TSINGHUA SCIENCE AND TECHNOLOGY
ISSN 1007-0214 10/10 pp114-123
Volume 21, Number 1, February 2016

DroidDetector: Android Malware Characterization and Detection Using Deep Learning

Zhenlong Yuan, Yongqiang Lu, and Yibo Xue*

Abstract: Smartphones and mobile tablets are rapidly becoming indispensable in daily life. Android has been the most popular mobile operating system since 2012. However, owing to the open nature of Android, countless malwares are hidden in a large number of benign apps in Android markets that seriously threaten Android security. Deep learning is a new area of machine learning research that has gained increasing attention in artificial intelligence. In this study, we propose to associate the features from the static analysis with features from dynamic analysis of Android apps and characterize malware using deep learning techniques. We implement an online deep-learning-based Android malware detection engine (DroidDetector) that can automatically detect whether an app is a malware or not. With thousands of Android apps, we thoroughly test DroidDetector and perform an in-depth analysis on the features that deep learning essentially exploits to characterize malware. The results show that deep learning is suitable for characterizing Android malware and especially effective with the availability of more training data. DroidDetector can achieve 96.76% detection accuracy, which outperforms traditional machine learning techniques. An evaluation of ten popular anti-virus softwares demonstrates the urgency of advancing our capabilities in Android malware detection.

Key words: Android security; malware detection; characterization; deep learning; association rules mining

1 Introduction

Android dramatically surpassed a billion shipments of its devices in 2014 and has remained the No.1 mobile operating system since 2013, according to a recent report from Gartner[1]. Android markets, such as the Google Play Store and other third-party markets, play an important role in the popularity of Android devices. However, the openness of Android makes these markets hot targets for malware attacks[2, 3] and causes countless instances of malware being hidden behind a large number of benign apps that seriously threatens users' security and privacy. Moreover, a report from McAfee Labs reveals that 3.73 million pieces of mobile malware were identified in 2013, increasing an astounding 197% from the end of 2012[4]. Consequently, an urgent need arises to develop powerful solutions for Android malware detection. Unfortunately, the Android market currently has no such solution.

Today, the main countermeasure to defense against malware on Android platforms is a risk communication mechanism that warns users about the permissions required before installing each app. This mechanism is rather ineffective[5, 6], as it presents permissions in a

• Zhenlong Yuan is with the Department of Automation and Research Institute of Information Technology (RIIT), Tsinghua University, Beijing 100084, China. E-mail: yuanzl11@mails.tsinghua.edu.cn.
• Yongqiang Lu is with the Department of Antivirus, Baidu Inc., Beijing 100085, China. E-mail: luyongqiang@baidu.com.
• Yibo Xue is with the Research Institute of Information Technology (RIIT) and Tsinghua National Lab for Information Science and Technology (TNList), Tsinghua University, Beijing 100084, China. E-mail: yiboxue@tsinghua.edu.cn.
∗ To whom correspondence should be addressed.
Manuscript received: 2016-01-01; accepted: 2016-01-07

stand-alone fashion, thus requiring too much technical knowledge for a user to be able to differentiate malware from benign apps. Note that both a benign and a malicious app may require the same permissions and are thus indistinguishable via this permission-based mechanism. In general, permission-based approaches are developed primarily for risk assessment[7–10] rather than malware detection.

Clearly, a better characterization of Android malware would achieve a better accuracy in their detection. DroidRanger[11] and RiskRanker[12], two typical signature-based methods, try to characterize malware using specific patterns in the bytecode and Application Program Interfaces (API) calls. However, these signature-based methods can be easily evaded by bytecode-level transformation attacks[13]. It has been pointed out that signature-based methods, such as those presented in Refs. [12, 14], cannot catch several types of code-loading-technique-based Android malware[15].

Previous research[16] has revealed that Android malware is rapidly evolving to circumvent signature-based characterizations and thus calls for the development of next-generation anti-mobile-malware solutions. Android malware evidently cannot be adequately characterized using only specific patterns (signatures). In view of this situation, machine-learning-based methods are being proposed to characterize Android malware that extract features by the static[17–20] or dynamic analysis[21] of Android apps and learn the distinctions between malware and benign apps automatically. In particular, these machine-learning-based methods can avoid the need to manually craft and update detection rules, which is crucial for keeping pace with the variety of Android malware.

Deep learning[22] is a new area of machine learning research that imitates the way the human brain works and has gained increasing attention in the field of artificial intelligence. It has motivated a great number of successful applications in speech recognition, image classification, and natural language processing. Preliminary work in deep learning as it applies to Android malware detection was presented in Ref. [23]. In this study, we first extracted a total of 192 features from static and dynamic app analyses and then applied the deep learning technique to distinguish malware from benign apps. Our premise is that deep learning with a deep architecture can evolve high-level representations by associating features from static analysis with those from dynamic analysis, which can then better characterize Android malware. Experiments on a large number of real-world apps show that deep learning is especially suitable for characterizing Android malware and can achieve a 96.76% detection accuracy, thereby significantly outperforming traditional machine learning techniques, such as Naïve Bayes, C4.5, Logistic Regression, Support Vector Machine (SVM), and Multi-layer Perceptron.

In this study, our contributions include: (1) We describe our development of a deep-learning-based Android malware detection engine (DroidDetector) that has been put online for user testing[24] and can automatically detect whether an app is a malware or not. (2) We crawl 20 000 apps from the Google Play Store and collect 1760 malwares from the well-known Contagio Community[25] and Genome Project[26]. With these real-world apps, we thoroughly test DroidDetector and perform an in-depth analysis on the features that deep learning essentially exploits to characterize malware using association rule mining techniques. (3) We conduct experiments on ten popular anti-virus softwares and reveal that they are extremely vulnerable to repackaging attacks. In the light of our analyses, we conclude that deep learning is a promising technique for Android malware detection.

The rest of this study is organized as follows. In Section 2, we present the extraction of a total of 192 features from static and dynamic app analyses. In Section 3, we introduce the details of our deep learning model and the design of DroidDetector. The experiments we conducted on large-scale app sets are described in Section 4. Finally, we offer a brief discussion in Section 5 and draw our conclusions in Section 6.

2 Feature Extraction

To systematically characterize Android apps (i.e., both malware and benign apps), we conduct static and dynamic analyses to extract features from each app, as shown in Fig. 1. All the features fall under one of three types: required permissions, sensitive APIs, and dynamic behaviors. Among them, required permissions and sensitive APIs are extracted through the static analysis, whereas dynamic behaviors are extracted through dynamic analysis. Specifically, all we need is the installation file (i.e., apk file) of each Android app.

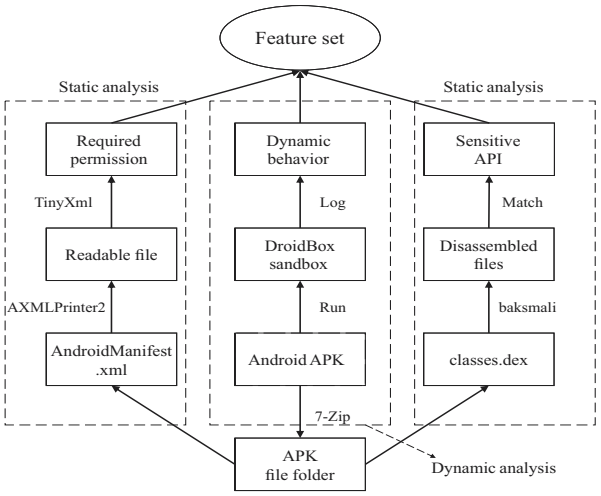

Fig. 1 Feature extraction for an Android app.

In the static phase, we uncompress the .apk file with the `7-Zip` tool and then focus on parsing the two files `AndroidManifest.xml` and `classes.dex`. By parsing the `Android Manifest.xml` file with the tool `AXML-Printer2` and the parser `TinyXml`, we can obtain the permissions required by the app. For example, `android.permission.call_phone` is the permission required for an app to make a phone call and `android.permission.camera` is the permission required for an app to access the camera. In this step, we looked for a total of 120 permissions. By parsing the `classes.dex` file with the disassembler `baksmali`, we can know which API functions are called. For example, `chmod` is a sensitive API that might be used for changing users' permissions on files and `ContentResolver;->delete` is a sensitive API that might be used for deleting users' messages or contacts. In this step, we looked for a total of 59 sensitive API functions.

In the dynamic phase, we install and run each app in `DroidBox`[27]. `DroidBox` is an Android application sandbox that extends TaintDroid[28], which can execute a dynamic taint analysis with system hooking at the application framework level and monitor a variety of app actions such as information leaks, network and file input/output, cryptography operations, Short Message Services (SMS), and mobile phone calls. In this study, we ran the apps inside `DroidBox` for a period of time to obtain the executed app actions (i.e., dynamic behaviors) of each app. In this phase, we monitored a total of 13 app actions. For instance, `action_sendnet` is the action that sends data over the network, `action_phonecalls` is the action that makes a phone call, and `action_sendsms` is the action that sends SMS messages.

In this way, we obtained a total of 192 features for each app through static and dynamic analyses. Note that each feature is binary, indicating that when a feature occurs in an app, its feature value is 1; otherwise, its feature value is 0. In addition, all the tools (i.e., `7-Zip`, `AXMLPrinter2`, `TinyXml`, `baksmali`, and `DroidBox`) referred to in this section are open source for use by the public.

3 Deep Learning Engine

Traditional machine learning models (e.g., SVM and C4.5) that have less than three layers of computation units are considered to have shallow architectures. Fortunately, deep learning models with a deep architecture change that situation. In practical use, a deep learning model can be constructed with different deep architectures[22], e.g., Deep Belief Networks (DBN) and convolutional neural networks. For this study, we chose DBN architecture to construct our deep learning model and characterize Android apps.

As shown in Fig. 2, the construction of a deep learning model has two phases, the unsupervised pre-training phase and supervised back-propagation phases. In the pre-training phase, the DBN is hierarchically built by stacking a number of Restricted Boltzmann Machines (RBM), with the deep neural network regarded as a latent variable model, which is beneficial for gradually evolving high-level representations. In the back-propagation phase, the pre-trained DBN is fine-tuned with labeled samples in a supervised manner. The deep learning model uses the same app set in both phases of the training process. In this way, the deep learning model is completely built.

We implemented the Android malware detection engine DroidDetector based on the deep learning

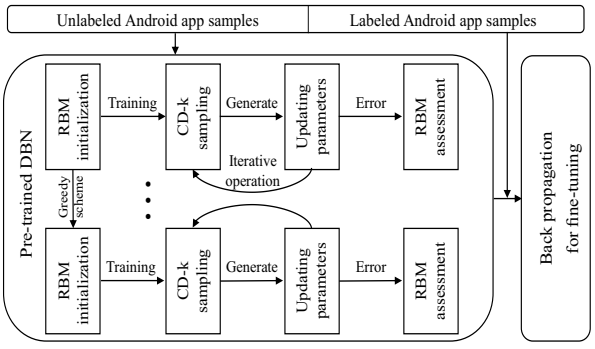

Fig. 2 Deep learning model constructed with DBN.

model, as shown in Fig. 3. DroidDetector has been open online[24] for user testing and can automatically detect whether a submitted app is a malware or not. Once the .apk file of an app is submitted, DroidDetector checks its integrity and determines whether it is a complete, correct, and legitimate Android application. Next, DroidDetector executes a static analysis to obtain the permissions and sensitive APIs that are used by this app. Then, DroidDetector executes a dynamic analysis by installing and running this app in DroidBox for a fixed period of time. In this way, DroidDetector identifies the dynamic behaviors that are being performed.

We have completely automated the static and dynamic analyses of DroidDetector. Once the total 192 binary features described in Section 2 have been collected, they are input in the deep learning model for classification. The detection results, including detailed information from the integrity check and both analyses, are then reported to the users. Since the new types of apps are constantly emerging, we have designed two crawler modules. One is used for crawling benign apps from the Google Play Store and the other is used for crawling malware from well-known malware sources (e.g., Contagio and Genome). Using this strategy, we expect DroidDetector to keep pace with the evolution of Android malware.

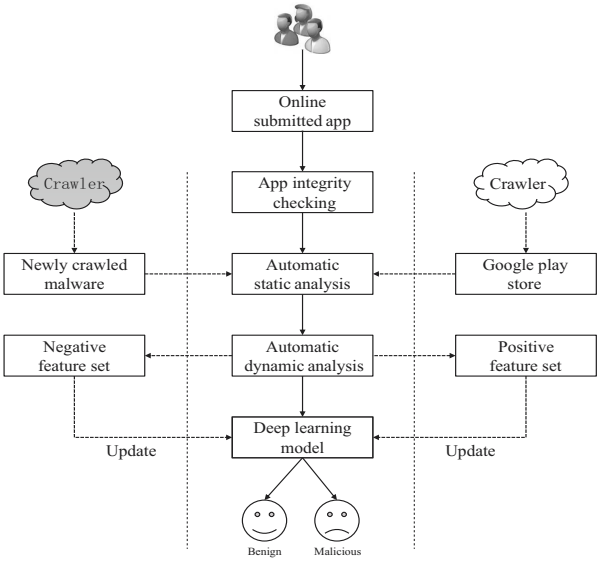

Fig. 3 Framework of DroidDetector.

4 Evaluation

To validate the ability of the deep learning model to detect Android malware and make an in-depth analysis on the features that deep learning essentially exploits to characterize malware, we conducted experiments on three public app sets. One benign app set was randomly crawled from the Google Play Store, which contains a large-scale of 20 000 apps. Although there might be a few malicious apps hidden among them, we regard all of them as benign apps. Another two malicious app sets were respectively collected from the Contagio Community (there are only about 400 apps at present, as we have accumulated for two years, 500 malicious apps are collected) and Genome Project (including 1260 malicious apps). So, the total number of malicious apps is 1760 while there are 20 000 benign apps.

4.1 Deep learning performs best

In the next part of the study, we mixed together an equal number of malicious and benign apps. In doing so, we obtained a training set and a test set, either of which included 880 malicious and 880 benign randomly selected apps. The following experiments were all performed on these two app sets.

Several parameters need to be set when building deep learning networks, including the number of layers, number of neurons in each layer, contrastive divergence (CD-k) value, and number of iterations. Table 1 shows

Table 1 Classification accuracies with different deep learning model constructions.

Number of layers	Number of neurons	Benign apps		Malicious apps		Overall accuracy (%)
		Precision (%)	Recall (%)	Precision (%)	Recall (%)	
6	[150, 150, 150, 150, 150, 150]	97.45	95.68	95.76	97.50	96.59
5	[150, 150, 150, 150, 150]	97.93	91.36	91.91	98.07	94.72
4	[150, 150, 150, 150]	93.45	95.68	95.58	93.30	94.49
3	[170, 170, 170]	97.22	91.36	91.85	97.39	94.38
3	[150, 150, 150]	97.68	95.68	95.77	97.73	96.70
3	[130, 130, 130]	97.23	95.68	95.75	97.27	96.48
2	[170, 170]	92.94	95.68	95.55	92.73	94.20
2	**[150, 150]**	**97.79**	**95.68**	**95.77**	**97.84**	**96.76**
2	[130, 130]	97.83	92.16	92.59	97.95	95.06
1	[150]	97.23	95.68	95.75	97.27	96.48

that the classification accuracy varies with the two key factors (i.e., number of layers and number of neurons in each layer). We can see that deep learning (DBN) can achieve a 96.76% accuracy when setting the number of layers to 2 and number of neurons in each layer to 150. In addition, we can see that the average accuracy under different model constructions is higher than 95%.

Moreover, we compared the classification accuracies of the deep learning model with the traditional machine learning models, as shown in Table 2. The five machine learning models were all optimized to achieve optimal accuracy using grid search techniques. We tested four types of kernel functions (i.e., linear, polynomial, radial basis, and sigmoid) for the SVM to achieve the highest possible accuracy in this experiment. From the table, we can clearly see that the deep learning model significantly outperforms other malware detection models, such as Naïve Bayes, C4.5, Logistic Regression, SVM, and Multi-layer Perceptron (a conventional neural network).

Moreover, to evaluate whether it is valuable to associate features from the static analysis with those from the dynamic analysis, we conducted experiments employing either static or dynamic features in the construction of a deep learning model. From Table 2, we see that it is vital to combine features from both static and dynamic analyses for effective malware detection. As there are currently millions of apps, even 1% improvement in accuracy is invaluable for practical use.

As noted above, although we regard all of the apps crawled from the Google Play Store as benign, there might be some malware concealed there. But while there is no guarantee that all 20 000 apps are truely benign and contain no malicious behavior, according to the previous studies of the Google Play Store, their malware infection rate is extremely low, especially after Google deployed its Bouncer[29] service for malware detection. For example, DroidRanger has reported an infection rate of approximately 0.02% in the Google Play Store, and RiskRanker reported an infection rate of only 0.0038% (i.e., two malicious apps out of 52 208). Even if the real malware infection rate is higher than these reported values, it is relatively very small compared with the accuracy improvement associated with the deep learning model. Moreover, note that machine learning (including deep learning) techniques are noise tolerant in training models. As such, we can conclude that the few incidences of hidden malware hardly influence our accuracy comparisons.

4.2 Features exploitation

Next, we conducted experiments on the app sets introduced in Section 4.1. We performed an in-depth analysis on the features exploited by deep learning to distinguish malicious and benign apps using association rule mining techniques. In these experiments, although we used 880 malicious apps and 880 benign apps in our analysis, we consider that the analysis results only reflect trends in the feature differences between them and are not absolute distinctions in real-world situations. First, we examined the ten top-ranked features in either malicious or benign classes. The results show that they both have the same seven features (i.e., Internet, action_fdaccess, action_accessdfiles, java_net_url_openconnection, action_dexclass_load, action_recvsaction, and access_network_state) and each has three individual features (i.e., contentresolver_query (*benign*), java_net_httpurlconnection_connect (*benign*), httpclient_execute (*benign*) and read_phone_state (*malicious*), write_external_storage (*malicious*), telephonymanager_getdeviceid (*malicious*)). Thus we obtained a total of 13 features from both classes.

As shown in Fig. 4, we calculated the ratio of each of the 13 features existing in either class. We can see that the same seven features have similar

Table 2 The comparison between deep learning and traditional machine learning models.

Input features	Machine learning model	Benign apps		Malicious apps		Overall accuracy (%)
		Precision (%)	Recall (%)	Precision (%)	Recall (%)	
Static & dynamic	C4.5	98.01	67.27	75.09	98.64	82.95
Static & dynamic	SVM	93.63	91.93	92.08	93.75	92.84
Static & dynamic	Naïve Bayes	90.27	75.91	79.22	91.82	83.86
Static & dynamic	Logistic regression	91.91	46.48	64.18	95.91	71.19
Static & dynamic	Multi-layer perceptron	97.88	78.75	82.22	98.30	88.52
Static & dynamic	**DBN**	**97.79**	**95.68**	**95.77**	**97.84**	**96.76**
Static only	DBN	97.77	79.89	83.00	98.18	89.03
Dynamic only	DBN	67.09	83.41	78.08	59.09	71.25

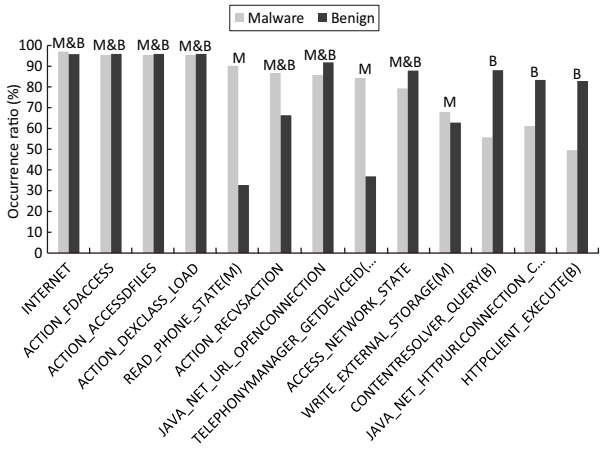

Fig. 4 Thirteen top-ranked features.

ratios in both classes, whereas those of the six individual features are much different. In particular, among the six individual features, read_phone_state (a permission) and telephonymanager_getdeviceid (a sensitive API) are very different, whereas the difference in the other four features is relatively small in both classes. Note that both read_phone_state and telephonymanager_getdeviceid belong to the malicious class, indicating that malicious apps may possibly tend to use them to attack users' mobile devices.

Next, we conducted experiments on the ten most different features between the malicious and benign classes. As shown in Fig. 5, we can see that the read_phone_state and telephonymanager_getdeviceid, also listed in Fig. 4, are two of the most different features as well. Although read_sms (a permission) and telephonymanager_getline1number (a sensitive API) do not appear in the ten top-ranked features, the difference between them in the malicious and benign classes is great and only occurs frequently in malicious

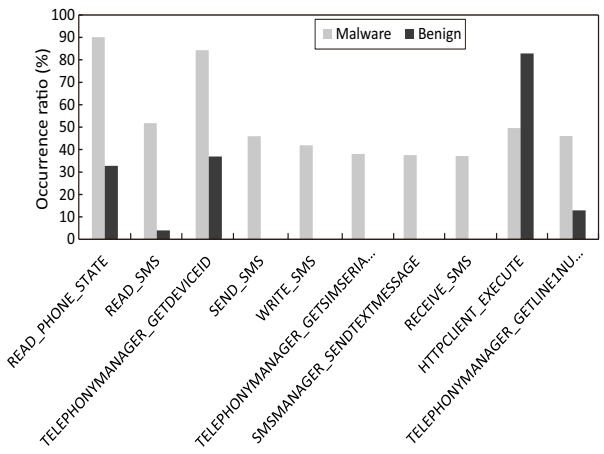

Fig. 5 Ten most different features.

apps. Specifically, read_sms is used for reading short messages and telephonymanager_getline1number is used for obtaining a user's phone number, possibly indicating that malicious apps prefer to steal users' sensitive information (i.e., SMS messages and phone numbers).

Moreover, we can see that httpclient_execute occurs more frequently in benign apps than in malicious apps. Particularly, httpclient_execute is generally used for requesting a remote HTTP service. This is an interesting and counterintuitive finding because we would expect that malicious apps would usually connect to remote HTTP servers to transfer more data. Based on this observation, however, malicious apps tend to use other concealed ways (e.g., socket communications and SMS messages) to secretly transmit data.

In addition, we can see that among the other five most different features (i.e., send_sms, write_sms, telephonymanager_getsimserialnumber, receive_sms, and smsmanager_sendtextmessage), three (i.e., send_sms, write_sms, and receive_sms) are permissions, whereas the other two are sensitive APIs. Only telephonymanager_getsimserialnumber is used to obtain the serial number of SIM cards, and the others all relate to SMS management. These five features occur frequently in malicious apps but occur rarely in benign apps. Although we used a large number of malicious and benign apps in this analysis, the app set is still relatively small compared with the millions of existing apps in Android markets. Therefore, the occurrence ratios of features in malicious and benign classes presented herein are considered to only reflect the trends of the feature differences between them and not their absolute values.

Furthermore, we performed experiments on the correlations among the 192 total features to dig out the association rules that tend to be used only by Android malware. Two measurement metrics exist: support and confidence. The support supp(X) of an itemset X is defined as the proportion of transactions (apps) in the data (app) set, which contain the itemset (multiple features), and confidence of a rule ($X \cup Y$) is defined as supp($X \cup Y$)/supp(X) where Y represents the *malicious class*. Specifically, *M-support* is defined as the proportion of apps in the *malicious class* that contain the itemset.

We used the Patient Rule Induction Method (PRIM) bump hunting algorithm[30] which can learn a set of association rules and maximize a target variable of interest, to obtain the top ten 2-itemset association

rules that have the maximum *confidence* values, as shown in Table 3. In other words, these 2-itemset association rules present the two features that tend to be simultaneously used only by Android malware. For this experiment, we eliminated the impacts of the five most different 1-itemset rules, since they were already shown in Fig. 5. Moreover, we continued mining the top five 3-itemset rules after eliminating the impacts of the top ten 2-itemset rules, the results of which are shown in Table 3.

As noted in the previous subsection, while there might be some malware hidden in the benign app set, such a small proportion of malware does not influence our results, since the focused features in the malicious and benign classes display such significant differences.

4.3 More data is much better

In real-world situations, the ratio between malicious and benign apps may not be 1:1, so we conducted experiments with various ratios of malicious to benign apps, including 1:1, 1:2, 1:5, 1:10, 1:20, 1:50, and 1:100. As shown in Table 4, we can see that more training data leads to a better accuracy when the ratio of malicious apps to benign apps is 1:1. Particularly, deep learning can achieve a high level of 96.60% detection accuracy when the training number of either class reaches 500. Moreover, we can see that although the precision and recall of malicious class fluctuates to some extent, the overall accuracy rises to near 100% with increased proportions of benign apps. We consider that this fluctuation of malicious class is reasonable since the ratio of malicious apps to benign apps is so lopsided, and there are too few malicious apps in the training and test sets. However, we believe that the classification accuracy of malicious class could be further improved by training with more malicious samples even under this lopsided ratio, because "deep learning happens to have the property that if you feed it more data it gets better and better"[31].

It is hard to make fair comparisons between the deep learning model and real-world anti-virus softwares because we do not know the number of malicious and benign apps used (or trained) by each anti-virus software to detect unknown apps. It is possible that the public apps we used for testing have also been collected

Table 3 Association rules mining based on the PRIM algorithm.

Number of items	Association rules (1 represents its occurrence while 0 represents not)	M-support (%)	Confidence (%)
2	[READ_PHONE_STATE=1, DEFAULTHTTPCLIENT_EXECUTE=0]	56	99.7
2	[READ_SMS=1, READ_PHONE_STATE=1]	50	99.8
2	[READ_SMS=1, GET_ACCOUNTS=0]	48	99.7
2	[READ_SMS=1, ACCESS_NETWORK_STATE=1]	46	99.8
2	[READ_SMS=1, TELEPHONYMANAGER_GETDEVICEID=1]	45	99.5
2	[HTTPCLIENT_EXECUTE=0, READ_PHONE_STATE=1]	44	99.5
2	[READ_SMS=1, GET_TASKS=0]	42	99.7
2	[HTTPCLIENT_EXECUTE=0, TELEPHONYMANAGER_GETDEVICEID=1]	40	99.7
2	[CONTENTRESOLVER_QUERY=0, ACTION_RECVSACTION=1]	36	99.5
2	[TELEPHONYMANAGER_GETLINE1NUMBER=1, WAKE_LOCK=0]	34	99.5
3	[JAVA_NET_HTTPURLCONNECTION_CONNECT=0, CAMERA_OPEN=0, JAVA_NET_URL_GETCONTENT=0]	38	90.0
3	[JAVA_NET_HTTPURLCONNECTION_CONNECT=0, CAMERA=0, JAVA_NET_URL_GETCONTENT=0]	37	90.0
3	[JAVA_NET_HTTPURLCONNECTION_CONNECT=0, CAMERA_OPEN=0, INTERNET=1]	35	90.0
3	[ACCESS_FINE_LOCATION=0, ACTION_OPENNET=1, RECORD_AUDIO=0]	35	89.9
3	[BLUETOOTHADAPTER_ENABLE=0, BROADCAST_STICKY=0, READ_CONTACTS=1]	35	89.9

Table 4 Deep-learning-based malware detection with different mixes of malicious and benign apps.

Ratio	Malware (Training/Test)	Benign (Training/Test)	Benign apps		Malicious apps		Overall accuracy (%)
			Precision (%)	Recall (%)	Precision (%)	Recall (%)	
1:1	100/100	100/100	94.06	95.00	94.95	94.00	**94.50**
1:1	200/200	200/200	95.52	96.00	95.98	95.50	**95.75**
1:1	**500/500**	**500/500**	**95.69**	**97.60**	**97.55**	**95.60**	**96.60**
1:2	100/100	200/200	95.15	98.00	95.74	90.00	**95.33**
1:5	100/100	500/500	98.01	98.60	92.78	90.00	**97.17**
1:10	100/100	1000/1000	97.84	99.80	97.50	78.00	**97.82**
1:20	100/100	2000/2000	97.99	99.80	93.65	59.00	**97.86**
1:50	100/100	5000/5000	99.60	99.62	80.81	80.00	**99.24**
1:100	100/100	10000/10000	99.62	99.92	88.57	62.00	**99.54**

in advance by these softwares for signature-based or other technology-based detection methodologies. Therefore, we only performed experiments on the ten popular anti-virus softwares with a total of 50 apps from the Contagio app set in 2014.

As shown in Fig. 6, we then evaluated these ten anti-virus softwares before and after repackaging the test apps. The repackaging of these apps is simple, involving just disassembling and reassembling them. In this way, the MD5 or SHA hash values of these apps will differ from their original values without changing any functionality. However, we can see that most of the anti-virus softwares experience a large drop in their detection rate in repackaged apps. These results reveal that most of these softwares are likely to detect Android malware based on signature-based approaches, which might employ a blacklist mechanism to match the hash values of their collected malware. We can also see that the detection accuracies of six of the softwares are below 10%, while eight of them are below 35% in the detection of repackaged malware. This indicates that these anti-virus softwares are extremely vulnerable to repackaging attacks.

Although BitDefender performs stably both before and after repackaging, it achieves a detection accuracy of only 72%. We must also note that BitDefender might have collected some of our test apps and detected them by comparing our test apps with its collected malware, using other signature-based approaches. Even if BitDefender did not collect any of the test apps in advance, its 72% accuracy shows that there remains serious challenge in real-world malware detection.

Machine-learning-based detection methods can hardly be impaired by repackaging attacks since these methods can characterize and detect malware using static or dynamic analysis of Android apps to extract a set of features. As demonstrated in our experiments,

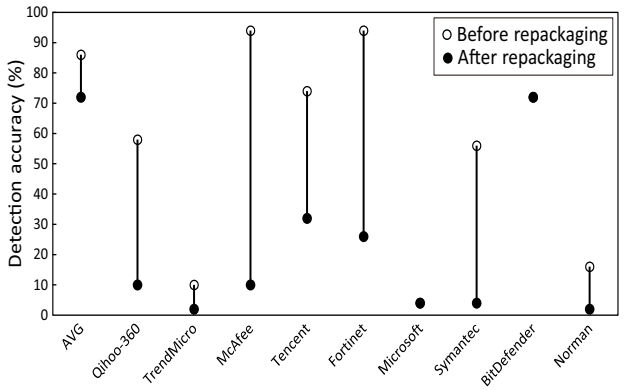

Fig. 6 Test on ten popular anti-virus softwares.

deep learning is probably the best machine learning technique for Android malware detection.

5 Discussion

We consider that there are three critical issues in machine-learning-based Android malware detection.

The first issue is the ***machine learning model*** used. In this study, we selected deep learning because it can learn high-level representations by associating features from static analysis with those from dynamic analysis, which makes it possible to better characterize Android malware. Our experiments also demonstrated that the deep learning model significantly outperforms traditional machine learning models.

The second issue is the ***features*** collected. Note that a more comprehensive and fine-grained set of features leads to the better characterization and detection of Android malware. For example, we combined the features from static and dynamic analyses to achieve a better accuracy in malware detection. In our opinion, if a malware cannot be identified correctly, its malicious characteristics must not have been properly learned by the machine learning model. Note that any app defined as a malware must have some special characteristics that have been defined as malicious behaviors. Therefore, to characterize and detect more types of malware, more fine-grained features that can cover more aspects of malware must be collected. For example, in addition to the 192 features presented in Section 2, many other characteristics can also be used as features, such as the four-layer profiles generated by ProfileDroid[32], the three tiered APIs exported by DroidScope[33], the program semantics generated by DroidSIFT[20], and the data-flow semantics extracted in Ref. [34].

Last but not least is the issue of ***training samples***. Obviously, the more types of training samples learned, the better accuracy a classifier will achieve in malware detection. However, the collection of Android malware is as yet a major challenge. Therefore, researchers and mobile users around the world should join together to contribute their new discoveries regarding Android malware in a public community like the Contagio Community, which has been collecting precious mobile malware resources for the public since 2008. In this way, by utilizing huge app samples, users can work together to defend against malware.

6 Conclusions and Future Work

Deep learning is a new area of machine learning

research. In this study, we extracted a total of 192 features from both static and dynamic analyses of Android apps and characterized malware using a DBN-based deep learning model. We designed DroidDetector and evaluated it with 20 000 benign apps crawled from the Google Play Store and 1760 malwares collected from the well-known Contagio Community and Genome Project. The results show that using DroidDetector with a deep learning model can achieve a superior accuracy under different conditions, significantly outperforming traditional machine learning techniques. At present, DroidDetector has been deployed online for user testing. Moreover, we delved deeper into the features that deep learning exploits to characterize Android malware using association rule mining techniques. The evaluation of ten popular anti-virus softwares indicates that it is a matter of urgency to make changes in Android malware detection.

Much more work is necessary. First, more fine-grained features should be extracted to characterize Android apps. A more comprehensive and fine-grained set of features can cover more aspects of Android malware and thus lead to a better characterization and detection of malware. In addition to the 192 total features in this study, we may also add the semantic-based features introduced in Refs. [20, 34] and types of features presented in Refs. [32, 33] to the feature set. In addition, richer discrete features rather than binary features can be used in establishing the deep learning model. For example, if one sensitive API function is called twice or one dynamic behavior occurs twice, we can set their corresponding feature values as 2 (i.e., discrete values) instead of 1 (i.e., binary values). And second, more app data (i.e., more types of malicious and benign samples) should be collected for training the deep learning model. More types of training samples could lead to a better optimization of the deep learning model, and thereby achieve a superior accuracy in real-world Android malware detection.

Acknowledgements

We would like to thank Zhen Chen for his insightful feedback and comments.

References

[1] Gartner, Gartner says Android has surpassed a billion shipments of devices, http://www.gartner.com/newsroom/id/2954317, 2015.

[2] T. Vidas, D. Votipka, and N. Christin, All your droid are belong to us: A survey of current Android attacks, in *Proceedings of the 5th USENIX Workshop on Offensive Technologies (WOOT)*, 2011, pp. 81–90.

[3] A. P. Felt, M. Finifter, E. Chin, S. Hanna, and D. Wagner, A survey of mobile malware in the wild, in *Proceedings of the 1st ACM Workshop on Security and Privacy in Smartphones and Mobile Devices (SPSM)*, 2011, pp. 3–14.

[4] McAfee, McAfee labs threats report, http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q4-2013.pdf, 2015.

[5] A. Mylonas, A. Kastania, and D. Gritzalis, Delegate the smartphone user? Security awareness in smartphone platforms, *Computers & Security*, vol. 34, pp. 47–66, 2013.

[6] Z. Fang, W. Han, and Y. Li, Permission based Android security: Issues and countermeasures, *Computers & Security,* vol. 43, pp. 205–218, 2014.

[7] J. Xu, Y.-T. Yu, Z. Chen, B. Cao, W. Dong, Y. Guo, and J. Cao, Mobsafe: Cloud computing based forensic analysis for massive mobile applications using data mining, *Tsinghua Science and Technology*, vol. 18, no. 4, pp. 418–427, 2013.

[8] R. Pandita, X. Xiao, W. Yang, W. Enck, and T. Xie, Whyper: Towards automating risk assessment of mobile applications, in *Proceedings of the 22nd USENIX Security Symposium (USENIX Security)*, 2013, pp. 527–542.

[9] Z. Qu, V. Rastogi, X. Zhang, Y. Chen, T. Zhu, and Z. Chen, Autocog: Measuring the description-to-permission fidelity in Android applications, in *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, 2014, pp. 1354–1365.

[10] D. Geneiatakis, I. N. Fovino, I. Kounelis, and P. Stirparo, A permission verification approach for Android mobile applications, *Computers & Security*, vol. 49, pp. 192–205, 2015.

[11] Y. Zhou, Z. Wang, W. Zhou, and X. Jiang, Hey, You, Get off of my market: Detecting malicious apps in official and alternative Android markets, in *Proceedings of the 19th Annual Symposium on Network and Distributed System Security (NDSS)*, 2012.

[12] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, Riskranker: Scalable and accurate zero-day Android malware detection, in *Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services (MobiSys)*, 2012, pp. 281–294.

[13] V. Rastogi, Y. Chen, and X. Jiang, Droidchameleon: Evaluating Android anti-malware against transformation attacks, in *Proceedings of the 8th ACM Symposium on Information, Computer and Communications Security (ASIA CCS)*, 2013, pp. 329–334.

[14] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, Unsafe exposure analysis of mobile in-app advertisements, in *Proceedings of the 5th ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec)*, 2012, pp. 101–112.

[15] S. Poeplau, Y. Fratantonio, A. Bianchi, C. Kruegel, and G. Vigna, Execute this! Analyzing unsafe and malicious dynamic code loading in Android applications, in *Proceedings of the 21th Annual Symposium on Network and Distributed System Security (NDSS)*, 2014.

[16] Y. Zhou and X. Jiang, Dissecting Android malware: Characterization and evolution, in *Proceedings of the 33rd IEEE Symposium on Security and Privacy (Oakland)*, 2012, pp. 95–109.

[17] D. Barrera, H. G. Kayacik, P. C. van Oorschot, and A. Somayaji, A methodology for empirical analysis of permission-based security models and its application to Android, in *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS)*, 2010, pp. 73–84.

[18] Y. Aafer, W. Du, and H. Yin, Droidapiminer: Mining api-level features for robust malware detection in Android, in *Proceedinds of the 9th International Conference on Security and Privacy in Communication Networks (SecureComm)*, 2013, pp. 86–103.

[19] D. Arp, M. Spreitzenbarth, M. Hbner, H. Gascon, K. Rieck, and C. Siemens, Drebin: Effective and explainable detection of Android malware in your pocket, in *Proceedings of the 21th Annual Symposium on Network and Distributed System Security (NDSS)*, 2014.

[20] M. Zhang, Y. Duan, H. Yin, and Z. Zhao, Semantics-aware Android malware classification using weighted contextual api dependency graphs, in *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS)*, 2014, pp. 1105–1116.

[21] I. Burguera, U. Zurutuza, and 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)*, 2011, pp. 15–26.

[22] Y. Bengio, Learning deep architectures for ai, *Foundations and Trends in Machine Learning,* vol. 2, no. 1, pp. 1–127, 2009.

[23] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, Droid-sec: Deep learning in Android malware detection, in *Proceedings of the 2014 ACM Conference on Special Interest Group on Data Communication (SIGCOMM, poster)*, 2014, pp. 371–372.

[24] DroidDetector: A deep learning based Android malware detection engine, http://analysis.droid-sec.com, 2015.

[25] Contagio mobile malware dump, http://contagiodump.blogspot.com, 2015.

[26] Android malware genome project, http://www.malgenomeproject.org, 2015.

[27] DroidBox: An Android application sandbox for dynamic analysis, http://www.honeynet.org/gsoc2011/slot5, 2015.

[28] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. Sheth, Taintdroid: An information-flow tracking system for realtime privacy monitoring on smartphones, in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[29] Bouncer: Android and security, http://googlemobile.blogspot.com/2012/02/android-and-security.html, 2015.

[30] J. H. Friedman and N. I. Fisher, Bump hunting in high-dimensional data, *Statistics and Computing*, vol. 9, no. 2, pp. 123–143, 1999.

[31] N. Jones, The learning machines, *Nature*, vol. 505, pp. 146–148, 2014.

[32] X. Wei, L. Gomez, I. Neamtiu, and M. Faloutsos, Profiledroid: Multi-layer profiling of Android applications, in *Proceedings of the 18th Annual International Conference on Mobile Computing and Networking (MobiCom)*, 2012, pp. 137–148.

[33] L.-K. Yan and H. Yin, Droidscope: Seamlessly reconstructing the os and dalvik semantic views for dynamic Android malware analysis, in *Proceedings of the 21st USENIX Security Symposium (USENIX Security)*, 2012, pp. 569–584.

[34] K. O. Elish, X. Shu, D. D. Yao, B. G. Ryder, and X. Jiang, Profiling user-trigger dependence for Android malware detection, *Computers & Security,* vol. 49, pp. 255–273, 2015.

Zhenlong Yuan is currently a PhD candidate at Department of Automation, Tsinghua University, China. He received his BEng degree from Sichuan University in 2011. Currently he works at the Network Security Lab under the supervision of Prof. Jun Li and Prof. Yibo Xue. His research interests include Android security, traffic classification, and anonymous communications.

Yongqiang Lu is currently a research engineer in Baidu Inc., Beijing, China. His research interests include mobile security, network management, and applied cryptography. He received his BEng and master degrees from Sichuan University in 2011 and 2014, respectively. He has been working on mobile security in the past several years.

Yibo Xue is currently a professor in the Research Institute of Information Technology (RIIT) at Tsinghua University, Beijing, China. He is an IEEE/ACM member and a CCF senior member. He received his BS and MS degrees in computer science from Harbin Institute of Technology in 1989 and 1992, respectively. He got his PhD degree in Institute of Computer Technology from Chinese Academy of Science in 1995. His main research interests are in the areas of network security, computer architecture, and social networks.