ISSN 2278-3091
Volume 9, No.4, July – August 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse288942020.pdf
https://doi.org/10.30534/ijatcse/2020/288942020

Classification of Android Malware using its Image Sections

Deepak Thakur[1], Jaiteg Singh[2], Parvez Faruki[3], Tanya Gera[4]
[1]Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India, deepak.thakur@chitkara.edu.in
[2]Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India, jaiteg.singh@chitkara.edu.in
[3]Department of Technical Education, AVPT Institute, Rajkot, Gujarat, India, parvezfaruki.kg@gmail.com
[4]Chitkara University Institute of Engineering and Technology, Chitkara University, Punjab, India, tanya.gera@chitkara.edu.in

ABSTRACT

Privacy is a big concern as hackers are stealing data and misusing it by engineering malicious applications. There is a rapid increase in malware attacks like spyware, premium-rate SMS Trojans, botnets, aggressive adware, privilege escalation, and banking trojans which were distributed through the applications present on Google play store as well as unofficial application stores. Malware uses dominant techniques such as packing, encryption, a transformation of code, environment-aware approaches to evade detection. The traditional methods such as static and dynamic analysis of Android malware consume high computation resources and time. Moreover, cybercriminals use automation tools to generate numerous malware variants of the same family. This paper proposes a method to advance the classification of Android malware using visualization techniques. The visualization technique tends to transform Android malware into different image sections. The GIST algorithm is used to extract the features from the image sections. The extracted features are classified using machine learning algorithms such as K-Nearest Neighbors, Support Vector Machines, Random Forests, and Naive Bayes. This study evaluates the classification performance metrics of each classifier against every image file section. Experiment results show that the Android manifest image files have achieved a high accuracy of 92.7% with the SVM classifier.

Key words : malware, texture, classification, machine learning, computer vision

1. INTRODUCTION

The widespread of smartphones can be credited to the availability of daily usage apps, such as calls, SMSs, navigation, games, etc, in one place for the ease of the user. Android has an open marketplace due to which it has a huge community and intensely popular APIs. This, however, has repercussions because the financial benefits of such a huge community attracted a lot of malware applications from the year 2010 and so on. Based on research conducted by academic researchers and anti-malware companies, the basic algorithms (based on signatures) and static analysis are quite vulnerable to being attacked by malware [1]. The existing techniques of dealing with malware like encryption, code transformation, environmental approaches etc are known to make other types of malware that might or might not be already discovered [2]–[4]. Thus behavior, anomaly, and dynamic-analysis-based algorithms are used. Pinpointing out one specific method to deal with the problems is not feasible due to which multiple choices are available [5].

Developers have been motivated by the growth of Android to develop clever solutions commonly known as apps. The google play store agrees to host third party applications at a very menial fee due to which attackers have no problems getting to the root of downloads. The play store has more than 3.2 million apps and all available for download. The iOS app store verifies the uploaded applications manually, unlike android and thus security starts to get jeopardized. It depends on Bouncer, which is a virtually simulated environment to protect the store from malicious attacks. Malware knows how to exploit these apps to fuel their own benefit and extract sensitive data that might damage the image of the developer. Also, Android's open-source functionality allows the installation of third-party apps, opening up stores of unofficial and potentially dangerous application stores. This protection from third-party apps that might cause harm is a matter of concern [6].

This paper proposes a method to advance the classification of Android malware using visualization techniques [7]. The visualization technique tends to transform Android malware into different image sections. The GIST algorithm [8] is used to extract the features from the image sections. The extracted features are classified using machine learning algorithms such as K-Nearest Neighbors, Support Vector Machines [9], Random Forests, and Naive Bayes.

The rest of the paper is organized as follows: section 2 discusses the related work; section 3 contains the methodology of the work; section 4 discusses the results; section 5 concludes the findings.

2. RELATED WORK

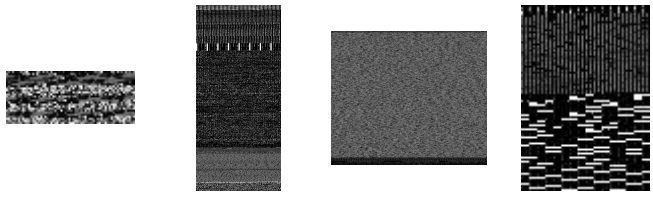
In [10] security rules were formed by the authors. Security properties of an application were compared with security rules. Application is declared as malicious if it does not match up the criteria of security rules. In [11] security checks were performed on a remote server where the replica of the exact phone is there on the virtual environment. Sensitive info was recorded using Tracer Tool and send to the cloud for replaying it. The authors performed the static and dynamic analysis of the application [12]. Classes.dex is extracted to convert into the human-readable format. The application was made to run in a sandbox and made use of a monkey runner to study malicious behavior. In [13], authors monitored the data (when the phone is connected over the network), battery consumption, CPU usage, and the processes currently running on the phone. Machine learning algorithms were applied to classify an application as benign or malicious. The authors used a crawler to download applications from five different stores [14]. They generated the behavioral footprints from 10 known Android Malware families. The proposed DroidRanger extracts the fundamental properties associated with each application. They also detected samples of known malware using permission-based filtering and by generating behavioral footprints and matching. They detected samples of unknown malware using heuristics based filtering and dynamic execution monitoring. The authors made use of metadata like the developer's information and other information available at market places to distinguish Android apps as malicious or benign. They do not perform sandboxing or code inspection in their work. Moreover, they implemented machine learning algorithms based on metadata collected. The authors first converted the apk file into smali file by making use of a tool named Baksmali [15]. Raw opcode sequences were then extracted from smali files. Their algorithm worked on smali files. They employed LSTM- a deep learning technique [16], [17] to make the system learn from opcode sequences to classify an app as malicious or benign. As entire raw opcode sequence may not possess malicious behavior, they also build the filter to separate the opcode subsequence which does not contain malicious behavior.

A literature survey suggests that traditional techniques make use of static and dynamic analysis. These techniques are dependent upon the feature engineering process which is a tedious task. Also, static analysis requires reverse engineering and dynamic analysis requires the execution of the application to detect malicious behavior. Visualization techniques helps to determine the non-intuitive features by visualizing the malware as an image. This paper uses the visualization technique to transform the maware into image and extracted the GIST features to classify the malware.

3. METHODOLOGY

The malware applications in the DREBIN dataset [18] were first converted into grayscale images. The four different types of images were created using Certificate (CR), Android Manifest (AM), Resources (RS), and Classes.dex (CL) files Android application structure. The 8-bit long substring is converted into decimal numbers ranging from 0 to 255. In this work, the width of the image is set to 192. The images of different malware families are depicted in Figure 1. GIST features are also known as handcrafted features that are used to extract the information from the images. We have used 512-texture descriptors for the classification purpose. The following machine learning algorithms were used to classify the GIST features:

Random Forest (RF): A forest is an ensemble of many trees. In a forest, several trees are being added together and each tree is trained separately on a bag of the data that is a random subset. The trees are trained in parallel and each tree trained does not depend on the other trees. The forest creates many trees on subsets of the data both bagged observations and subsets of variables. This is done to increase the difference in the trees in order to improve predictive power. The probability of the target response is averaged from each tree to create the final predicted probability. This lowers the interpretability of the forest compared to that of a single tree. There is still some interpretability because the forest can be viewed as an average of the trees inside of it.



GingerMaster malware family | Plankton malware family | Opfake malware family | GoldDream malware family

Figure 1 : Illustration of malware images

Support Vector Machine (SVM-RBF): Another popular machine learning algorithm is the support vector machine. It is a classification model that constructs a hyperplane to separate observations into classes. There is an infinite number of orientations for the hyperplane that would still separate these points completely. A margin is defined around this plane and formulate an optimization problem that strives to

maximize the margin. The more we can separate the classes, the more accurate our model will tend to be when applied to new data. The points that define the margin are called support vectors. SVM still attempts to construct the best separating hyperplane by introducing the concept of a penalty for observations that fall on the wrong side of the margin. Note that the size of the margin governs a trade-off between correctly classifying the training data and generalizing to future data. A wide margin will result in more training points being misclassified but we'll generalize much better and we should always keep in mind that applying your models to future data is the ultimate goal as opposed to achieving the best fit for the existing data.

KNN: KNN is one of the popular machine learning algorithms where K represents the number from 1 to n. The algorithm means that classified data point is as good as its K neighbors. There are three types of algorithm called supervised learning, unsupervised learning, and reinforcement learning. Reinforcement learning is sometimes also called semi-supervised learning. KNN or K nearest neighbor is a supervised learning algorithm and it tries to predict the classification of new sample data points form a particular population set. KNN is also called a lazy algorithm that does not learns itself or take decisions rather when it gets the test instance, it uses the stored instance in memory in order to find the possible classification of data points.

Naïve Bayes (NB): The naive bayes algorithm is a machine learning algorithm for classification problems. Naïve bayes is robust enough to fast and quick predictions. It comes under a supervised machine learning group. It is based on the concepts of probabilistic logics. The backbone of naïve bayes is bayes theorem. Bayes theorem is the form of mathematical probabilistic technique where the probability of an event is calculated. Naive bayes uncovers the set of probabilities. These probabilities get input to the training set and we get the predictions as the result of that. It uses probability to make predictions for each attribute from each class set. The data model which is yielded is called the predictive model with a probabilistic problem at the foundation. Each attribute in a class is independent and does not have any correlation among them.

4. RESULTS

The DREBIN dataset was used to conduct the experiments. Four types of Malware images were formed using files Certificate (CR), Android Manifest (AM), Resources (RS), and Classes.dex (CL). GIST descriptors were used to extract the features from the malware images. Four classifiers namely KNN, SVM, RF, and NB were used to perform the classification.

The classifier naive bayes did not perform well on CR malware images and show the classification accuracy of 47.1% as shown in Figure 2 . The precision and recall of naïve bayes are observed as 43.83% and 58.05% respectively. The classification accuracy of the classifiers KNN and RF are observed to be 80.1% and 82.92% on CR malware images. SVM is the top performer against the CR malware images files. SVM delivered the accuracy of 83.08%. The precision and recall of SVM are observed as 72.51% and 68.97% respectively.

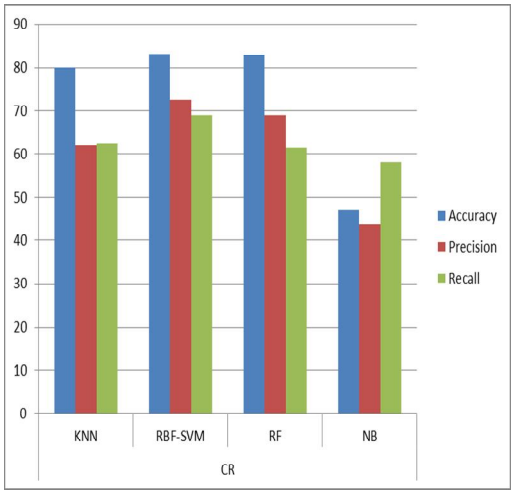


Figure 2 : Classification performance on CR malware images

In the next phase of experimentation, GIST features were extracted from malware images created using classes.dex (CL) files. The classification performance using four classifiers is depicted in Figure 3**.** The classifier naïve bayes showed the poor classification accuracy of 67.06%. The accuracy of KNN and SVM is observed to be 90.74% and 89.37% respectively. The RF obtain the highest classification accuracy of 91.06% with precision and recall of 90.68% and 82.82% respectively for CL malware images.

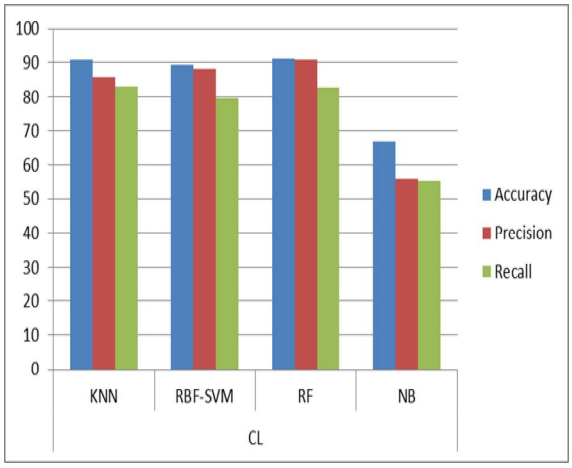


Figure 3 : Classification performance on CL malware images

The classification performance is also observed on RS malware images as shown in Figure 4. The RS malware

images were classified using GIST descriptors and classifiers. The classifier NB had attained the lowest accuracy of 45.73%. The classifiers RF, KNN, and SVM have achieved decent classification results with an accuracy of 88.14%, 87.88%, and 83% respectively.

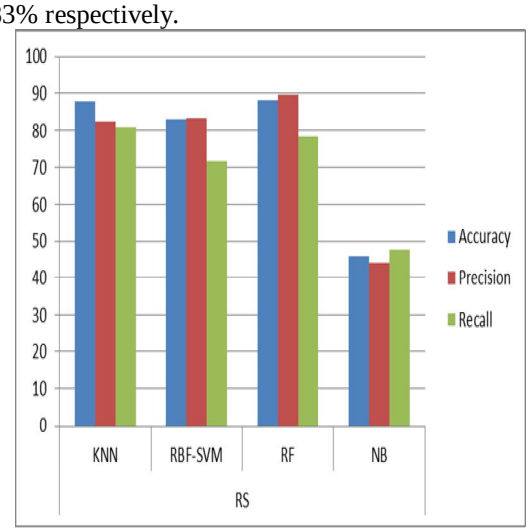


Figure 4 : Classification performance on RS malware images

The three classifiers namely SVM, KNN, and RF worked extremely well on malware images created using AM files as shown in Figure 5. SVM, KNN, and RF attained the accuracy of 92.7%, 92.38%, and 91.01% using the GIST descriptors. Even NB performed better on AM malware images. It attained an accuracy of 76.81%.

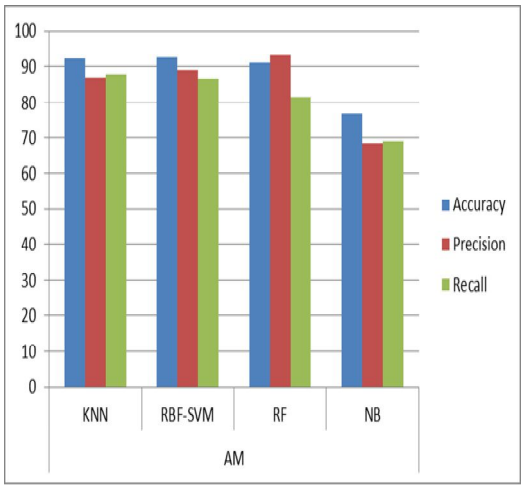


Figure 5 : Classification performance on AM malware images

In a nutshell, the combination GIST+NB could not perform well on any of the malware images used in this work. The maximum accuracy achieved by all classifiers is on AM malware images only. The RF achieved the highest precision of 93.16%. Therefore, it can be concluded that the AM file contains essential information for the classification of malware images.

5. CONCLUSION

In this work, Android malware images are created using different sections of malicious applications. Four types of malware images generated are – Android Manifest (AM), Classes.dex (CL), Resources (RS), and Certificate (CR). The GIST algorithm is used to extract the features from the malware images. To perform the classification various machine learning classifiers such as SVM, KNN, RF, and NB are used in this work. This paper compares the results obtained by different classifiers. The model GIST +SVM outperformed all other classifiers and attained the classification accuracy of 92.7% on AM malware images. In our future work, we will explore the other handcrafted features and deep learning techniques to classify the malware images.

REFERENCES

[1] D. Maiorca, D. Ariu, I. Corona, M. Aresu, and G. Giacinto, “Stealth attacks: An extended insight into the obfuscation effects on Android malware,” *Comput. Secur.*, vol. 51, pp. 16–31, 2015.

[2] D. Wermke, N. Huaman, Y. Acar, B. Reaves, P. Traynor, and S. Fahl, “A Large Scale Investigation of Obfuscation Use in Google Play,” 2018. https://doi.org/10.1145/3274694.3274726

[3] J. Crussell, C. Gibler, and H. Chen, “AnDarwin: Scalable Detection of Android Application Clones Based on Semantics,” *IEEE Trans. Mob. Comput.*, vol. 14, no. 10, pp. 2007–2019, 2015.

[4] Z. Ren, G. Chen, and W. Lu, “Malware visualization methods based on deep convolution neural networks,” *Multimed. Tools Appl.*, pp. 1–19, 2019.

[5] P. Faruki *et al.*, “Android security: A survey of issues, malware penetration, and defenses,” *IEEE Commun. Surv. Tutorials*, vol. 17, no. 2, pp. 998–1022, 2015.

[6] J. Fu, J. Xue, Y. Wang, Z. Liu, and C. Shan, “Malware visualization for fine-grained classification,” *IEEE Access*, vol. 6, pp. 14510–14523, 2018.

[7] D. Vasan, M. Alazab, S. Wassan, H. Naeem, B. Safaei, and Q. Zheng, “IMCFN: Image-based malware classification using fine-tuned convolutional neural network architecture,” *Comput. Networks*, vol. 171, no. April, p. 107138, 2020. https://doi.org/10.1016/j.comnet.2020.107138

[8] S. Yajamanam, V. R. S. Selvin, F. Di Troia, and M. Stamp, “Deep Learning versus Gist Descriptors for Image-based Malware Classification.,” in *ICISSP*, 2018, pp. 553–561.

[9] S. Ibrahim, M. H. C. Rozan, and N. Sabri, “Comparative analysis of support vector machine (SVM) and convolutional neural network (CNN) for white blood cells classification,” *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 8, no. 1.3, pp. 394–399, 2019.

https://doi.org/10.30534/ijatcse/2019/6981.32019

[10] W. Enck, M. Ongtang, and P. Mcdaniel, "On Lightweight Mobile Phone Application Certification ∗ † Categories and Subject Descriptors," *Proc. 16th ACM Conf. Comput. Commun. Secur. CCS '09, , New York, NY, USA, ACM.*, pp. 235–245, 2009.

[11] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos, "Paranoid Android: Versatile Protection For Smartphones," *Annu. Comput. Secur. Appl. Conf.*, pp. 347–356, 2010.

[12] T. Bläsing, L. Batyuk, A. D. Schmidt, S. A. Camtepe, and S. Albayrak, "An android application sandbox system for suspicious software detection," *Proc. 5th IEEE Int. Conf. Malicious Unwanted Software, Malware 2010*, pp. 55–62, 2010.

[13] A. Shabtai, U. Kanonov, Y. Elovici, C. Glezer, and Y. Weiss, "'Andromaly': A behavioral malware detection framework for android devices," *J. Intell. Inf. Syst.*, vol. 38, no. 1, pp. 161–190, 2012.

[14] 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," *Proc. 19th Annu. Netw. Distrib. Syst. Secur. Symp.*, no. 2, pp. 5–8, 2012.

[15] J. Yan, Y. Qi, and Q. Rao, "LSTM-based hierarchical denoising network for Android malware detection," *Secur. Commun. Networks*, vol. 2018, 2018. https://doi.org/10.1155/2018/5249190

[16] B. Dudi and D. V Rajesh, "Medicinal Plant Recognition Based On CNN And Machine Learning," *Int. J. Adv. Trends Comput. Sci. Eng. ISSN*, pp. 2278–3091, 2019.

[17] A. Abd Almisreb, N. Jamil, S. M. Norzeli, and N. M. Din, "Deep transfer learning for ear recognition: A comparative study," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, no. 1.1 Special Issue, pp. 490–495, 2020.

[18] D. Arp, M. Spreitzenbarth, M. Hübner, H. Gascon, and K. Rieck, "Drebin: Effective and Explainable Detection of Android Malware in Your Pocket," *Proc. 2014 Netw. Distrib. Syst. Secur. Symp.*, 2014. https://doi.org/10.14722/ndss.2014.23247