A Two-Stage Deep Learning Based Malware Detection Engine for Android

Tarun Chinmai Sekar
1213181929
tsekar@asu.edu

Soundarya Sankarasubramanian
1213119620
ssanka24@asu.edu

Sandhya Chandrasekaran
1215159426
schand61@asu.edu

Aravind Shanmugam
1213174948
ashanmu5@asu.edu

I. Abstract

Smartphones with Android Operating System have a diverse set of applications with a variety of features ranging from information storage to carrying out online transactions. As highly valuable and sensitive information is involved, it is of extreme importance to protecting our information from security threats. The presence of signatures and fingerprint-based techniques is not sufficient for the detection of malware. Deep learning based methods can be integrated to significantly improve the detection rate of malware in phones. Several static and dynamic techniques are present to extract features and classify appropriately. This paper presents a novel approach to significantly improve the existing malware detection techniques.

II. Introduction

In an increasingly mobile-centric world where most sensitive data now resides on mobile phones, security is of paramount importance. A Person's mobile phone has become a crucial part of their identity, both in the online world and in the physical world. Countries are increasingly adopting mobile based citizen identification systems [1] to authenticate citizens at the port of entries. As the lines between the physical and digital world are blurred, a compromise in the digital world directly affects the physical world. Android and iOS are the two dominant mobile operating systems in the world, and this paper focuses exclusively on detecting malware attacks on Android phones.

III. Fine grained android malware detection based on Deep learning

Android smartphones have been suffering from a lot of security issues these days. Though signature and fingerprint-based techniques are used commonly to identify and protect a user from any harm, it is not sufficient to protect against advanced attacks. Deep Learning-based approaches that use behavioral analysis techniques are required for detecting malicious behavior as and when it happens. Initial attempts [2] to use deep learning to train a classifier to detect malware focused on the static features available in the package, such as function calls, permissions requested, etc. A dataset comprising of major malware types was labeled and categorized into finer classes. The features were extracted from the Manifest.xml and the classes.dex files. The dataset had over 131K samples with about 5,560 malware samples and 123,453 benign samples. Multi-layer Neural networks with three layers were used to perform the classification. A number of layers and neurons were varied in each attempt to develop a model with the best architecture. The architecture was modeled to use Gradient descent to find the parameters during backpropagation. PRelu activation function was used for the hidden layers. Softmax layer was used as the final layer to obtain the probabilities of the samples belonging to various classes. Logarithmic log-likelihood loss function was used to learn the weights during backpropagation. The area under the ROC curve came to about 0.9977 for the problem.

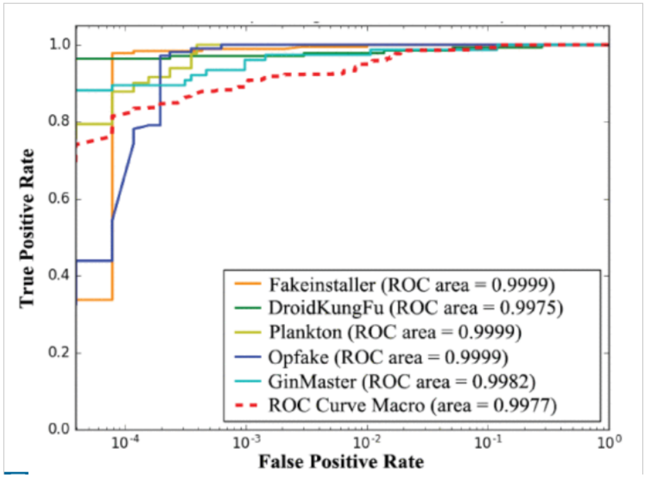


Fig. 1. ROC Curve [2]

When different architectures were employed, with an increase in the number of hidden layers, there was an increase in the precision, recall and F-1 scores. The best score was achieved with six hidden layers with each layer containing about 256 neurons. When three hidden layers with 1024 neurons were used in each, the precision value was the highest at 0.97. Figure 1 shows the ROC curve obtained for the classification problem.

IV. Sensitivity analysis of static features for Android Malware Detection

Later studies focused more on how apps exploit permissions granted to them for malicious purposes. The Apriori algorithm was used to construct the machine learning model. Schmid et al. came up with an approach based on signatures to detect malware. Supervised classification mechanism was used and it gave around 96 percent detection accuracy. The model was based on the extraction of features through function calls. [3]

This paper deals with feature sensitivity analysis. The AndroidManifest.xml file contains a lot of information about the application including the name of the application, package, information about the components such as services, activities, permissions, etc. The Classes.dex file contains the bytecode for the classes. Sensitivity analysis determines whether the input features have an effect on the output variable. Sensitivity index is the metric proposed for this mechanism. It calculates the difference between the minimum and maximum values and divides it by the maximum value. K Nearest Neighbours approach was used for classification. 10 fold cross-validation was used to tune the hyperparameters. ROC curve was constructed based on the output to analyze the performance of the model.

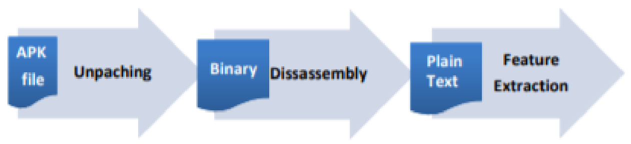


Fig. 2. Feature Extraction Process [4]

V. MMDA: METADATA BASED MALWARE DETECTION OF ANDROID

It is a mechanism which relies on metadata for classification. The dataset had two classes such as malicious samples and benign samples totaling to about 20045 samples. 10 fold cross validation mechanism was used to tune the hyper parameters of a random forest classifier. The performance in terms of the detection rate was compared to several other malware detectors available.

Features extracted include permissions, hardware features, and receiver actions. Some of the applications might ask for permission to access some external files. Some of the hardware features such as a camera may be required by the application. Receivers have the ability to accept notifications that have been broadcasted. The features which provide high discrimination between the two classes were used for the construction of feature vector space. In the end, 122 features were used for classification. Naive Bayes, Logistic Regression, Decision Trees and Support Vector Machines were used for classification. Classification accuracy was calculated based on True Positive Rate, False Positive Rate, ROC area, and accuracy metrics.

Two major experiments were done in the process. In the first, Classyshark was used to obtain the metadata of the applications. Classyshark is a tool developed by Google to get the metadata from AndroidManifest.xml file. After the construction of feature vector space, 10 fold cross-validation was performed to get a detection rate of 94 percent. Random Forest gives good accuracy in comparison to the other classifiers.

Scanner	Mmda-RandomForest	Ikarus	Cyren	AVG
Detection Rate	94.1%	96.6%	95.1%	94.4%

Fig. 3. Result for Experiment 1 [5]

McAfee	AVware	F-Secure	BitDedenfer	Tencent
93.7%	88.9%	80.9%	74.7%	61.1%

Fig. 4. Result for Experiment 1 [5]

The second experiments add to the first by performing validation using the most recent samples. The outcome of this model gave a moderate accuracy as well in comparison to the other scanners available.

Scanner	Mmda-RandomForest	Ikarus	Cyren	AVG
Detection Rate	90.3%	87.9%	84.7%	76.1%

Fig. 5. Result for Experiment 2 [5]

McAfee	AVware	F-Secure	BitDedenfer	Tencent
87.7%	71.9%	69.4%	55.6%	50.2%

Fig. 6. Result for Experiment 2 [5]

VI. NATIVE MALWARE DETECTION IN SMARTPHONES WITH ANDROID OS USING STATIC ANALYSIS, FEATURE SELECTION AND ENSEMBLE CLASSIFIERS

It is a method for malware analysis and detection which is carried out natively in the device. It can monitor new apps or updates on the device as well as the applications already installed in the device. Static analysis validates the permissions, hardware and software features requested by the applications. A model which incorporates different feature selection algorithms and machine learning classifiers classifies the application as either malware or benign. An off-device analysis process was carried out in the first stage, which determines the best machine learning classification model based on less execution time and higher precision. To obtain the best classification model, feature-selection was performed on the ensemble of machine learning classifiers. Finally, an on-device analysis process was performed to analyze the application and classify it as malware or benign. This approach can perform analysis on board smartphones without relying on an external server.

The data set consists of 2754 samples, of which one-half of the samples are malware samples whereas the remaining are benign. The first phase in the off-device analysis process (Figure 7). Feature extraction is performed for all the samples

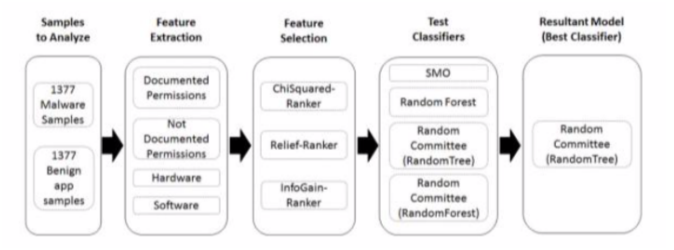


Fig. 7. Off-device Analysis process [6]

using the Android Asset Packaging Tool (AAPT). Static analysis on the applications recovered 135 documented permissions in the Android specification, 266 undocumented permissions and 55 documented hardware and software features. These features were represented in the form of a binary vector having 456 components. To perform the process of feature selection, Chi-square, relief, and information-gain algorithms were used which improves classification performance and accuracy. The usage of different feature selection methods provides a reduced feature-vector which balances the execution speed and classification precision. Out of these algorithms, Relief feature selection algorithm was found to produce the best results. The features obtained from the feature selection algorithm were used to train and evaluate machine learning classifiers such as SMO, Random Forest, Random Committee with Random Tree and Random Committee with Random Forest. After evaluation, Random Committee with Random Tree was found out to be the best classifier.

The second phase is the on-device analysis process Fig. 8. Random Committee with Random Tree was installed in three different devices. Broadcast Receiver Component monitors the device for any new application installed or new updates for existing applications and sends an alert to the malware detection application. Relief algorithm performs feature selection using Android PackageManager class and represents features in the form of a binary vector. Random Committee with Random Tree classification model classifies this application using the binary vector. The result of this model indicates whether this application is malware or benign.

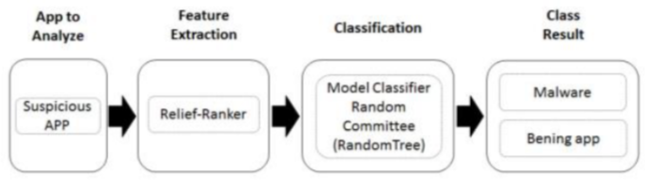
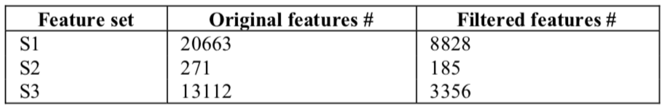


Fig. 8. On-device Analysis process [6]

VII. Ultra-lightweight Malware Detection of Android Using 2-level Machine Learning

This paper proposes an ultra-weight malware detection method which detects unknown malicious Android applications with limited resources. The model is based on a 2-level Support Vector Machine (SVM) which will be operating on new malware. Every Android application is inspected using a static analysis method which extracts different features from the application's manifest and dex code. These features are classified into three independent sets. These features are organized in a set of three feature spaces - particular method calls, hardware components & requested permissions and app components & Intents. These extracted feature sets are then mapped to their corresponding joint vector spaces where patterns and combinations of features are analyzed geometrically. These independent vector spaces represent the behavior of Android applications in different aspects. Behaviors will be mapped into specific regions in the vector spaces.

Static analysis is combined with machine learning to identify Android malware with high detection rate. This method is independent of manually crafted detection patterns. Each feature vector is fed to the RBF-SVM (Radial Basis Function - Support Vector Machine) algorithm which constructs detection rules for extracted features. The dimensionality of the vectors is reduced by filtering elements. A 2-level machine learning process is performed to classify malicious and benign applications. At the first level, each vector set is divided into two classes - the malicious set and benign set. As there are three independent vector spaces, each application gets a maximum of three result values. At the second level, the three result values of each application are gathered as a result group. These groups are embedded in a 3-dimensional vector space. Second level machine learning is performed which categorizes the result vectors into two groups. Finally, the malicious group is identified. This method is an ultra light-weight analysis because even though it uses RBF-SVM machine learning algorithm, it quickly detects Android malware due to the dimensionality reduction of the feature vectors (Figure 9).



Feature set	Original features #	Filtered features #
S1	20663	8828
S2	271	185
S3	13112	3356

Fig. 9. Performance of dimensionality reduction [7]

VIII. On Behavior-based Detection of Malware on Android Platform

This paper suggests a behavior-based malware detection system which captures the run-time behavior of software execution by examining system calls. A collection of real-world malware and benign samples are taken and machine learning approaches such a Naive Bayes learning, Support Vector Machines (SVM) are employed to learn the dynamic behavior of software execution. The detection system consists of offline training and online training phases. There is a training set which consists of a system calls generated by both malicious and benign software applications which are used to train the classifier. In the online detection phase, this classifier can be used to distinguish the software instance as malware or benign.

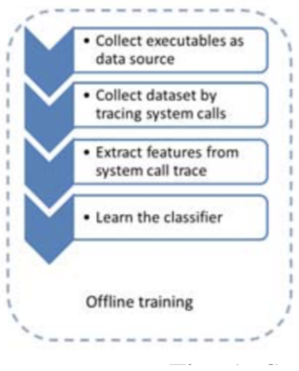

• Collect executables as data source
• Collect dataset by tracing system calls
• Extract features from system call trace
• Learn the classifier
Offline training

Fig. 10. Offline training phase [8]

In the offline training phase, real-world benign software and malware samples are collected. If the software exhibits similar activities and if these activities are used to learn the detection profile quantified by features, then the software is a benign software. During run-time, if the behavior of software instance is different from the profile, then the software is malign. This collection of malware and benign samples are then categorized into different groups. The collected software is then executed to record system calls using the tool Strace. These system calls are then parsed and mapped to the format required for the input to machine learning algorithms. This mapping process is done using SVM and Naive Bayes techniques. SVM is a learning system which uses hypothesis space of linear functions in a high dimensional feature space. It trains with the help of a learning algorithm from optimization theory. Naive Bayes is a probabilistic classifier based on Bayes theorem. The mapping process is done using SVMLight and Full and Naive Bayes Classifiers. SVMLight takes an integer as input whereas Full and Naive Bayes Classifier takes input in the form of a table format. The mapped system call information is taken as the input to the machine learning algorithms and then the classifier is trained.

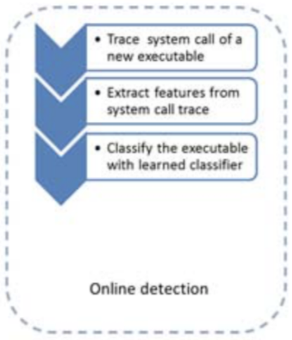

• Trace system call of a new executable
• Extract features from system call trace
• Classify the executable with learned classifier
Online detection

Fig. 11. Online detection phase [8]

In the online detection phase, a software instance is classified as malware or benign using the trained classifier. At first, the software instance is executed and generated system calls are dumped. These sequence of system calls are then mapped to the format required by the machine learning algorithms. SVMLight is used to classify a software instance. The classification module results in a positive number when the software is benign and a negative number when the software instance is malware. Naive Bayes classifier also classifies the software instance which generates a result file containing the classification result. The effectiveness of the proposed malware detection system is evaluated in terms of learning algorithms, the size of the training set, the length of n-grams, and overhead in training and detection processes. The detection system accurately determines malware accurately and rapidly.

IX. Idea: A Novel two phase system to detect malware

The proposed system will operate in a two-phase model. An Off-Device, offline analysis engine that learns the distinction between malicious and benign apps by training a deep neural network, and an on-device app that uses the learned information to automatically perform classification in real time. Information about the permissions required, hardware features needed, services and activities of an app can be obtained from the Manifest.xml and the method calls can be obtained from the classes.dex file. The model is also given the mapping of system calls and permissions so that it can understand the relation between them. Moreover, the app's category of use is also used as an input (game, banking/finance, social network, etc). The model now has an idea of what category of apps needs what permissions and what syscalls are allowed to be made by the app. This ”learned” information is sent to the on-device application for real-time classification.

The on-device component of the system uses the information learned by the offline system to classify apps installed on the phone as malicious or benign in real time. It monitors permission requests and syscalls made by every application and takes into account the context of the system call. For instance, if an app requests access to the camera or microphone when it is running in the background, it will be flagged as malicious because background applications do not have any legitimate reason to access the camera or the microphone. In addition to monitoring the permissions and system calls, the on-device component also monitors the network traffic and data being sent by the apps and flags any suspicious activity. Moreover, this on-device component also reports back its findings to the offline scanner, so that the global model can be periodically updated. Moreover, having an agent on the device allows the system to periodically monitor new apps and updates to existing apps installed on the device as well. The architecture of the system is depicted in Fig. 12

A. Feasibility

The dataset for the offline scanner can be easily built by scrapping Google Play Store to obtain the category, intended use of the app, etc, and to download the apks for the apps. Disassembly techniques can then be used on the downloaded apps to obtain the Manifest.xml and the classes.dex file. The

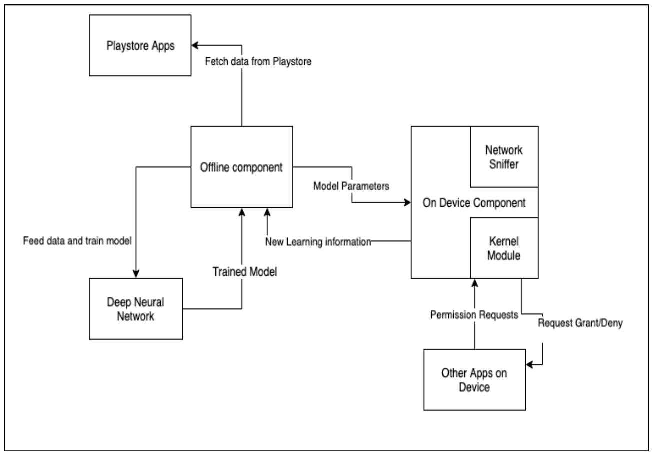


Fig. 12. Two Stage Deep Learning based Malware detection engine

model can then be trained on the extracted Manifest.xml and classes.dex files. The on-device component is slightly more complicated to implement. Auditing all system calls and permission requests will require root privileges and the installation of a kernel module. Installing a kernel module gives the application more flexibility, as it can actively deny permission requests it thinks are malicious instead of just reporting them. Moreover, to monitor network traffic, it is essential to install a Certificate Authority certificate to the root trust store on the device in order to perform TLS decryption. Once this is performed, the application can then monitor all network calls, monitor data sent out and flag apps that send personal data over insecure channels (HTTP) or to various other domains as well.

1) Development of Kernel Module: The development of the kernel module is the hardest part of this process, as usually the kernel modules are built with the headers from the Kernel currently installed on the device. However, owing to Android's massive fragmentation issue [9] varied versions of kernels exist on a large number of devices. This causes compatibility issues and makes the build and deployment of the kernel at scale to be difficult. Moreover, the kernel sources for some of the manufacturers are not available, leading to potential compatibility issues with their devices. This can hamper the widespread availability of this system.

2) Root Privileges: The nature of this application requires it to obtain and hold root privileges on the device. Android's SafetyNet [10] feature monitors devices for security threats (including rooting) and assigns trust scores to the device. A number of banking and financial apps use inputs from SafetyNet to determine whether to continue operating on the device or not. A rooted device almost always fails the SafetyNet tests although bypassing techniques exist [11]. Moreover, rooting a device requires technical knowledge and skill and hence is not an operation that can be performed by an average user. All of these factors impede the rapid deployment and proliferation of the app.

B. Drawbacks

The system relies heavily on the official app store - Google Play store for its training data and there are apps that are available for Android but are not on the play store. Examples of these include internal company-specific apps, apps distributed on alternative app stores, etc. As some of these app stores host ”cracked” versions of apps, it is not prudent to collate these apps into a dataset of benign apps. Hence, the system might not be aware of all characteristics of benign apps. Moreover, the on-device part of the system requires root privileges and the installation of a kernel module giving it privileged access to the device. This opens up an interesting attack vector, as any flaw in this app is disastrous and gives an attacker complete control over the device.

1) Usability Issues: When the system cannot be completely certain that a certain activity is malicious, it flags the action to the user, and the user is supposed to allow or deny it. However, if a large number of actions are ambiguous to the system, the system will keep prompting the user for input again and again. This leads to warning fatigue [12] leading the user to click on something they did not mean to. Moreover, some actions can only be classified as malicious or benign by a technically advanced user, limiting the target user base for the application.

X. Conclusion

Applying deep learning and machine learning techniques to malware analysis is a growing plot of study, and there's no perfect solution yet. As mobiles become more and more integrated with day-to-day life, threats become more sophisticated and hence the defense mechanisms need to evolve to keep up. Signature-based systems cannot adapt to the evolving threat landscape, and hence deep learning based systems are the way forward.

References

[1] US Customs and Border Protection, https://www.cbp.gov/travel/us-citizens/mobile-passport-control

[2] D. Li, Z. Wang and Y. Xue, ”Fine-grained Android Malware Detection based on Deep Learning,” 2018 IEEE Conference on Communications and Network Security (CNS), Beijing, 2018, pp. 1-2. doi: 10.1109/CNS.2018.8433204

[3] Schmidt, A.-D., et al. Static analysis of executables for collaborative malware detection on android. in Communications, 2009. ICC'09. IEEE International Conference on. 2009. IEEE.

[4] S. H. Moghaddam and M. Abbaspour, ”Sensitivity analysis of static features for Android malware detection,” 2014 22nd Iranian Conference on Electrical Engineering (ICEE), Tehran, 2014, pp. 920-924. doi: 10.1109/IranianCEE.2014.6999667

[5] Kun Wang, Tao Song, and Alei Liang. ”Mmda: Metadata Based Malware Detection on Android.” 2016 12th International Conference on Computational Intelligence and Security (CIS) (2016): 598-602. Web.

[6] S. Morales-Ortega, P. J. Escamilla-Ambrosio, A. Rodriguez-Mota and L. D. Coronado-De-Alba, ”Native malware detection in smartphones with android OS using static analysis, feature selection and ensemble classifiers,” 2016 11th International Conference on Malicious and Unwanted Software (MALWARE), Fajardo, 2016, pp. 1-8.

[7] L. Ma, Y. Yang, X. Wang and J. He, ”Ultra-Lightweight Malware Detection of Android Using 2-Level Machine Learning,” 2016 3rd International Conference on Information Science and Control Engineering (ICISCE), Beijing, 2016, pp. 729-733.

[8] Wei Yu, Hanlin Zhang, Linqiang Ge and R. Hardy, ”On behavior-based detection of malware on Android platform,” 2013 IEEE Global Communications Conference (GLOBECOM), Atlanta, GA, 2013, pp. 814-819.

[9] Park, Je-Ho, Young Bom Park, and Hyung Kil Ham. ”Fragmentation problem in Android.” 2013 International Conference on Information Science and Applications (ICISA). IEEE, 2013.

[10] Google Official SafetyNet documentation,
https://developer.android.com/training/safetynet/attestation.html

[11] Bypass SafetyNet using Magisk module
https://magiskroot.net/bypass-safetynet-issue-cts/

[12] Cash, Jared J. ”Alert fatigue.” American Journal of Health-System Pharmacy 66.23 (2009): 2098-2101.