IntelliAV: Building an Effective On-Device Android Malware Detector

Mansour Ahmadi
Northeastern University

Angelo Sotgiu
Università degli studi di Cagliari

Giorgio Giacinto
Università degli studi di Cagliari

IntelliAV: Building an Effective On-Device Android Malware Detector

Mansour Ahmadi[1,2]*, Angelo Sotgiu[1], and Giorgio Giacinto[1]

[1] University of Cagliari, Italy
[2] Northeastern University, USA

Abstract. The importance of employing machine learning for malware detection has become explicit to the security community. Several anti-malware vendors have claimed and advertised the application of machine learning in their products in which the inference phase is performed on servers and high-performance machines, but the feasibility of such approaches on mobile devices with limited computational resources has not yet been assessed by the research community, vendors still being skeptical. In this paper, we aim to show the practicality of devising a learning-based anti-malware on Android mobile devices, first. Furthermore, we aim to demonstrate the significance of such a tool to cease new and evasive malware that can not easily be caught by signature-based or offline learning-based security tools. To this end, we first propose the extraction of a set of lightweight yet powerful features from Android applications. Then, we embed these features in a vector space to build an effective as well as efficient model. Hence, the model can perform the inference on the device for detecting potentially harmful 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, named `IntelliAV` [3][4], provides more satisfying performances than the popular major anti-malware products. Moreover, we evaluate the robustness of `IntelliAV` against common obfuscation techniques where most of the anti-malware solutions get affected.

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

1 Introduction

Android is the most popular platform for mobile devices, with almost 85% of the market share in the first quarter of 2017 [2]. More interestingly, Android is now the most popular operating system connected to the Internet after overtaking Windows [3]. The majority of the security issues affecting Android systems

* Corresponding author: m.ahmadi@northeastern.edu

[3] This paper is the extended version of `IntelliAV` conference paper [1]

[4] IntelliAV application is available online: http://www.intelliav.com

can be attributed to third-party applications (app) rather than to the Android OS itself. According to a report in 2017 from G DATA (a security vendor), a new instance of Android malware emerges nearly every 10 seconds [4]. Besides, another recent report from McAfee shows that the malware infection rate of Android mobile devices is soaring [5]. We believe that this huge amount of mobile malware needs to be timely detected, possibly by intelligent tools running on the device, because it has been shown that malware can bypass offline security checks (e.g., by relying on so-called droppers, that load the malicious payload after being activated), and live in the wild for a while. In fact, to the best of our knowledge, even the most recent versions of Android anti-malware products are still not satisfactory to cope with most of the novel and obfuscated malware.

Today, machine learning is one of the most successful helper techniques for Android malware detection and classification [6,7,8,9]. The recent advances in the optimization of machine learning tools that can execute on mobile platforms, such as Android [10], increase the possibility of empowering security applications with machine learning tools. Despite the improvement in processor and RAM of mobile devices, the development of any mobile anti-malware product should take into account the processing time 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 a learning-based Android anti-malware product does not need to be necessarily sophisticated, as it has been shown that Android malware seems to perform simpler tasks than the desktop counterparts [11]. All the reasons above stimulate a proposal for an on-device machine learning solution to detect potential malicious applications.

Contribution. Accordingly, in this paper, we introduce an intelligent risk-based anti-malware approach for Android devices, called `IntelliAV`, which is built on top of the open-source and multi-platform TensorFlow library. In fact, we illustrate the feasibility and the advantages of such an approach on the device by leveraging on the existing literature, and, in particular, on previous works by the authors, to tackle the deficiencies of existing Android anti-malware products, mostly based on pattern matching techniques, as well as offline learning-based approaches. As far as we know, the performances of learning-based malware detection systems for Android have been only tested off-device, i.e., with the availability of computational power and memory space well beyond the capabilities of mobile devices. More specifically, the three main contributions of `IntelliAV` are as follows:

(i) We design a system relying on a trained model on a sizeable set of applications. The model is carefully constructed to be both effective and efficient by wisely selecting a set of lightweight, discriminative, and effective features. Moreover, the model is precisely validated by tuning its parameters to be practical for the capabilities of Android devices. We then show how the crafted model can be embedded in the `IntelliAV` application, and can readily be deployed on Android devices.

(ii) The performances of `IntelliAV` are evaluated through a cross-validation process, where our system can achieve 92% detection rate, that is compa-

rable to other off-device learning-based Android malware detection while relying on a comparatively small set of features. Moreover, as a supplementary experiment, IntelliAV is analyzed on two different sets of more recent malware samples with respect to the samples included in the training specimens. Interestingly, IntelliAV can achieve 96% detection rate from an independent test by a 3rd party organization, and it obtains 72% detection rate —on a set of randomly gathered apps by us— that is higher than the performances of the top 5 commercial Android anti-malware products.

(*iii*) To understand the robustness of IntelliAV, we evaluate the impact of two common evasion techniques, i.e., dropper and obfuscation, on the proposed system. For droppers, interestingly, we show how IntelliAV can stop them on the device while offline machine learning techniques would fail to detect. For the second examination, we prove the power of IntelliAV on identifying heavily obfuscated malware, which perfectly shows how machine learning can add to the cost evasion.

Paper organization. The rest of the paper is organized as follows: First, we highlight the importance of an on-device risk-based malware detection (§2) and review the existing works on this area (§3). Next, we reveal the structure of IntelliAV, motivating the choice of features, and the procedure followed to construct the model (§4). We then assess the approach (§5) and remark the limitations of IntelliAV (§6). Finally, we conclude our paper discussing future directions of IntelliAV (§7).

2 Motivation of having a Risk-based On-Device Approach

There are several incentives for the security community to develop a risk-based mobile security approach, possibly based on machine learning, as well as performing real-time on the mobile device in addition to employing it on servers.

2.1 Malware in Google Play store

There have been various reports on the practicability of by-passing Google security mechanisms by malware coders. Consequently, malware keeps sneaking on the Google Play store and remains accessible to users until a security vendor/researcher reports it to Google as harmful. For instance, the Check Point security firm described a zero-day mobile ransomware found on Google Play in January 2017 [12]. This malware was dubbed as a *Charger* application and more than a million users downloaded this app. Another report from the same vendor indicates the case of new variants of the famous Android malware family HummingBad [13] on Google Play. Another specimen of malware that could infiltrate Google Play is a packed malware that sends fraudulent premium SMS messages and charges users for fake services without their knowledge [14]. More detail on vetting these samples is available in Section 5.2.

2.2 Install Malware from unknown sources

Third-party app stores are popular among mobile users because they usually offer applications at great discounts, as well as users from specific countries can find there those applications whose access is restricted by the Google Play store according to each country's rules. Nevertheless, security checks on the third-party stores are not as effective as those available in the Google Play store. Therefore, third-party markets are a breeding ground for mobile malware propagation, and this fact sometimes leads people to download spoofed versions of well-known applications. A large number of reports on malicious applications found in these stores have been published during the past few years. In addition to the third-party markets, direct download from unknown websites is another source of infection. It is quite often that users can be eluded 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 contamination is though phishing SMS messages that contain links to malicious applications. Recent reports by Lookout and Google [15,16] show how a targeted attack malware, namely *Chrysaor*, which is presumed of infecting devices via a phishing attack, could remain undetected for a few years. More detail on vetting these samples is available in Section 5.2.

2.3 Distribute malware in supply chain

Users of mobile devices are frequently recommended by information security experts to be cautious when downloading applications from untrusted sources or even when they install not very popular apps from Google Play. However, there are fewer warnings for the users on the reliability of safety claims of new devices, especially when they might be shipped with pre-installed malware. Despite the concerns of manufacturers on securing the 'supply chain', it can be compromised by attackers, for the number of people and companies involved in the supply chain of the components. There is a recent report that shows how devious hackers spread malware on Android devices somewhere along the supply chain before the user obtained the phone [17]. More detail on vetting these samples is available in Section 5.2.

2.4 Easy to evade pattern matching

Almost all of the major Android anti-malware kernels operate, to the extent of our knowledge, by matching signatures or patterns. These types of scanning engines let both malware variants of known families, as well as zero-day malware threaten our own devices. There are claims by a few Android anti-malware vendors on the use of heuristic approaches like machine learning in their products. However, no evidence regarding the implementation of the kernel including machine learning on the device is available. Hence, to find out further, we evaluate them on very recent malware samples, as well as on a few obfuscated malware samples. More detail is available in Section 5.2 and Section 5.4.

2.5 Droppers dodge offline vetting tools

A typical kind of evasion technique employed by any malware to evade off-device vetting mechanisms is by developing droppers, and Android malware makes no exception [18]. Droppers don't directly perform malicious activities, and they are designed to install some sort of malicious application to a device. Therefore, detecting droppers is not a straightforward task even by advanced off-device machine learning techniques as a dropper itself usually exhibits a few standard behaviors that are common in legitimate applications as well, so that it does not obviously reveal malicious actions. We analyze a few representative malware samples using this attack vector in Section 5.3.

2.6 Conclusion

All of the above observations encourage to empower Android devices with a machine-learning anti-malware engine, either as a complement to pattern matching techniques, or as an independent complete solution.

3 Related Work

The problem of detecting Android malware through machine learning approaches has been explored quite a lot since 2010 [19]. While an entire overview is outside of the scope of this paper, we suggest the interested reader resorting to one of the recent surveys on this subject, e.g., the taxonomy proposed in [20]. Additionally, it is out of the scope of the paper a review of dynamic malware analysis approaches [21,6,22] as dynamic analysis has its specific advantages and pitfalls. For instance, we are dealing with an on-device tool, and it is not officially possible that a process accesses system calls of another process without root privileges, which makes the dynamic analysis approaches almost impractical on the end user device. Hence, we provide here some of the more closely relevant papers that rely on static analysis technique. The existing methods are classified into two distinct levels, namely off-device and on-device malware detection.

3.1 Off-Device Malware Detection

Offline testing usually has no hard computational restrictions, thanks to the availability of computational power compared to the one available on mobile devices. Some of the prominent malware detection models are MudFlow [23], AppAudit [24], and DroidSIFT [25] relying on information flow analysis[26], while DroidMiner [8], and MaMaDroid [27] are based on API sequences. Although this allows constructing complex models capable of detecting malware with a very high accuracy, the use of elaborate features such as information flows and API sequences makes these approach harder to be carried out on the device. Lighter approaches, such as Drebin [7], DroidAPIMiner [28], and DroidSieve [9] that make use of meta-data, as well as syntactic features, allow for their porting to on-device applications.

3.2 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. Drebin [7] is one of them, which has been cited the most on this topic. While the paper shows some screenshots of the UI, the application itself is not available. Besides, while the proposed system is for both workstation and mobile devices, the actual needs for a learning-based malware detection engine on the device was not specified. Among the commercial Android anti-malware tools, two of them claim to use machine learning techniques, as reported in Section 5.2, but the extent to which machine learning is used in these tools is not disclosed. 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 publicly available [29].

Year	Method	Detection		Feature
		On-Device	**Available**	
2014	DroidAPIMiner [28]	–	–	API,PKG,PAR
2014	DroidMiner [8]	–	–	CG,API SEQ
2014	Drebin [7]	✓	–	PER,STR,API,INT
2014	DroidSIFT [25]	–	–	API-F
2015	AppAudit [24]	–	✓	API-F
2015	MudFlow [23]	–	✓	API-F
2017	MaMaDroid [27]	–	✓	CG,API SEQ
2017	DroidSieve [9]	–	–	API,PER,INT,PN,STR,ST
2017	Qualcomm [29]	✓	–	Not Available
Ours	**IntelliAV**	✓	✓	PER,INT,API,ST

API: Application Programming Interface, API-F: Information Flow between APIs, INT: Intents, CG: Call Graph, SEQ: Sequence, PER: Requested Permissions, STR: Embedded strings, ST: Statistical features, PN: Package names

Table 1: Android Malware detection techniques based on machine learning techniques and static analysis. All of the systems that are mostly based on API, API-F, and API SEQ would fail against reflection. IntelliAV is the only on-device system that is available in the market.

3.3 Summary

As an overall comparison (see Table 1) with the previous approaches, we believe that `IntelliAV` provides for an effective and practical on-device anti-malware solution for Android systems, totally based on machine learning techniques.

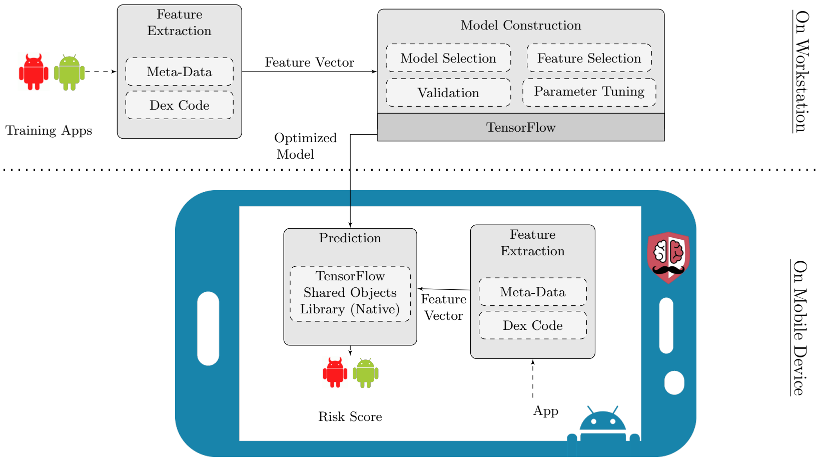


Fig. 1: Overview of `IntelliAV`.

`IntelliAV` is available online, and can move a step toward having an advanced security tool on mobile devices.

4 System Design

Figure 1 illustrates the architecture of the proposed `IntelliAV` system. Its design consists of the following two main phases:

(*i*) **Training the model offline.** As a first step, we resort to a conventional computing environment to build a classification model. To conduct the training phase, we gathered a relatively large number of applications (§5.1). 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 (§4.1). Next, a classification function is learned by associating each feature vector to the type of applications it has been extracted from, i.e., malware or goodware (§4.2).

(*ii*) **Model operation on the device.** As the second phase, the optimized model is embedded in the `IntelliAV` Android application so that `IntelliAV` can produce a risk score for each downloaded APK or installed apps on the device, without resorting to any external interaction, e.g., with cloud services (§4.3).

4.1 Feature Engineering

The feature extraction step is the core phase for any learning-based system. To address Android malware detection, the security community has suggested various types of features as discussed in Section3. However, some sets of features related to primary Android functions, like permissions, APIs, and Intents, usually allow achieving reasonable detection results, with the aim to alert for the presence of probably harmful applications [7,9]. Extracting this set of features is also feasible on mobile devices because they do not need deep static analysis, thus expecting a limited computational effort.

With the aim of extracting a set of efficient and effective features for our proposed system, we resort to the following four categories of characteristics. Three of them are derived from the '*manifest*' of Android applications, namely Permissions, Intent Filters, components statistics, and the fourth one is extracted from the DEX code, notably APIs. A typical Android app has a single DEX file, and most of the standard malware detection approaches focus on this file. Because of the Dalvik Executable specification, the total number of methods that can be referenced within a single DEX file is limited to 65,536 -including Android framework methods, library methods, and methods in the code. This limitation can be handled by multidex configuration if an application extends to more than 65K methods. Therefore, multidex can be a desirable technique for Android malware to split the payload into multiple DEX files to make the detection harder [30]. For this reason, to be more comprehensive in the terms of extracting the characteristics of apps, `IntelliAV` has to parse all of the DEX files of an application although it makes the feature extraction process somewhat slower (see section 5.5).

To construct the feature vector, we consider all the permissions and the intent-filters inquired by the samples included in the training set. Besides, 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 somewhat the extent 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 goodware, except for the case of malware that is built by repackaging benign applications. Moreover, we manually select a set of 179 APIs as features and include them in the feature vector. The selected APIs are those that reveal some particular characteristics of the 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 (i.e., a technique for hiding APIs) 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. 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. By contrast, intent-filters are integer-valued features, as they represent the number of times an intent-filter is declared in the manifest. Considering this count for intent-filter features makes them more meaningful rather than simply

Table 2: 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

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.).

In total, the feature vector contains 4000 features. To avoid overfitting and make IntelliAV faster on the mobile device, we decided to reduce the number of features by selecting the most discriminative ones through a feature selection procedure (see Section 4.2). During the feature selection step, we consider four thresholds, namely, 25%, 50%, and 75% of the top features, as well as the whole feature set. Interestingly, a model containing 25% (i.e., 1000) of the top features can achieve the best result while uses fewer features. Therefore, the final assortment consists of 322 features related to permissions, 503 features linked to Intent filters, four statistical features from components (e.g., count of activities), and 171 features associated with API usage (see Table 2).

4.2 Model Construction

As we mentioned earlier, the model needs to be trained on a workstation, and it is not necessary to conduct the training phase on the device because it has to be performed once we need to update the model according to the evolution of malware. In other words, in contrast to pattern matching techniques, our system does not require frequent updates by the end user, which diminishes the amount of computation and network traffic transferred. 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 from new malware families [31].

To discriminate malware from goodware, we need to rely on binary classification techniques. Over the past years, a large number of classification techniques have been proposed by the scientific community, and the choice of the most suitable classifier for a given task is often guided by preceding experience in different domains, as well as by trial-and-error procedures. However, among all of the existing classifiers, Random Forest classifier [32] have shown high performances in a variety of tasks [33]. The Random Forests algorithm belongs to the ensemble learning methods in which many decision trees are constructed at training time by randomly selecting the features used by each decision tree. Eventually, the algorithm outputs the class of an instance at the testing time based on the combined decision of the tress. One of the main reasons that Random forest models often achieve better results compared to others is that it is an ensemble classifier, which means it helps to reduce the variance in performances of the decision trees. So, the final model exhibits low bias and low variance, which makes the model more robust against both the *underfitting* and *overfitting* problems [34].

We build `IntelliAV` on the top of TensorFlow [35] library to be able to train our model offline, as well as to test it on Android devices. TensorFlow is an open source library for machine learning, which was published by Google in November 2015. A TensorFlow model is highly portable as it supports the vast majority of platforms such as Linux, Mac OS, Windows, and mobile computing platforms like 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. To the best of our knowledge, `IntelliAV` is the first anti-malware tool that has proposed employing TensorFlow. More specifically, we employ an implementation of Random Forests in TensorFlow, called TensorForest [36].

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 new unknown specimens, we employ feature selection to reduce the size of the feature set by excluding irrelevant and noisy features. In particular, as done in [37], 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.

4.3 On-Device Testing

As we mentioned before, TensorFlow facilitates the task of using machine learning models on mobile devices. So, we embed the trained model, obtained according to the procedure described in Section 4.2, in `IntelliAV`. It is important to acknowledge that as far as the model is resident on the device and not in the cloud, the network traffic generated by `IntelliAV` is zero, compared to approaches that communicate with the cloud by sending a hash/APK to obtain the risk associated to an app.

The size of TensorFlow models depends on the complexity of the model. For instance, if the number of trees in the TensorForest model increases, consequently the size of the model increases as well. The size of `IntelliAV` model that we transferred to the device is about 14.1MB. Having said that, when it is embedded into the APK and because the model is originally saved in a textual format, the final size of the model after compression in the APK becomes just 3.3MB.

Whenever an application needs to be tested, first, `IntelliAV` loads the model, then extracts the features from the application on the device, and finally, it feeds the model by the extracted features to acquire the application's risk type. The model provides a likelihood value between 0 and 1, denoting the degree of maliciousness of the application. Then, the likelihood is thresholded to three types of risks to make it more understandable for the end user. We empirically provide the following guideline for interpreting the likelihood. If the likelihood is lower than 0.4, the risk is low and we suggest the user consider the application as safe. If the likelihood is between 0.4 and 0.6, then the application should be removed if the application is not so popular in Google Play or if the user is not sure about the trustworthiness of the source application. Finally, the application has to be removed with high confidence if the likelihood is higher than 0.6. These thresholds have been set after testing the system on a set containing different applications. It is worth to mention that the analyses of apps that identify potential harmful apps may yield both false positives and false negatives.

As shown in figure 2, the two main capabilities of `IntelliAV` are the verification of the risk of all the installed applications on the device (*Quick Scan*), as well as analyzing the risk of downloaded/dropped APKs (*Custom Scan*). The latter ability is necessary as it helps the user to check the risk of the application before installation. One of the prominent use cases of this ability is stopping droppers on the device, while offline machine learning systems can be simply lured by the use of droppers in malware (see Section 5.3). For the *Custom Scan*, `IntelliAV` needs the `READ_EXTERNAL_STORAGE` permission to access the contents of the application's package on the external storage. For the *Quick Scan*, `IntelliAV` has to read `base.apk` file in a sub-directory with a name corresponding to the package name, which is located in the */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` needs neither any permission nor a rooted device to evaluate the installed applications.

5 Experimental Analysis

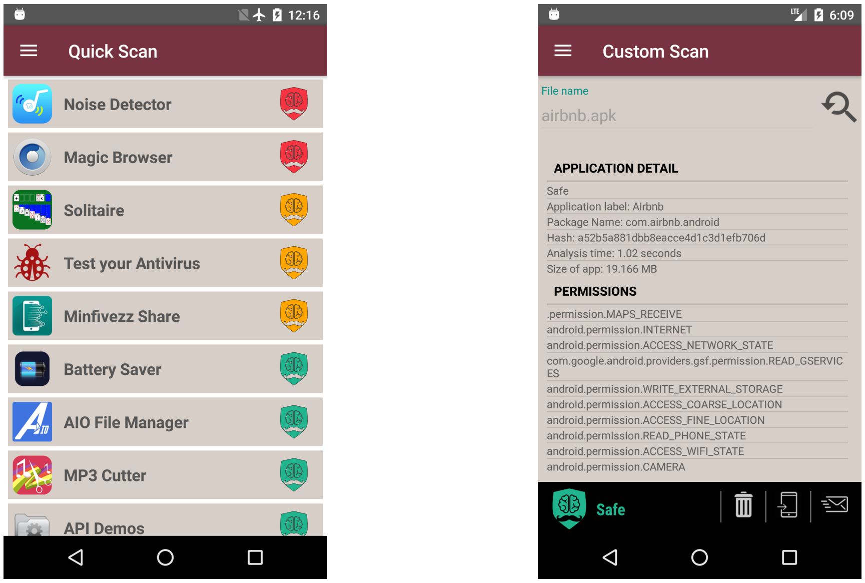
In this section, we address the following research questions:

- Is `IntelliAV` able to detect new and unseen malware (§5.2 and §5.2)?
- Are the performances of `IntelliAV` comparable to the ones of popular mobile anti-malware products, although `IntelliAV` is completely based on machine learning techniques (§5.2)?
- How much is `IntelliAV` robust against prevalent evasion techniques like droppers (§5.3) and obfuscation (§5.4)?
- Which is the overhead of `IntelliAV` on real devices (§5.5)?

Before addressing these questions, we explain the experimental settings and the data used for building our model (§5.1).

5.1 Experimental Setup

To train `IntelliAV`, we have collected 19,722 applications from VirusTotal[38], divided into 10,058 benign and 9,664 malicious applications. We considered the diversity of malicious applications, by including samples belonging to different categories, such as Adware, Ransomware [39], and GCM malware [40]. In addition, in order to cover the variety of malware characteristics throughout its evolution during time, we also take into account samples from the first versions



(a) Scan installed applications

APPLICATION DETAIL

Safe

Application label: Airbnb

Package Name: com.airbnb.android

Hash: a52b5a881dbb8eacce4d1c3d1efb706d

Analysis time: 1.02 seconds

Size of app: 19.166 MB

PERMISSIONS

.permission.MAPS_RECEIVE

android.permission.INTERNET

android.permission.ACCESS_NETWORK_STATE

com.google.android.providers.gsf.permission.READ_GSERVICES

android.permission.WRITE_EXTERNAL_STORAGE

android.permission.ACCESS_COARSE_LOCATION

android.permission.ACCESS_FINE_LOCATION

android.permission.READ_PHONE_STATE

android.permission.ACCESS_WIFI_STATE

android.permission.CAMERA

(b) Scan an APK

Fig. 2: `IntelliAV` abilities.

of Android malware, namely January 2011, until recent versions as of December 2016. We consider an application as being malicious if it was tagged as being malware by at least 10 of the tools used by VirusTotal.

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. The two metrics that have been used for assessing the performance of our approach were 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, which can help to find the best decision threshold by considering the trade-off.

5.2 Results

To adequately evaluate the effectiveness of `IntelliAV`, the following scenarios were considered.

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

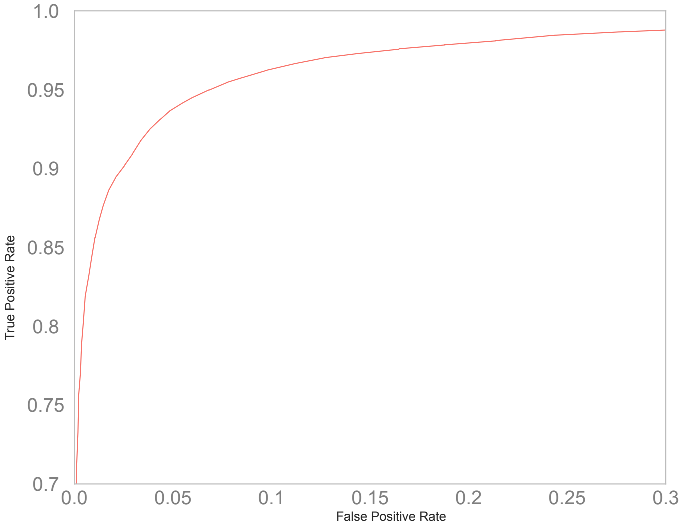


Fig. 3: ROC curve of TensorForest (5-fold cross validation). FPR and TPR are respectively 4.2% and 92.5%.

find the optimum-tuned parameters to be used for building the final model as a trade-off between correct detection and generalization capability. Consequently, we evaluate `IntelliAV` on the set of applications described in Section 5.1 through a 5-fold cross-validation procedure, 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 created 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 repeat the procedure by running the Random Forest algorithm multiple times to obtain the most appropriate parameters while keeping the size of the model lower. 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 Although cross-validation is important for setting the model parameters, we do not desire the model exhibit high misclassification on the training samples. Therefore, as an additional investigation to verify the effectiveness of the tuned parameters based on the cross-validation procedure explained in Section 5.2, we test the model, which was constructed on all 19.7K samples, on the same set, namely on all the samples used for training. Table 3 shows the classification results on the training set. It shows that

Table 3: Training on the set of samples explained in Section 5.1 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

IntelliAV misclassifies just a few training samples. This confirms 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.

Evaluation on new Malware To realize the sustainability [41] of our model during a period of 3 months after the model is deployed, we examine the performance 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 consider 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 that were newer than the samples in the training set, and thus they were not part of the training set.

As shown in Table 4, the detection rate on the test set is 71.96%, which is quite satisfying if compared with the performances of other Android anti-malware solutions that are available in the market (See Section 5.2). 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 occasionally. 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 remarking that our classification of false positives is linked to the ground-truth classification provided by VirusTotal at the time of evaluation. 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. We expect in 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 [42]. In fact, we believe that people should be 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 There is a growth in the number of anti-malware vendors that resort to machine learning approaches [38]. However, the

Table 4: Training on the set of samples described in Section 5.1, 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

foremost focus of these products appears to be on desktop malware, especially Windows PE malware. Based on the publicly available information, there are just a few pieces of evidence of two anti-malware vendors that use machine learning for Android malware detection, namely Symantec[43] and TrustLook [44], and their products are installed by more than 10 million users. Despite the lack of clearance of using machine learning by these tools to us, we consider them as two candidates for comparison with `IntelliAV`. To provide a rational comparison, in addition to the Symantec and Trustlook products, we choose three other Android anti-malware products, i.e., AVG, Avast, and Qihoo 360, that are the most popular security tools among Android users as they have been installed more than 100 million times.[5]

We compared the performances of `IntelliAV` on the test dataset (see Section 5.2) 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 a supplementary experiment, we carried out a measurement in detection performance by considering a set of top and very recent malware threats reported by four vendors, namely Check Point, Fortinet, Lookout, and Google (see Table 5). The considerable performances of `IntelliAV` compared to the ones of other products, confirms the effectiveness of the selected lightweight features and the training procedure, especially if we consider that 21 of the analyzed 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 5), 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.

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

Table 5: 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.

			`IntelliAV`			2017 check			
#	MD5	Size	Unseen	time(s)	Risk Score	AV1	AV3	AV5	VT 1st check
	Reported malware by Checkpoint [17,12,13,14]								
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)
24	84a62599a40e36be2180485245e8123f	5.7 MB	✓	0.17	81% (✓)	(✗)	(✗)	(✗)	2017-05 (02/62)
25	c2132651331a77d41a323fefa71bfbd0	5.1 MB	✓	0.29	87% (✓)	(✗)	(✗)	(✗)	2017-05 (04/59)
	Reported malware by Fortinet [45,46,47,48]								
26	193058ae838161ee4735a9172ebc25ec	1.4 MB	✓	0.56	89% (✓)	(✓)	(✗)	(✗)	2017-01 (05/24)
27	f479f2a29354a8b889cb529a2ee2c1b4	1.1 MB	✓	0.35	61% (✓)	(✗)	(✓)	(✗)	2017-03 (12/59)
28	cad94ac28640c771b1d2de5e786dc352	776 KB	✓	0.37	96% (✓)	(✓)	(✓)	(✗)	2016-11 (20/56)
29	40507254b8156de817f02c0ed111e99f	0.2 MB	✓	0.37	83% (✓)	(✓)	(✓)	(✓)	2016-11 (08/57)
	Reported malware by Lookout and Google [15,16]								
30	cc9517aafb58279091ac17533293edc1	57 KB	✓	0.63	89% (✓)	(✗)	(✗)	(✗)	2016-02 (00/53)
31	7c3ad8fec33465fed6563bbfabb5b13d	252 KB	✓	0.37	82% (✓)	(✗)	(✓)	(✗)	2017-04 (03/60)
32	3a69bfbe5bc83c4df938177e05cd7c7c	19 KB	✓	0.36	81% (✓)	(✗)	(✗)	(✗)	2017-04 (01/60)
					$\frac{30}{32}$ ☺	$\frac{19}{32}$	$\frac{16}{32}$	$\frac{12}{32}$	

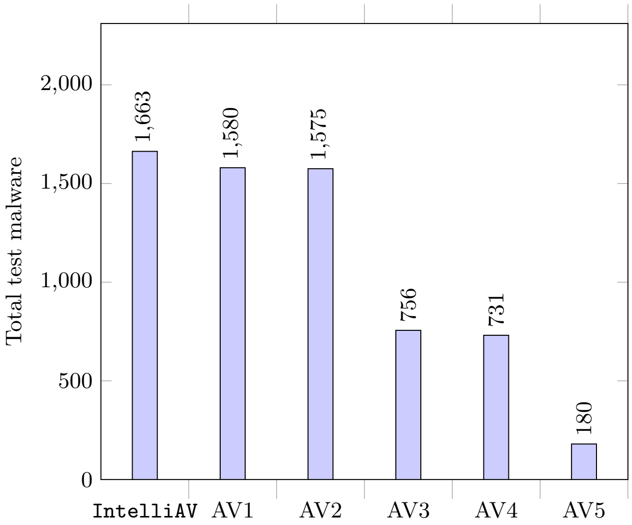


Fig. 4: Comparison between the detection rate of IntelliAV with top five Android anti-malware. We didn't put the name of vendors as we don't aim to rank other anti-malware products.

Independent Test by a Third Party To avoid any bias regarding testing `IntelliAV`, we requested a third-party independent anti-malware testing organization to assess our tool. The test performed on a test suit involving 500 most common and recent Android malware, which all have been first seen by VirusTotal in 2017 —more recent than the samples in our training test. Interestingly, as is shown in Figure 5, `IntelliAV` can successfully classify 478 of the samples as malware, i.e., 96% malware detection rate. Except for `Fakeapp` family, `IntelliAV` achieves very good results on the rest of malware families. As specimens in `Fakeapp` family regularly abuse legitimate application by repackaging them, the presence of more benign characteristics in such malware seems to be the main reason of misclassification.

To measure the false positive rate, `IntelliAV` was tested on 50 benign applications that have been downloaded more than 10,000 times from Google Play, as well as were not tagged as malware by any anti-malware product. Two out of the 50 apps were misclassified by `IntelliAV`. However, one of the two misclassified samples has been removed from Google Play recently. Although the main reason for elimination is not clear, the application surely violates some of the Google Play policies and `IntelliAV` seems to detect such a kind of violation perfectly.

5.3 Stop Droppers on Device

Dynamic code loading techniques are employed by both the benign and malicious applications [49,50]. There are different techniques to dynamically load a piece of

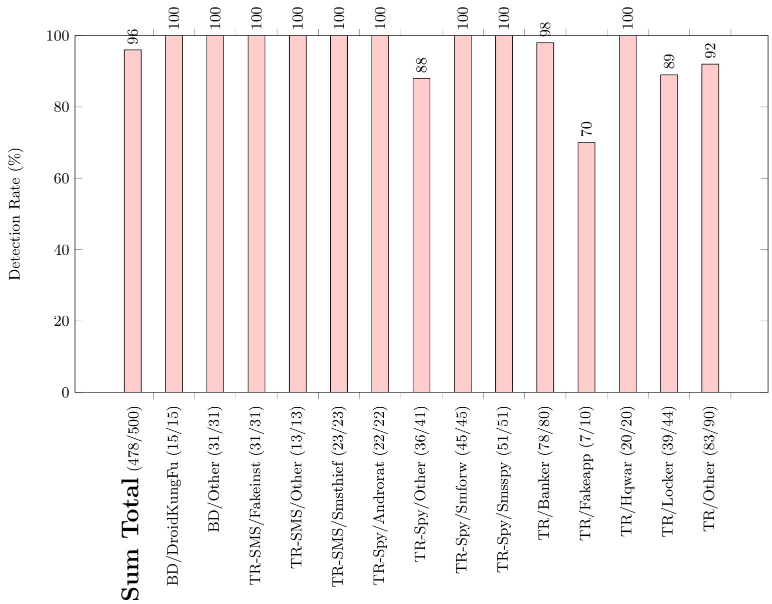
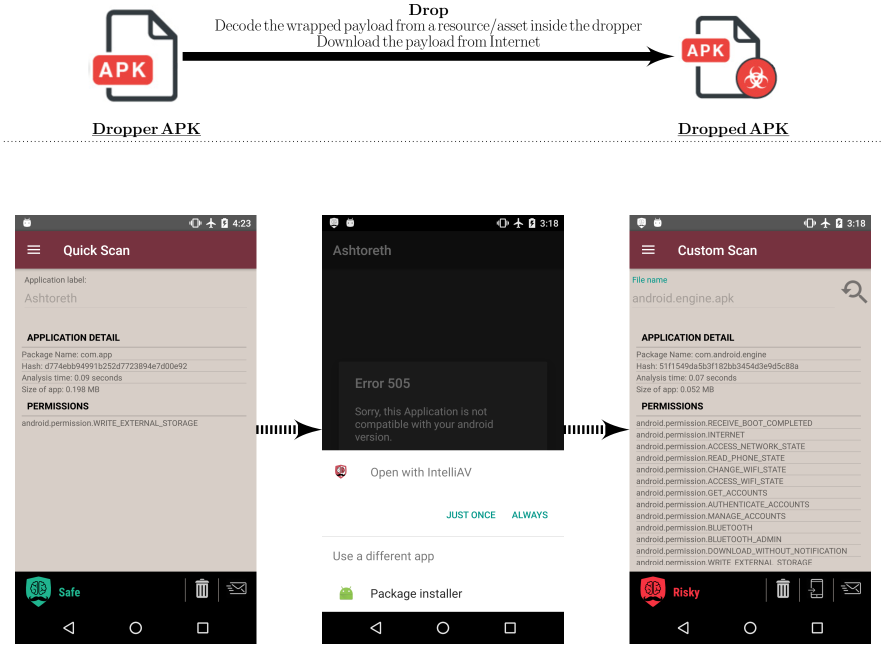


Fig. 5: Results of the independent test on IntelliAV in July 2017. BD and TR refer to backdoor and trojan respectively. The detection rate is separated based on the malware families as well.

code in Android applications, and the technique that turns to be more popular among malware developers has not been yet clearly evaluated. However, there are many reports by anti-malware vendors [51,52] that designates APK installation technique, also known as droppers, as one of the most popular approaches by which attackers deliver a malware. A dropper is a malware installer that surreptitiously carries any kind of malicious software so they can be executed on the compromised machine. They do not carry any malicious activities by themselves, but they just open a way for the attack by downloading/decompressing and then installing the core malware payload onto a target machine without detection. Depending on the way they retrieve the payload, i.e., by downloading or decompressing, they might be called `Downloader Trojans` or `Wrappers`.

The overall technique employed by droppers is summarized in Figure 6. The upper part of the figure is the overall scheme of the attack technique, and the bottom part is showing how `IntelliAV` can stop this attack on the device when the malware has to reveal its full capabilities while offline machine learning techniques would fail. To dissect this attack vector in more detail, we evaluate a very recent dropper Android malware, which was reported in July 2017 by Trendmicro [51]. This malware misleads users to install a secondary app by showing a fake message. When the dropper app[6] is launched, an error pops

[6] Dropper APK MD5: d774ebb94991b252d7723894e7d00e92


Drop
Decode the wrapped payload from a resource/asset inside the dropper
Download the payload from Internet
APK
APK
Dropper APK
Dropped APK

Quick Scan

Application label: Ashtoreth

APPLICATION DETAIL

Package Name: com.app
Hash: d774ebb94991b252d7723894e7d00e92
Analysis time: 0.09 seconds
Size of app: 0.198 MB

PERMISSIONS

android.permission.WRITE_EXTERNAL_STORAGE

Safe

Ashtoreth

Error 505

Sorry, this Application is not compatible with your android version.

Open with IntelliAV

JUST ONCE ALWAYS

Use a different app

Package installer

Custom Scan

File name: android.engine.apk

APPLICATION DETAIL

Package Name: com.android.engine
Hash: 51f1549da5b3f182bb3454d3e9d5c88a
Analysis time: 0.07 seconds
Size of app: 0.052 MB

PERMISSIONS

android.permission.RECEIVE_BOOT_COMPLETED
android.permission.INTERNET
android.permission.ACCESS_NETWORK_STATE
android.permission.READ_PHONE_STATE
android.permission.CHANGE_WIFI_STATE
android.permission.ACCESS_WIFI_STATE
android.permission.GET_ACCOUNTS
android.permission.AUTHENTICATE_ACCOUNTS
android.permission.MANAGE_ACCOUNTS
android.permission.BLUETOOTH
android.permission.BLUETOOTH_ADMIN
android.permission.DOWNLOAD_WITHOUT_NOTIFICATION
android.permission.WRITE_EXTERNAL_STORAGE

Risky

Fig. 6: Detecting dropper attacks by IntelliAV on the device. Offline machine learning techniques would fail to detect this kind of attack. The example is a very recent Android malware, which was reported in July 2017 by Trendmicro [51]

out that: "*Sorry, the application is not compatible with your android version*". Meanwhile, it decodes a string from the APK resource file, which is actually a malicious APK[7], and requests the user to install it. As the dropper does not contain malicious activities and just needs one permission to drop the secondary APK, `IntelliAV` does not produce any risk alert. However, before the user is infected by the dropped APK, they can request `IntelliAV` to scan it before installation (i.e., by its custom scan capability), which results in the detection of the real malicious app.

It is obvious that `IntelliAV` can generalize this detection and prevention approach to any malware that uses APK installation technique. Another example is an Android malware[8], reported by an ESET researcher [52], that asks just for the storage permission to save the the secondary loaded malicious APK[9]. After the first APK executes, it immediately drops the secondary APK, embedded in an image located in an APK asset, and requests the user to install it. In another

[7] Dropped APK MD5: 51f1549da5b3f182bb3454d3e9d5c88a
[8] Dropper APK MD5: 1f41ba0781d51751971ee705dfa307d2
[9] Dropped APK MD5: 90886b56372f5191a78a20dcb3f9fe6e

report by the same vendor, a dropper[10] is a kind of logic bomb [53] and drops a malicious apk[11] twenty minutes after the first run [54].

More advanced offline machine learning techniques relying on static analysis, even those deeply look inside an APK to find an embedded malware like DroidSieve [9], cannot model all of the feasible dropping techniques because the secondary app can be concealed by custom methods. Although offline approaches relying on dynamic analysis [55] might be able to detect this kind of attack if they are system-centric and designed perfectly to disclose logic bombs, there has not been any research specifically to address this issue.

5.4 Robustness against Obfuscation

Obfuscation techniques [56,57] are widely used by both the benign and malicious Android applications to protect themselves against reverse engineering. Although it is clear that there is no bulletproof approach, a formidable detection system should not be influenced by obfuscation dramatically. Hence, to evaluate the resiliency of `IntelliAV`, we employ a recently released obfuscation framework, namely AVPASS [58], that has been shown to be able to evade almost all of the commercial Android anti-malware products. AVPASS uses different common obfuscation techniques to encrypt and remove/add features from/to an application. These techniques are summarized in Table 6.

Table 6: Obfuscation techniques used for evaluating IntelliAV. These techniques preserve the original functionality of the application. The techniques like removing permissions are not considered here as they destroy the functionality of the application.

Obfuscation Type	Affect on IntelliAV
1 Insert dummy bytes between all instructions	✗
2 Insert benign permissions to AnadroidManifest.xml	Possible
3 Image file obfuscation	✗
4 Change variables name	✗
5 Modify user-defined class/package name	✗
6 Insert API between two existing APIs	Possible
7 Encrypt strings	✗
8 API hiding by Java Reflection	Possible
9 Resource XML obfuscation	✗

Among the aforementioned obfuscation techniques, three of them might directly affect on `IntelliAV`, and they are 1) adding benign permissions, 2) hiding APIs by Java reflection, and 3) inserting APIs. To realize how much the techniques in Table 6 can undermine `IntelliAV`, we randomly select ten small size

[10] Dropper APK MD5: add7572e94b54bb408d34c8c19972b0c

[11] Dropped APK MD5: 243028fe2a1a9358187d6e694343fcd9

Android malware (see Table 6). The main reason of being small size is that employing all of the obfuscation techniques needs a huge processing time and large sized apps cannot be done on a laptop.

We investigate `IntelliAV` in two different scenarios, i.e., testing on the ten original malware as well as testing on the obfuscated variant of the malware. Then, we use the identical approach for testing the samples against those anti-malware products that are accessible in Virus Total. Interestingly, `IntelliAV` is completely robust against all of the obfuscation techniques in Table 6. While the obfuscated variant can simply evade most of the anti-viruses, evasion from `IntelliAV` needs more elaborate efforts. Among the anti-malware that have detected the obfuscated variant, three of them performs very well. However, it seems that they fingerprint AVPASS footsteps because they detect even a benign app, obfuscated by AVPASS, as a malware. Moreover, they use the same family name from all the obfuscated malware by AVPASS. In this case, if an attacker applies a custom obfuscation/evasion technique, they can readily dodge not carefully crafted anti-malware products. It is worth to mention that none of the antimalware, which have claimed they are based on machine learning, are able to detect more than five samples. This evaluation again arises the attention toward the lack of either use of machine learning or generalization of the detection model.

Table 7: Point to point comparison of `IntelliAV` and VirusTotal on some randomly selected malware as well as obfuscated variant of the malware. The percentage numbers show the risk score calculated by IntelliAV. The `VT` column shows the number of anti-malware vendors that are able to detect a malware before and after obfuscation. Almost all of the anti-malware products have difficulty to detect the obfuscated malware. There are three anti-malware that able to detect most of obfuscated malware. However, it seems that they fingerprint AVPASS as they detect even obfuscated benign apps by AVPASS as malware. The test has been performed on VT in early August, 2017.

		`Original Malware`		Obfuscated Variant	
#	MD5	IntelliAV	VT	IntelliAV	VT
1	60806c69e0f4643609dcdf127c8e7ef5	82% (✓)	31/59	84% (✓)	4/60
2	4e91ff9ac7e3e349b5b9fe36fb505cb4	87% (✓)	34/60	75% (✓)	4/61
3	26b8a840b4cc15d0b05533705d21854b	100% (✓)	43/60	98% (✓)	3/61
4	19723d489712cff0ea2907f142937b8b	100% (✓)	40/62	84% (✓)	5/61
5	4760ac11bf995d26c0486cf3b73f5f9a	100% (✓)	43/61	92% (✓)	6/60
6	0085ae9415d115a6bde1e9ff72b6dc7f	100% (✓)	43/59	98% (✓)	3/60
7	9eebbc312fa1cdc891e151f367e6bd9d	84% (✓)	31/61	81% (✓)	8/59
8	711d00941aa78e912051d644f84748be	100% (✓)	31/62	98% (✓)	7/61
9	cabdc04298dbf6bce3857328d24c96dd	100% (✓)	38/61	84% (✓)	6/58
10	a0113b794e69d7ca1d603aadf977b8a7	100% (✓)	46/60	92% (✓)	7/61

5.5 IntelliAV Overhead on Device

Despite the belief that running a detection model on mobile devices is computationally infeasible, we acknowledge the efficiency of IntelliAV by exposing the time consumption for the feature extraction as well as the classification phases. We select some popular medium/large-sized applications and analyze them by IntelliAV 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 time in which the feature vector is passed to the model, and the time of 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.

As shown in Table 8, 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, and the computational power of mobile devices is increasing even on cheap models. 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. Extraction of API features is even much slower in the case an application is made up of multiple dex files. For instance, 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.

6 Limitations

As far as IntelliAV is based on static analysis, it inherits some of the well-known limitations of static analysis approaches. For instance, although we partially addressed dynamic code loading techniques, the more complicated one, which hides the malicious behavior in the native code, might affect our system. Moreover, in the current proposed implementation, IntelliAV cannot detect the malicious actions executed by JavaScript. 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 resemble the ones of benign apps [59,60,61]. 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 performance of the proposed system against a few obfuscation techniques (e.g., adding benign permissions/APIs as well as hiding APIs) is a good starting point for further

Table 8: 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		Huawei P8 Lite		LG D280 L65		Emulator	
		Marshmallow (3GB)		Lollipop (2GB)		KitKat (1GB)		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

development. Moreover, employing the multiple classifier systems approaches, 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.

7 Conclusions and future work

In this work, we investigated the practicality of building a learning-based anti-malware tool for the devices running Android platform. To consider both the effectiveness and the efficiency of the tool, we emphasize on a careful selection of a set of lightweight features, as well as a solid training phase. The reported results show that `IntelliAV` is robust against common obfuscation techniques. In addition, as far as `IntelliAV` runs on the device, it can track and scan all downloaded, dropped, and installed apps on the fly, which makes it more robust compared to off-device systems. 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.

8 Acknowledgment

This work has been supported by the project “PISDAS” jointly carried out with Innovery SpA, funded by the Regional Administration of Sardinia under the program “Pacchetti Integrati di Agevolazione Industria Artigianato e Servizi ai sensi della deliberazione G.R. n. 46 del 31/10/2013”, proposal 165. We appreciate

VirusTotal's collaboration for providing us the access to a large set of Android applications. As a note, some of the antivirus vendors existing in VirusTotal platform do not always exactly perform the same as their public commercial versions. However, it is the best available option that can be used for a large scale testing. Moreover, we would like to acknowledge the third party antivirus testing organization for evaluating `IntelliAV` for free.

References

1. M. Ahmadi, A. Sotgiu, G. Giacinto, Intelliav: Toward the feasibility of building intelligent anti-malware on android devices, in: A. Holzinger, P. Kieseberg, A. M. Tjoa, E. Weippl (Eds.), Machine Learning and Knowledge Extraction, Springer International Publishing, Cham, 2017, pp. 137–154.
2. IDC, Smartphone os market share, q1 2017 (2017).
URL `http://www.idc.com/promo/smartphone-market-share/os`
3. StatCounter, Android overtakes windows for first time (2017).
URL `https://goo.gl/oh7Gzy`
4. GDATA, 8,400 new android malware samples every day (2017).
URL `https://www.gdatasoftware.com/blog/2017/04/29712-8-400-new-android-malware-samples-every-day`
5. McAfee, Mobile threat report (2016).
URL `https://www.mcafee.com/us/resources/reports/rp-mobile-threat-report-2016.pdf`
6. B. Amos, H. Turner, J. White, Applying machine learning classifiers to dynamic android malware detection at scale, in: 2013 9th International Wireless Communications and Mobile Computing Conference (IWCMC), 2013, pp. 1666–1671.
7. D. Arp, M. Spreitzenbarth, M. Hubner, H. Gascon, K. Rieck, Drebin: Effective and explainable detection of android malware in your pocket., in: NDSS, 2014.
8. C. Yang, Z. Xu, G. Gu, V. Yegneswaran, P. Porras, DroidMiner: Automated Mining and Characterization of Fine-grained Malicious Behaviors in Android Applications, 2014, pp. 163–182.
9. G. Suarez-Tangil, S. K. Dash, M. Ahmadi, J. Kinder, G. Giacinto, L. Cavallaro, Droidsieve: Fast and accurate classification of obfuscated android malware, in: Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy, CODASPY '17, 2017, pp. 309–320.
10. Google, Tensorflow mobile (2017).
URL `https://www.tensorflow.org/mobile/`
11. M. Aresu, D. Ariu, M. Ahmadi, D. Maiorca, G. Giacinto, Clustering android malware families by http traffic, MALWARE, 2015, pp. 128–135.
12. CheckPoint, Charger malware calls and raises the risk on google play (January 2017).
URL `http://blog.checkpoint.com/2017/01/24/charger-malware/`
13. CheckPoint, A whale of a tale: Hummingbad returns (January 2017).
URL `http://blog.checkpoint.com/2017/01/23/hummingbad-returns/`
14. CheckPoint, Expensivewall: A dangerous packed malware on google play that will hit your wallet (2017).
URL `https://blog.checkpoint.com/2017/09/14/expensivewall-dangerous-packed-malware-google-play-will-hit-wallet/`

15. Lookout, Pegasus for android (April 2017).
URL https://info.lookout.com/rs/051-ESQ-475/images/lookout-pegasus-android-technical-analysis.pdf
16. Google, An investigation of chrysaor malware on android (April 2017).
URL https://android-developers.googleblog.com/2017/04/an-investigation-of-chrysaor-malware-on.html
17. CheckPoint, Preinstalled malware targeting mobile users (March 2017).
URL http://blog.checkpoint.com/2017/03/10/preinstalled-malware-targeting-mobile-users/
18. Lookout, What are droppers & what can they do to me? (2015).
URL https://blog.lookout.com/security-101-droppers
19. A. Shabtai, Y. Fledel, Y. Elovici, Automated static code analysis for classifying android applications using machine learning, in: 2010 International Conference on Computational Intelligence and Security, 2010, pp. 329–333. doi:10.1109/CIS.2010.77.
20. A. Sadeghi, H. Bagheri, J. Garcia, s. Malek, A taxonomy and qualitative comparison of program analysis techniques for security assessment of android software, IEEE Transactions on Software Engineering PP (99) (2016) 1–1.
21. S. K. Dash, G. Suarez-Tangil, S. Khan, K. Tam, M. Ahmadi, J. Kinder, L. Cavallaro, Droidscribe: Classifying android malware based on runtime behavior, in: 2016 IEEE Security and Privacy Workshops (SPW), 2016, pp. 252–261.
22. I. Burguera, U. Zurutuza, S. Nadjm-Tehrani, Crowdroid: Behavior-based malware detection system for android, in: Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM '11, ACM, New York, NY, USA, 2011, pp. 15–26.
23. V. Avdiienko, K. Kuznetsov, A. Gorla, A. Zeller, S. Arzt, S. Rasthofer, E. Bodden, Mining apps for abnormal usage of sensitive data, ICSE, 2015, pp. 426–436.
24. M. Xia, L. Gong, Y. Lyu, Z. Qi, X. Liu, Effective real-time android application auditing, in: IEEE Symposium on Security and Privacy, IEEE Computer Society, 2015, pp. 899–914.
25. M. Zhang, Y. Duan, H. Yin, Z. Zhao, Semantics-aware android malware classification using weighted contextual api dependency graphs, CCS, New York, NY, USA, 2014, pp. 1105–1116.
26. S. Arzt, S. Rasthofer, C. Fritz, E. Bodden, A. Bartel, J. Klein, Y. Le Traon, D. Octeau, P. McDaniel, 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, PLDI '14, ACM, New York, NY, USA, 2014, pp. 259–269.
27. E. Mariconti, L. Onwuzurike, P. Andriotis, E. De Cristofaro, G. Ross, G. Stringhini, MaMaDroid: Detecting Android Malware by Building Markov Chains of Behavioral Models, in: ISOC Network and Distributed Systems Security Symposiym (NDSS), San Diego, CA, 2017.
28. Y. Aafer, W. Du, H. Yin, DroidAPIMiner: Mining API-Level Features for Robust Malware Detection in Android, Springer International Publishing, Cham, 2013, pp. 86–103.
29. N. Islam, S. Das, Y. Chen, On-device mobile phone security exploits machine learning, IEEE Pervasive Computing 16 (2) (2017) 92–96.
30. Symantec, Five ways android malware is becoming more resilient (2016).
URL https://www.symantec.com/connect/blogs/five-ways-android-malware-becoming-more-resilient

31. AV-TEST, Security report 2015/16 (2017).
URL https://goo.gl/FepOGQ
32. L. Breiman, Random forests, Mach. Learn. 45 (1) (2001) 5–32.
33. M. Fernández-Delgado, E. Cernadas, S. Barro, D. Amorim, Do we need hundreds of classifiers to solve real world classification problems?, J. Mach. Learn. Res. 15 (1) (2014) 3133–3181.
34. C. M. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics), 1st Edition, 2007.
35. M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, Tensorflow: A system for large-scale machine learning, OSDI, USENIX Association, 2016, pp. 265–283.
36. T. Colthurst, D. Sculley, G. Hendry, Z. Nado, Tensorforest: Scalable random forests on tensorflow, in: Machine Learning Systems Workshop at NIPS, 2016.
37. M. Ahmadi, D. Ulyanov, S. Semenov, M. Trofimov, G. Giacinto, Novel feature extraction, selection and fusion for effective malware family classification, CODASPY, 2016, pp. 183–194.
38. VirusTotal, Virustotal blog (March 2017).
URL http://blog.virustotal.com/2017_03_01_archive.html
39. N. Andronio, S. Zanero, F. Maggi, Heldroid: Dissecting and detecting mobile ransomware, in: Proceedings of the 18th International Symposium on Research in Attacks, Intrusions, and Defenses - Volume 9404, RAID 2015, Springer-Verlag New York, Inc., New York, NY, USA, 2015, pp. 382–404.
40. M. Ahmadi, B. Biggio, S. Arzt, D. Ariu, G. Giacinto, Detecting misuse of google cloud messaging in android badware, SPSM, 2016, pp. 103–112.
41. R. Jordaney, K. Sharad, S. K. Dash, Z. Wang, D. Papini, I. Nouretdinov, L. Cavallaro, Transcend: Detecting concept drift in malware classification models, in: 26th USENIX Security Symposium (USENIX Security 17), USENIX Association, Vancouver, BC, 2017, pp. 625–642.
URL https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/jordaney
42. V. F. Taylor, I. Martinovic, Securank: Starving permission-hungry apps using contextual permission analysis, in: Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices, SPSM '16, ACM, New York, NY, USA, 2016, pp. 43–52.
43. eweek, Symantec adds deep learning to anti-malware tools to detect zero-days (January 2016).
URL https://goo.gl/YYmxxs
44. Trustlook, Trustlook ai (March 2017).
URL https://www.trustlook.com/
45. Fortinet, Teardown of a recent variant of android/ztorg (part 1) (March 2017).
URL https://blog.fortinet.com/2017/03/15/teardown-of-a-recent-variant-of-android-ztorg-part-1
46. Fortinet, Teardown of android/ztorg (part 2) (March 2017).
URL http://blog.fortinet.com/2017/03/08/teardown-of-android-ztorg-part-2
47. Fortinet, Android locker malware uses google cloud messaging service (January 2017).
URL https://blog.fortinet.com/2017/01/16/android-locker-malware-uses-google-cloud-messaging-service

48. Fortinet, Deep analysis of android rootnik malware using advanced anti-debug and anti-hook (January 2017).
URL https://goo.gl/dq5w8R

49. S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, G. Vigna, Execute This! Analyzing Unsafe and Malicious Dynamic Code Loading in Android Applications, in: Proceedings of the ISOC Network and Distributed System Security Symposium (NDSS), San Diego, CA, 2014.

50. Y. Zhauniarovich, M. Ahmad, O. Gadyatskaya, B. Crispo, F. Massacci, Stadyna: Addressing the problem of dynamic code updates in the security analysis of android applications, in: Proceedings of the 5th ACM Conference on Data and Application Security and Privacy, CODASPY '15, ACM, New York, NY, USA, 2015, pp. 37–48. doi:10.1145/2699026.2699105.
URL http://doi.acm.org/10.1145/2699026.2699105

51. Trendmicro, Android backdoor ghostctrl can silently record your audio, video, and more (July 2017).
URL https://blog.trendmicro.com/trendlabs-security-intelligence/android-backdoor-ghostctrl-can-silently-record-your-audio-video-and-more/

52. L. Stefanko, Android malware drops banker from png file (2016).
URL http://b0n1.blogspot.pt/2015/11/android-malware-drops-banker-from-png.html

53. Y. Fratantonio, A. Bianchi, W. Robertson, E. Kirda, C. Kruegel, G. Vigna, Triggerscope: Towards detecting logic bombs in android applications, in: 2016 IEEE Symposium on Security and Privacy (SP), 2016, pp. 377–396. doi:10.1109/SP.2016.30.

54. L. Stefanko, Bankbot trojan returns to google play with new tricks (September 2017).
URL https://www.welivesecurity.com/2017/09/25/banking-trojan-returns-google-play/

55. K. Tam, S. J. Khan, A. Fattori, L. Cavallaro, Copperdroid: Automatic reconstruction of android malware behaviors, in: 22nd Annual Network and Distributed System Security Symposium, San Diego, California, USA, 2015.

56. V. Rastogi, Y. Chen, X. Jiang, Droidchameleon: Evaluating android anti-malware against transformation attacks, in: Proceedings of the 8th ACM SIGSAC Symposium on Information, Computer and Communications Security, ASIA CCS '13, ACM, New York, NY, USA, 2013, pp. 329–334. doi:10.1145/2484313.2484355.
URL http://doi.acm.org/10.1145/2484313.2484355

57. A. Apvrille, R. Nigam, Obfuscation in android malware, and how to fight back, in: Virus Bulletin,, 2014.

58. J. Jung, C. Jeon, M. Wolotsky, I. Yun, T. Kim, AVPASS: Leaking and Bypassing Antivirus Detection Model Automatically (to appear), in: Black Hat USA Briefings (Black Hat USA), Las Vegas, NV, 2017.

59. A. Demontis, M. Melis, B. Biggio, D. Maiorca, D. Arp, K. Rieck, I. Corona, G. Giacinto, F. Roli, Yes, machine learning can be more secure! a case study on android malware detection, IEEE Transactions on Dependable and Secure Computing PP (99) (2017) 1–1. doi:10.1109/TDSC.2017.2700270.

60. W. Yang, D. Kong, T. Xie, C. A. Gunter, Malware detection in adversarial settings: Exploiting feature evolutions and confusions in android apps, in: Annual Computer Security Applications Conference, ACSAC '17, 2017.

61. K. Grosse, N. Papernot, P. Manoharan, M. Backes, P. McDaniel, Adversarial Examples for Malware Detection, Springer International Publishing, Cham, 2017, pp.

62–79. doi:10.1007/978-3-319-66399-9_4.
URL https://doi.org/10.1007/978-3-319-66399-9_4