two main phases, namely offline training the model, and then its operation on the device to detect potential malware samples. As a first phase, a classification model is built offline, by resorting to a conventional computing environment. It is not necessary to perform the training phase on the device, because it has to be performed on a substantial set of samples whenever needed to take into account the evolution of malware. The number of times the model needs to be updated should be quite small, as reports showed that just the 4% of the total number of Android malware is actually new malware [10]. To perform the training phase we gathered a relatively large number of applications (§3.1). Then, a carefully selected set of characteristics (features) is extracted from the applications to learn a discriminant function allowing the distinction between malicious and benign behaviors (§2.1). Next, the extracted features are passed to the model construction step in which a classification function is learnt by associating each feature to the type of applications it has been extracted from, i.e., malware or goodware (§2.2). Finally, as the second phase, the model is embedded in the `IntelliAV` Android application that will provide a risk score for each application on the device (§2.3).

2.1 Feature Extraction

The feature extraction step is the core phase for any learning-based system. Various kinds of features have been proposed for Android malware detection by the security community, such as permissions, APIs, API dependencies, Intents, statistical features, etc. (see Section5 for a detailed discussion on the issue of feature extraction for Android malware detection). However, some sets of features related to basic Android behaviors, like permissions, APIs, and Intents, usually allow achieving reasonable detection results, with the aim to alert for the presence of probably harmful applications [8, 36]. Extracting this set of features is also feasible on mobile devices because they do not need deep static analysis, thus requiring a limited computational effort. Therefore, with the aim of extracting a set of efficient and effective features for `IntelliAV`, we resorted to the following four sets of features: permissions, Intent Filters, statistical features based on the '*manifest*' of Android applications, and the APIs, which are extracted from the dex code. Therefore, to construct the feature vector, we considered all the permissions and intent-filters that are used by the samples included in the training set. In addition, four statistical features from application's components such as the total number of activities, services, broadcast receivers, and content providers are added to the feature vector as they can reveal the amount of abilities each application has. For instance, the number of activities in many malware categories is usually fewer than the number of activities available in benign applications, except for the case of malware that is built by repackaging benign applications. Moreover, we manually selected a set of 179 APIs as features and included in the feature vector. The selected APIs are those that reveal some particular characteristics of application that are known to be peculiar to either goodware or malware. For instance, the `invoke` API from the `java.lang.reflect.Method` class shows whether an application uses reflection or not. Note that permissions and APIs are coded as binary features, which means that their value is either one or zero depending on the feature being or not present in the application. By contrast, intent-filters are integer-valued features, as they represent the number of times an intent-filters is declared in the manifest. Considering this count for intent-filter features makes them more meaningful rather than simply considering their presence or not in the application. Similarly, the application's components are represented as integer valued features, as we count the number of components for each different type (e.g., activities, services, etc.). On the other hand, if we considered the number of permissions, we would have ended up with useless information, as each permission needs to be declared just once in the `manifest`. The same reasoning motivates the use of binary feature to represent API usage. The main reason is that although it is possible to get the count of the usage of an API in an application, the procedure would increase the processing time without producing more useful information, so that we ignored it. In total, the feature vector contains 3955 features. To avoid overfitting, and make `IntelliAV` faster on the mobile device, we decided to reduce the number of feature by selecting the 1000 meaningful features thorough a feature selection procedure (see Section 2.2). The final set

Table 1: Features used in `IntelliAV`.

Category	Number of Features	Type
Meta-Data		
Permissions	322	Binary
Intent Filters	503	Count
Statistical	4	Count
Dex Code		
APIs	171	Binary

consists of 322 features related to permissions, 503 features related to Intent filters, 4 statistical features from components (e.g., count of activities), and 171 features related to API usage (see Table 1).

2.2 Model Construction

To discriminate malware from benign applications, we need to rely on binary classification algorithms. Over the past years, a large number of classification techniques have been proposed by the scientific community, and the choice of the most appropriate classifier for a given task is often guided by previous experience in different domains, as well as by trial-and-error procedures. However, among all of the existing classifiers, Random Forest classifier [14] have shown high performances in a variety of tasks [19]. Random Forests algorithm is an ensemble learning method in which a number of decision trees are constructed at training time by randomly selecting the features used by each decision tree, and it outputs the class of an instance at testing time based on the collective decision of the ensemble. As far as the Random forest model is an ensemble classifier, it often achieves better results than a single classifier. The main reason of achieving good results by Random Forests is that ensemble methods reduce the variance in performances of a number of decision trees, which in turn are complex models with low bias. So, the final model exhibits low bias, and low variance, which makes the model more robust against both the *underfitting* and *overfitting* problems [13].

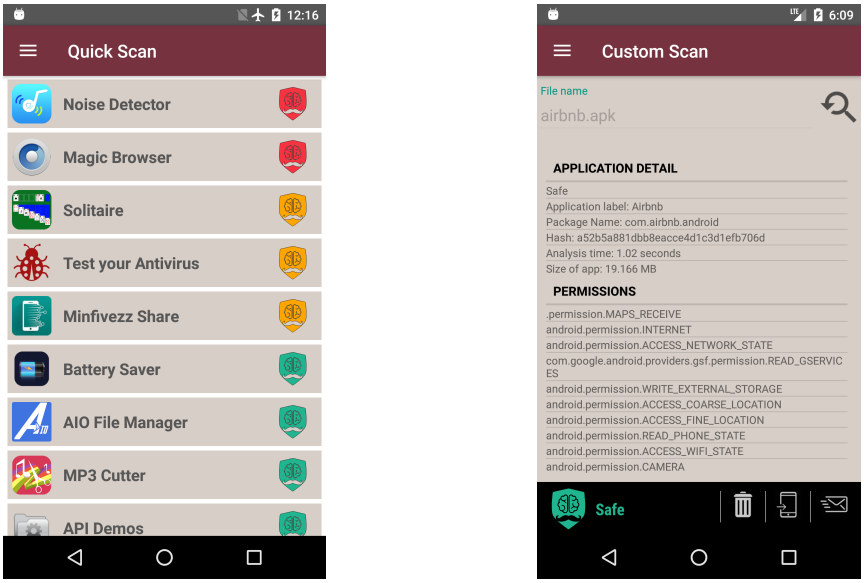
To be able to train our model offline, as well as to test it on Android devices, we built `IntelliAV` on top of TensorFlow [2]. More specifically, we employ an implementation of Random Forests in TensorFlow, called TensorForest [16]. TensorFlow is an open source library for machine learning, which was released by Google in November 2015. To the best of our knowledge, `IntelliAV` is the first anti-malware tool that has proposed employing TensorFlow. The TensorFlow model is highly portable as it supports the vast majority of platforms such as Linux, Mac OS, Windows, and mobile computing platforms including Android and iOS. TensorFlow computations are expressed as data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicating between them.

As mentioned in the previous subsection, to simplify the learning task and reduce the risk of the so-called *overfitting* problem, i.e., to avoid that the model

fits the training set but exhibits a low generalization capability with respect to novel unknown samples, we exploited feature selection that reduced the feature set size by removing irrelevant and noisy features . In particular, as done in [4], we computed the so-called *mean decrease impurity* score for each feature, and retained those features which have been assigned the highest scores. Note that the mean decrease impurity technique is often referred to as the Gini impurity, or information gain criterion.

2.3 On-Device Testing

As we mentioned before, TensorFlow eases the task of using machine learning models on mobile devices. So, we embedded in `IntelliAV` the trained model obtained according to the procedure described in Section 2.2. The size of the TensorFlow models depends on the complexity of the model. For instance, if the number of trees in TensorForest increases, consequently the size of the model increases as well. The size of `IntelliAV` model that we obtained according to the above procedure and that we transferred to the device, is about 14.1MB. Having said that, when it is embedded into the apk, the model is compressed and the total size of the model becomes just 3.3MB. Whenever an application needs to be tested, first, `IntelliAV` extracts the features from the application on the device, then it loads the model, and finally it feeds the model by the extracted features to get the application's risk score. The model provides a likelihood value between 0 and 1, denoting the degree of maliciousness of the application, that we scale to a percentage that we called *risk score*, to make it more understandable for the end user. We empirically provide the following guideline for interpreting the risk score. If the risk score is lower than 40%, the risk is low and we suggest to consider the application as being benign. If the risk score is between 40% and 50%, then the application should be removed if the user isn't sure about the trustworthiness of the application. Finally, the application has to be removed if the risk score is higher than 50%. These thresholds have been set after testing the system on a set containing different applications. We deployed `IntelliAV` so that two main abilities are provided, as shown in figure 2. `IntelliAV` can scan all of the installed applications on the device, and verify their risk scores (Quick Scan). In addition, when a user downloads an apk, it can be analyzed by `IntelliAV` before installation to check the related risk score, and take the appropriate decision (Custom Scan). To access the contents of an application's package on the external storage, `IntelliAV` needs the `READ_EXTERNAL_STORAGE` permission. To access the contents of the packages of installed applications, `IntelliAV` needs to read `base.apk` in a sub-directory with a name corresponding to the package name, which is located in */data/app/* directory. As far as the permission of `base.apk` file is `-rw-r--r--`, which means every user can read the content of this file, `IntelliAV` doesn't need neither any permission, nor a rooted device to evaluate the installed applications.



(a) Scan installed applications (b) Scan an APK

Fig. 2: `IntelliAV` abilities.

3 Experimental Analysis

In this section, we address the following research questions:

- Is `IntelliAV` able to detect new and unseen malware (§3.2)?
- Are the performances of `IntelliAV` comparable to the ones of popular mobile anti-malware products, although `IntelliAV` is completely based on machine learning techniques (§3.2)?
- Which is the overhead of `IntelliAV` on real devices (§3.3)?

Before addressing these questions, we discuss the data used, and the experimental settings of our evaluation (§3.1).

3.1 Experimental Setup

To evaluate `IntelliAV`, we have collected 19,722 applications, divided into 10,058 benign and 9,664 malicious applications from VirusTotal[39]. When gathering malicious applications, we considered their diversity, by including samples belonging to different categories, such as Adware, Ransomware [6, 28], GCM malware [3], etc. All of the gathered samples have been first seen by VirusTotal between January 2011 and December 2016. The whole process of feature extraction and model construction was carried out on a laptop with a 2 GHz quad-core

processor and 8GB of memory. Two metrics that are used for evaluating the performance of our approach are the False Positive Rate (FPR) and the True Positive Rate (TPR). FPR is the percentage of goodware samples misclassified as badware, while TPR is the fraction of correctly-detected badware samples (also known as detection rate). A Receiver-Operating-Characteristic (ROC) curve reports TPR against FPR for all possible model's decision thresholds.

3.2 Results

To better understand the effectiveness of `IntelliAV`, we evaluate it in following scenarios.

Cross Validation One might fit a model on the training set very well, so that the model will perfectly classify all of the samples that are used during the training phase. However, this might not provide the model with the generalization capability, and that's why we evaluated the model by a cross-validation procedure to find the best tuned parameters to be used for constructing the final model as a trade-off between correct detection and generalization capability. Consequently, we evaluated `IntelliAV` on the set of applications described in Section 3.1 through a 5-fold cross validation, to provide statistically-sound results. In this validation technique, samples are divided into 5 groups, called folds, with almost equal sizes. The prediction model is built using 4 folds, and then it is tested on the final remaining fold. The procedure is repeated 5 times on different folds to be sure that each data point is evaluated exactly once. We repeated the procedure by running the Random Forest algorithm multiple times to obtain the most appropriate parameters. The ROC of the best fitted model is shown in Figure 3. The values of FPR and TPR are respectively 4.2% and 92.5% which is quite acceptable although the set of considered features is relatively small, namely 1000 features.

Evaluation on the training set To verify the effectiveness of the tuned parameters based on the cross-validation procedure explained in Section 3.2, we tested the model on all the samples used for training. Table 2 shows the results on the training set. It shows that `IntelliAV` misclassified just a few training samples. This shows how the model is carefully fitted on the training set, so that is able to correctly classify almost all of the training samples with very high accuracy, while it avoids being overfitted, and thus can detect unseen malware with a high accuracy as well (see the following).

Evaluation on new Malware We then tested the system on a set made up of 2311 malware samples, and 2898 benign applications, that have been first seen by VirusTotal between January and March of 2017. We considered an application as being malicious when it was labeled as malware by at least 5 of the tools used by VirusTotal. This set of test samples contains randomly selected applications

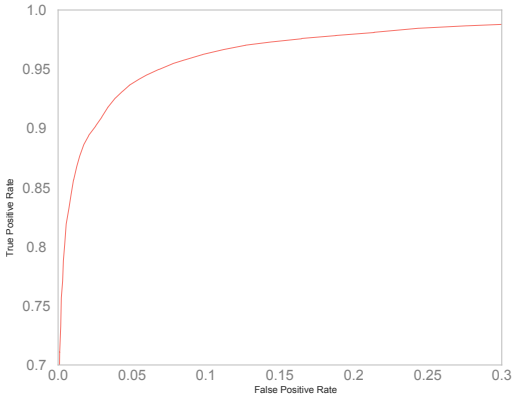


Fig. 3: ROC curve of TensorForest (5-fold cross validation). FPR and TPR are respectively 4.2% and 92.5%.

Table 2: Training on the set of samples explained in Section 3.1 and testing on the same set. GT refers to the Ground-truth of samples.

Train	Test		
#Samples	GT (#Samples)	Classified as Malicious	Benign
19,722	Malicious (9,664)	9,640 (TPR = 99.75%)	24
	Benign (10,058)	7 (FPR = 0.07%)	10,051

that were newer than the samples in the training set, and thus they were not part of the training set.

Test results are shown in Table 3. The detection rate on the test set is 71.96%, which is quite good if compared with the performances of other Android anti-malware solutions that are available in the market, as shown in Section 3.2. Moreover, the false positive rate is around 7.52%, which is acceptable if we consider that an individual user typically installs a few dozen applications, and thus it might receive a false alert from time to time. This casual alert allows the user that the application has some characteristics similar to badware, and so it can be used only if the source is trusted. It is also worth noting that our classification of false positives is related to the classification provided by VirusTotal at the time of writing. It is not unlikely that some of these applications might turn out to be classified as malware by other anti-malware tools in the near future, as we have already noticed during the experiments. However, due to the small time frame, we haven't the possibility to collect enough examples to provide reliable statistics, as the samples used for the test phase are quite recent. We expect in

Table 3: Training on the set of samples described in Section 3.1, and testing on new samples in 2017. GT refers to the Ground-truth of samples.

Train	Test		
#Samples	GT (#Samples)	Classified as Malicious	Benign
19,722	Malicious (2311)	1,663 (TPR = 71.96%)	648
	Benign (2898)	218 (FPR = 7.52%)	2,680

a future work to show how many applications were correctly predicted as being malicious before their signatures were created. However, our experience suggests that even if the application is benign but labeled as being potentially risky by `IntelliAV`, then the user might look for less risky alternatives applications in Google Play [37]. In fact, we believe that it is better that people is aware of some applications that might be potentially harmful, even if it turns out not to be so, rather than missing some real threats.

Challenging Modern AV vendors Based on the recent reports by Virustotal [39], there is an increase in the number of anti-malware developers that resort to machine learning approaches for malware detection. However, the main focus of these products appears to be on desktop malware, especially Windows PE malware. Based on the available public information, there are just a few evidences of two anti-malware developers that use machine learning approaches for Android malware detection, namely Symantec[18] and TrustLook [38]. Their products are installed by more than 10 million users. While it is not clear to us how these products use machine learning, we considered them as two candidates for comparison with `IntelliAV`. To provide a sound comparison, in addition to the Symantec and Trustlook products, we selected three other Android anti-malware products, i.e., AVG, Avast, and Qihoo 360, that are the most popular among Android users as they have been installed more than 100 million times.[2] We compared the performances of `IntelliAV` on the test dataset (see Section 3.2) with the ones attained by these five popular Android anti-malware As shown in Figure 4, `IntelliAV` performs slightly better than two of the products used for comparison, while it outperforms the other three. As we gathered the label assigned by anti-malware products to the test samples at most two months after they are first seen in VirusTotal, the comparison could be more interesting if we had the label given to samples at the time they are first seen in the wild. As an additional check, we performed a comparison in detection performance by considering a set of very recent malware reported by four vendors, namely Check Point, Fortinet, Lookout, and Google (see Table 4). The good performances of `IntelliAV` compared to the ones of other products, shows that

[2] http://www.androidrank.org/

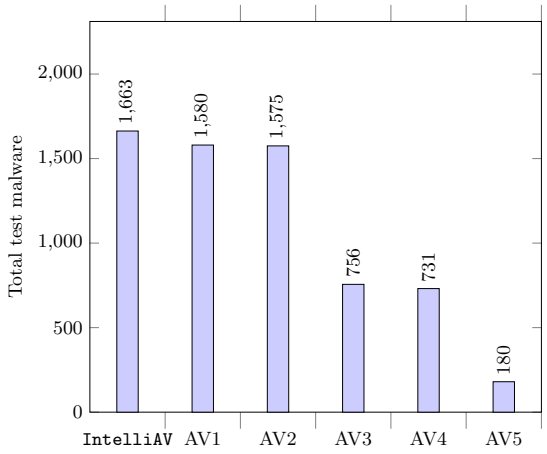


Fig. 4: Comparison between the detection rate of IntelliAV with top five Android anti-malware. We didn't put the name of vendors as we don't aim to rank other anti-malware products.

the selected lightweight features and training procedure allows attaining very good performances, especially if we consider that 21 of the considered samples were first seen before 2017, so it is expected that they can be detected by anti-malware tools either by signatures, or by the generalization capability provided by their machine learning engines. If we have a close look at the two misclassified samples by `IntelliAV` (Table 4), we can see that the associated risk scores are quite close to the decision threshold that we set at training time. The main reasons for the misclassification of these two samples can be related to the use of the `runtime.exec` API to run some shell commands, and to the presence of native-code that is used to hide some of their malicious behaviors.

3.3 `IntelliAV` Overhead on Device

To better understand the efficiency of `IntelliAV`, we show the time consumption for feature extraction as well as classification of some medium/large-sized applications on three devices with different technical specifications. The three mobile devices used for the reported experiments are a Samsung Galaxy S6 Edge (released in April, 2015), a Huawei P8 Lite (released in May, 2015), and an LG D280 L65 (released in June, 2014), which respectively have 3GB, 2GB, and 1GB of RAM. In addition, we computed the time required on the Android Emulator that is dispatched along with Android Studio. The time is simply computed by specifying a timer before starting the feature extraction procedure, that stops when the features from both the manifest and the dex code are extracted. For classification, the reported time refers to the interval between the feature vector is passed to the model, to the production of the risk score. The time required to load the model is negligible, and so we are not reporting it for the sake of clarity.

Table 4: Point to point comparison of `IntelliAV` and three anti-malware vendors on some recent and well-known malware reported by Check Point, Fortinet, Lookout, and Google from January to April of 2017. These samples were evaluated on an Android emulator. The `time` column refers to the required time for performing both feature extraction and classification on the emulator.

#	MD5	Size	`IntelliAV` Unseen	time(s)	Risk Score	2017 check AV5	AV1	AV3	VT 1st check
	Reported malware by Checkpoint [32, 31, 33]								
1	60806c69e0f4643609dcdf127c8e7ef5	66 KB	✓	0.38	83% (✓)	(✓)	(✓)	(✓)	2016-01 (02/56)
2	fcbb243294bb87b039f113352a8db158	12.4 MB	✓	0.40	37% (✗)	(✗)	(✓)	(✗)	2016-03 (19/55)
3	4e91ff9ac7e3e349b5b9fe36fb505cb4	48 KB	✓	0.37	93% (✓)	(✓)	(✓)	(✓)	2016-03 (13/57)
4	944850ee0b7fc774c055a2233478bb0f	842 KB	✓	0.51	98% (✓)	(✗)	(✗)	(✗)	2014-02 (00/48)
5	629da296cba945662e436bbe10a5cdaa	3.7 MB	✓	0.69	92% (✓)	(✓)	(✓)	(✗)	2014-07 (13/51)
6	1aac52b7d55f4c1c03c85ed067bf69d9	3.5 MB	✓	0.75	94% (✓)	(✓)	(✓)	(✓)	2013-11 (23/47)
7	379ec59048488fdb74376c4ffa00d1be	2.2 MB	✓	0.57	79% (✓)	(✓)	(✓)	(✓)	2015-09 (26/56)
8	d5f5480a7b29ffd51c718b63d1ffa165	9.1 MB	✓	0.82	89% (✓)	(✗)	(✓)	(✗)	2015-12 (03/55)
9	4d904a24f8f4c52726eb340b329731dd	13.2 MB	✓	0.95	72% (✓)	(✗)	(✓)	(✗)	2014-08 (11/51)
10	59b62f8bc982b31d5e0411c74dbe0897	2.5 MB	✓	0.45	83% (✓)	(✓)	(✓)	(✓)	2016-01 (31/55)
11	9ed38abb335f0101f55ad20bde8468dc	8.1 MB	✓	0.77	67% (✓)	(✗)	(✓)	(✗)	2016-02 (16/55)
12	4a3a7b03c0d0460ed8c5beff5c20683c	575 KB	✓	0.42	68% (✓)	(✓)	(✓)	(✓)	2017-03 (00/55)
13	660638f5212ef61891090200c354a6d5	32.7 MB	✓	1.13	96% (✓)	(✓)	(✓)	(✗)	2016-07 (13/55)
14	f48122e9f4333ba3bb77fac869043420	349 KB	✓	0.40	81% (✓)	(✓)	(✓)	(✗)	2015-09 (04/57)
15	0e987ba8da76f93e8e541150d08e2045	12.8 MB	✓	0.98	88% (✓)	(✗)	(✓)	(✗)	2017-03 (07/60)
16	51c328fccf1a8b4925054136ccdb1cda	874 KB	✓	0.44	83% (✓)	(✗)	(✗)	(✓)	2014-08 (05/53)
17	3f188b9aa8f739ee0ed572992a21b118	1.57 MB	✓	0.48	89% (✓)	(✓)	(✓)	(✓)	2014-04 (24/51)
18	7fff1e78089eb387b6adfa595385b2c9	13.4 MB	✓	0.52	63% (✓)	(✓)	(✓)	(✓)	2015-03 (02/57)
19	2b83bd1d97eb911e9d53765edb5ea79e	2.3 MB	✓	0.43	77% (✓)	(✗)	(✗)	(✓)	2017-01 (16/58)
20	48ff097022ea7886b53f80edf2972033	1.3 MB	✓	0.47	63% (✓)	(✗)	(✗)	(✓)	2017-03 (28/59)
21	a3836485ecac78f576e1753269350824	14.6 MB	✓	0.84	38% (✗)	(✗)	(✗)	(✗)	2016-12 (14/57)
22	a4e75471dbf0bb0d3ec26d854cb7fe12	14.1 MB	✓	0.72	62% (✓)	(✗)	(✗)	(✓)	2016-12 (10/56)
23	7253e0a13d2d1db1547e9984a4ce7abd	1.3 MB	✓	0.57	63% (✓)	(✗)	(✗)	(✗)	2017-03 (26/59)
	Reported malware by Fortinet [22, 23, 20, 21]								
24	193058ae838161ee4735a9172ebc25ec	1.4 MB	✓	0.56	89% (✓)	(✗)	(✓)	(✗)	2017-01 (05/24)
25	f479f2a29354a8b889cb529a2ee2c1b4	1.1 MB	✓	0.35	61% (✓)	(✗)	(✗)	(✓)	2017-03 (12/59)
26	cad94ac28640c771b1d2de5e786dc352	776 KB	✓	0.37	96% (✓)	(✗)	(✓)	(✓)	2016-11 (20/56)
27	40507254b8156de817f02c0ed111e99f	0.2 MB	✓	0.37	83% (✓)	(✓)	(✓)	(✓)	2016-11 (08/57)
	Reported malware by Lookout and Google [27, 24]								
28	cc9517aafb58279091ac17533293edc1	57 KB	✓	0.63	89% (✓)	(✗)	(✗)	(✗)	2016-02 (00/53)
29	7c3ad8fec33465fed6563bbfabb5b13d	252 KB	✓	0.37	82% (✓)	(✗)	(✗)	(✓)	2017-04 (03/60)
30	3a69bfbe5bc83c4df938177e05cd7c7c	19 KB	✓	0.36	81% (✓)	(✗)	(✗)	(✗)	2017-04 (01/60)
					$\frac{28}{30}$ ☺	$\frac{12}{30}$	$\frac{19}{30}$	$\frac{16}{30}$	

Table 5: Overhead of `IntelliAV` on different devices for very large applications. F.E. refers to feature extraction time and C. refers to classification time. The number in parenthesis shows the RAM size of the device.

		Galaxy S6 Edge Marshmallow (3GB)		Huawei P8 Lite Lollipop (2GB)		LG D280 L65 KitKat (1GB)		Emulator Marshmallow (1.5GB)	
App	APK Size (MB)	F.E. (s)	C. (s)	F.E. (s)	C. (s)	F.E. (s)	C. (s)	F.E. (s)	C. (s)
Google Trips	8.19	0.67	0.003	0.82	0.005	3.86	0.012	0.43	0.001
LinkedIn Pulse	12.9	1.28	0.003	1.14	0.005	4.40	0.012	0.55	0.001
Stack Exchange	8.15	1.27	0.004	1.27	0.006	5.13	0.014	0.60	0.001
Telegram	12.41	1.36	0.005	1.74	0.007	5.52	0.016	0.69	0.002
WhatsApp	27.97	2.29	0.006	3.22	0.008	12.91	0.018	1.10	0.002
SoundCloud	33.14	2.67	0.006	2.84	0.008	11.83	0.018	1.14	0.002
Spotify	34.65	2.51	0.006	3.03	0.008	13.67	0.018	1.22	0.002
Twitter	31.77	4.53	0.004	5.95	0.006	24.46	0.016	2.26	0.002
LinkedIn	40.39	4.67	0.004	4.69	0.006	16.73	0.016	2.40	0.001
Airbnb	54.34	8.24	0.006	8.79	0.008	35.71	0.018	4.23	0.002
Messenger	59.43	5.85	0.011	7.94	0.013	19.13	0.028	3.35	0.004
Uber	37.26	6.66	0.004	7.64	0.006	43.88	0.016	4.29	0.002
Average	30.05	3.50	0.005	4.08	0.007	16.43	0.016	1.86	0.002

As shown in Table 5, the time required to analyze even large applications is less than 10 seconds, which makes `IntelliAV` practical and reasonable as the number of installed applications on each device is not too large. The classification part is performed in native code, that provides a fast execution. As expected, it can be noted that the largest fraction of the time required by `IntelliAV` is spent for feature extraction, especially for the extraction of the API features. This figure is even worse in the case an application is made up of multiple dex files, because the extraction of API features is much slower. For example, the Uber app is made up of 10 dex files, so that searching for a specific API requires much more time compared to applications having just one dex file.

4 Limitations

As far as `IntelliAV` is based on static analysis, it inherits some of the well-known limitations of static analysis approaches. For instance, we didn't address reflection and dynamic code loading techniques that are used to hide the malicious code. Moreover, in the proposed implementation, `IntelliAV` doesn't handle those malware samples that use JavaScript to perform an attack. However, the most common evasion techniques are based on obfuscation of names, and the use of downloaders that download the malicious payload at run-time. The reported test results show that `IntelliAV` is robust against these common obfuscation techniques as it doesn't rely on features extracted from strings or names of classes or methods. In addition, as far as `IntelliAV` runs on the device, it can track all downloaded and installed apps, scanning them on the fly. Consequently, it can be more robust compared to off-device systems. In addition, we are aware that the system can be a victim of evasion techniques against the learning approach, such as mimicry attacks that let an attacker inject some data to the app

so that its features resembles the ones of benign apps [12]. Consequently, more methodological and experimental analysis will be needed to make a quantitative evaluation of the robustness of `IntelliAV` in an adversarial environment, to provide the system with the required hardening. Nonetheless, we believe that the good performances of the proposed system is a good starting point for further development. Moreover, employing the multiple classifier system approach, considering a larger number of semantic features, as well as performing a fine grained classifier parameter tuning, can provide a degree of robustness against adversarial attacks against the machine learning engine.

5 Related works

At present, a large number of papers addressed the topic of detecting Android malware by proposing different systems. The proposed approaches can be divided into two main categories, namely offline malware detection, and on-device malware detection. While a complete overview is outside of the scope of this paper, and we suggest the interested reader to resort to one of the good survey that have been recently published (e.g., the recent taxonomy proposed in [34]), we provide here some of the more closely related papers that rely on static analysis technique. We omit reviewing the malware classification systems based on dynamic analysis [17, 5, 15] as they have their own benefits and pitfalls. Moreover, as we are dealing with an on-device tool, it is not officially possible that a process access system calls of other process without root privileges, which makes the dynamic analysis approaches almost impractical on the end user device.

Offline Malware Detection. Usually, offline testing has no hard computational constraints, thanks to the availability of computational power compared to the one available on mobile devices, thus allowing for sophisticated application analysis. Hence, a number of approaches have been proposed to construct complex models capable of detecting malware with a very high accuracy. Some of the prominent approaches that focus on building a model and offline testing of Android applications by static analysis techniques are briefly summarized. MudFlow [11], AppAudit [40], and DroidSIFT [42] rely on information flow analysis[9], while DroidMiner [41], and MaMaDroid [29] use API sequences to detect malware. The use of complex features such as information flows and API sequences, makes these approach more difficult to be carried out on the device. Lighter approaches have been proposed, such as Drebin [8], DroidAPIMiner [1], and DroidSieve [36] that make use of meta-data as well as syntactic features, and that allow for their porting to on-device applications.

On-Device Malware Detection. Based on the best of our knowledge, there are a few approaches in the research community that used machine learning for on-device malware detection, and none of them is publicly available for performance comparison. One that of the most cited research works on this topic is Drebin, and while the paper shows some screenshots of the UI, the application itself is not available. Among the commercial Android anti-malware tools, two of them claim to use machine learning techniques, as evaluated and reported

in Section 3.2, but the real use of machine learning by these tools is blurred. Finally, Qualcomm recently announced the development of a machine learning tool for on-device mobile phone security, but the details of the system, as well as its performances are not available [26].

As an overall comparison with the previous approaches, we believe that `IntelliAV` provides a first practical example of an on-device anti-malware solution for Android systems, completely based on machine learning techniques, that can move a step toward having an advanced security tool on mobile devices.

6 Conclusions and future work

In this paper, we introduced a practical learning-based anti-malware tool for Android systems on top of TensorFlow, in which both the efficiency and the effectiveness are considered. We showed that through the careful selection of a set of lightweight features, and a solid training phase comprising both a robust classification model, and a representative set of training samples, an efficient and effective tool can be deployed on Android mobile device. Our tool will be freely available so that it can help the end user to provide easy protection on the device, as well as allowing researchers to better explore the idea of having intelligent security systems on mobile devices. As a future plan, we aim to address the limitations of IntelliAV, to improve its robustness against attacks on the machine learning engine, while keeping the efficiency intact.

7 Acknowledgement

We appreciate VirusTotal's collaboration for providing us the access to a large set of Android applications.

References

1. Aafer, Y., Du, W., Yin, H.: DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android, pp. 86–103. Springer International Publishing, Cham (2013)
2. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: Tensorflow: A system for large-scale machine learning. pp. 265–283. OSDI, USENIX Association (2016)
3. Ahmadi, M., Biggio, B., Arzt, S., Ariu, D., Giacinto, G.: Detecting misuse of google cloud messaging in android badware. pp. 103–112. SPSM (2016)
4. Ahmadi, M., Ulyanov, D., Semenov, S., Trofimov, M., Giacinto, G.: Novel feature extraction, selection and fusion for effective malware family classification. pp. 183–194. CODASPY (2016)
5. Amos, B., Turner, H., White, J.: Applying machine learning classifiers to dynamic android malware detection at scale. In: 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC). pp. 1666–1671 (July 2013)

6. Andronio, N., Zanero, S., Maggi, F.: Heldroid: Dissecting and detecting mobile ransomware. In: Proceedings of the 18th International Symposium on Research in Attacks, Intrusions, and Defenses - Volume 9404. pp. 382–404. RAID 2015, Springer-Verlag New York, Inc., New York, NY, USA (2015)
7. Aresu, M., Ariu, D., Ahmadi, M., Maiorca, D., Giacinto, G.: Clustering android malware families by http traffic. pp. 128–135. MALWARE (2015)
8. Arp, D., Spreitzenbarth, M., Hubner, M., Gascon, H., Rieck, K.: Drebin: Effective and explainable detection of android malware in your pocket. In: NDSS (2014)
9. Arzt, S., Rasthofer, S., Fritz, C., Bodden, E., Bartel, A., Klein, J., Le Traon, Y., Octeau, D., McDaniel, P.: Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In: Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 259–269. PLDI '14, ACM, New York, NY, USA (2014)
10. AV-TEST: Security report 2015/16 (2017), https://goo.gl/FepOGQ
11. Avdiienko, V., Kuznetsov, K., Gorla, A., Zeller, A., Arzt, S., Rasthofer, S., Bodden, E.: Mining apps for abnormal usage of sensitive data. pp. 426–436. ICSE (2015)
12. Biggio, B., Corona, I., Maiorca, D., Nelson, B., Šrndić, N., Laskov, P., Giacinto, G., Roli, F.: Evasion Attacks against Machine Learning at Test Time, pp. 387–402 (2013)
13. Bishop, C.M.: Pattern Recognition and Machine Learning (Information Science and Statistics). 1 edn. (2007)
14. Breiman, L.: Random forests. Mach. Learn. 45(1), 5–32 (Oct 2001)
15. 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. pp. 15–26. SPSM '11, ACM, New York, NY, USA (2011)
16. Colthurst, T., Sculley, D., Hendry, G., Nado, Z.: Tensorforest: Scalable random forests on tensorflow. In: Machine Learning Systems Workshop at NIPS (2016)
17. Dash, S.K., Suarez-Tangil, G., Khan, S., Tam, K., Ahmadi, M., Kinder, J., Cavallaro, L.: Droidscribe: Classifying android malware based on runtime behavior. In: 2016 IEEE Security and Privacy Workshops (SPW). pp. 252–261 (May 2016)
18. eweek: Symantec adds deep learning to anti-malware tools to detect zero-days (January 2016), http://www.eweek.com/security/symantec-adds-deep-learning-to-anti-malware-tools-to-detect
19. Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D.: Do we need hundreds of classifiers to solve real world classification problems? J. Mach. Learn. Res. 15(1), 3133–3181 (Jan 2014)
20. Fortinet: Android locker malware uses google cloud messaging service (January 2017), https://blog.fortinet.com/2017/01/16/android-locker-malware-uses-google-cloud-messaging-service
21. Fortinet: Deep analysis of android rootnik malware using advanced anti-debug and anti-hook (January 2017), https://goo.gl/dq5w8R
22. Fortinet: Teardown of a recent variant of android/ztorg (part 1) (March 2017), https://blog.fortinet.com/2017/03/15/teardown-of-a-recent-variant-of-android-ztorg-part-1
23. Fortinet: Teardown of android/ztorg (part 2) (March 2017), http://blog.fortinet.com/2017/03/08/teardown-of-android-ztorg-part-2
24. Google: An investigation of chrysaor malware on android (April 2017), https://android-developers.googleblog.com/2017/04/an-investigation-of-chrysaor-malware-on.html

25. IDC: Smartphone os market share, q2 2016 (2016), http://www.idc.com/promo/smartphone-market-share/os
26. Islam, N., Das, S., Chen, Y.: On-device mobile phone security exploits machine learning. IEEE Pervasive Computing 16(2), 92–96 (2017)
27. Lookout: Pegasus for android (April 2017), https://info.lookout.com/rs/051-ESQ-475/images/lookout-pegasus-android-technical-analysis.pdf
28. Maiorca, D., Mercaldo, F., Giacinto, G., Visaggio, A., Martinelli, F.: R-packdroid: Api package-based characterization and detection of mobile ransomware. In: ACM Symposium on Applied Computing (2017)
29. Mariconti, E., Onwuzurike, L., Andriotis, P., De Cristofaro, E., Ross, G., Stringhini, G.: MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models. In: ISOC Network and Distributed Systems Security Symposiym (NDSS). San Diego, CA (2017)
30. McAfee: Mobile threat report (2016), https://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2016.pdf
31. Point, C.: Charger malware calls and raises the risk on google play (January 2017), http://blog.checkpoint.com/2017/01/24/charger-malware/
32. Point, C.: Preinstalled malware targeting mobile users (March 2017), http://blog.checkpoint.com/2017/03/10/preinstalled-malware-targeting-mobile-users/
33. Point, C.: A whale of a tale: Hummingbad returns (January 2017), http://blog.checkpoint.com/2017/01/23/hummingbad-returns/
34. Sadeghi, A., Bagheri, H., Garcia, J., s. Malek: A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software. IEEE Transactions on Software Engineering PP(99), 1–1 (2016)
35. f secure: Mobile threat report q1 2014 (2014), https://www.f-secure.com/documents/996508/1030743/Mobile_Threat_Report_Q1_2014.pdf
36. Suarez-Tangil, G., Dash, S.K., Ahmadi, M., Kinder, J., Giacinto, G., Cavallaro, L.: Droidsieve: Fast and accurate classification of obfuscated android malware. In: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy. pp. 309–320. CODASPY '17 (2017)
37. Taylor, V.F., Martinovic, I.: Securank: Starving permission-hungry apps using contextual permission analysis. In: Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices. pp. 43–52. SPSM '16, ACM, New York, NY, USA (2016)
38. Trustlook: Trustlook ai (March 2017), https://www.trustlook.com/
39. VirusTotal: Virustotal blog (March 2017), http://blog.virustotal.com/2017_03_01_archive.html
40. Xia, M., Gong, L., Lyu, Y., Qi, Z., Liu, X.: Effective real-time android application auditing. In: IEEE Symposium on Security and Privacy. pp. 899–914. IEEE Computer Society (2015)
41. Yang, C., Xu, Z., Gu, G., Yegneswaran, V., Porras, P.: DroidMiner: Automated Mining and Characterization of Fine-grained Malicious Behaviors in Android Applications, pp. 163–182 (2014)
42. Zhang, M., Duan, Y., Yin, H., Zhao, Z.: Semantics-aware android malware classification using weighted contextual api dependency graphs. pp. 1105–1116. CCS, New York, NY, USA (2014)