Research Article

Feature Engineering based on Hybrid Features for Malware Detection over Android Framework

Aqeel Nawaz [a], Jawad hussain [b] , NZ Jhanjhi [c] , MN Talib [d] ,Muhammad Nadim[e]

[a] Data scientist and Software Developer, Work as course Instructor of "Adv programming, Machine Learning and Data mining" a Kamyab Javan program in Pakistan, ORCID iD: https://orcid.org/0000-0001-8744-8841. Email: aqeelnawaz44@gmail.com
[b]Assistant Professor Arid Agriculture University Rawalpindi, Barani Institute of information technology. PhD degree in Multimedia Communication from the Massey University of New Zealand. Email: Syedjawad2k@gmail.com
[C]School of Computer Science and Engineering, SCE Taylor's University, Malaysia, email: noorzaman.jhanjhi@taylors.edu.my
[d] Papua New Guinea University of Technology, Lae, PNG, email: muhammad.talib@pnguot.ac.pg
[e] Barani Institute of Sciences Sahiwal Campus ARID Agriculture University Rawalpindi email: Muhammadnadim096@gmail.com

Article History: Received: 10 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 28 April 2021

Abstract: Android is the operating system of this modern world. Today, every tech-savvy people across the world are giving first preference to Android devices for their personal and official use. Because of the growing use of Android devices attackers are turning their attention toward android application. Because of this alarming increase in Android malware attacks there is a need to develop a defence mechanism against such attacks that must be fruitful and cost-effective. State-of-the-art malware detection techniques perform static, dynamic or hybrid analysis. Static analysis involves examining the source code malware samples without executing them. However, dynamic analysis monitors the run time behaviour of application during the actual execution of the app. Static analysis is a straightforward way to analyze the malware samples regarding the Android platform. In this research, we perform hybrid analysis using four different categories of Android application features such as permissions, intents, and network features. We extract permissions and intent from a manifest file while Network-based features extracted from java files. Our results show that the greatest precision of 0.99 can achieve by performing feature selection using Info Gain Method. Through, feature selection and results achieved by those selected features we come to know that permission are the most relevant features among all other three feature categories. We have observed that performing Ensemble method is best among all four machine learning classifiers. We have seen that network features (IP addresses, Email addresses, URL) are the relevant and effective feature for malware detection in the proposed framework.

Keywords: Malwares analysis, Dynamic analysis, Static analysis, Hybrid analysis, Android Security, cyber Security.

1. Introduction

Because of the exponential adaptation of mobile devices, having open-source operating systems (Android systems considered having captured 87.5% of the mobile phone market (Kharpal, 2016)), introducing malware applications within legitimate ones are also clear at an exponential rate. "Malware", a malicious code, designed with an unsafe intent and commonly floated on mobile application stores, giving a perception of being an ordinary and a safe app for use. They are injected or downloaded by users and installed in the mobile devices without suspect. They can be of different forms like; virus, Trojan horses or worms. A recent statistic reveals that there are approximately one million android based mobile applications which are malware (Wood, Nahorney,Chandrasekar, Wallace, & Haley, 2015). It shows another alarming statistics in a survey (cyber-crime report 2014) is that worldwide the financial damages caused by these malware applications exceed over 400 billion annually. These statistics make the malware application detection systems an intrinsic need of the hour.

Over the past few years, various approaches have been adapted to detect malware applications from the legitimate ones, which can grouped as static, a dynamic and a combination of the two called the hybrid approaches. We will discuss each one.

Static based approaches analyze the suspect application's code without executing or running it. Common features used by such approaches for malware detection are; Permissions and API calls (Almin & Chatterjee, 2015). Static approaches are very fast, requires minimum efforts, have lower costs in terms of time and budget. However they can only detect already known malware types (Rao & Hande, 2017).

Dynamic approaches as their name test and gives a deep analysis of the suspected application by executing it. It analyzes the behaviour of the application during runtime by checking its code and system calls (Canfora, Medvet, Mercaldo, & Visaggio, 2015). These approaches try to segregate malware based on the observations what sort of system calls used by normal applications versus abnormal ones. Such behaviour identifications can surely lead to

the detection of new malware and already identified ones. Although dynamic approaches are far better than the static ones, requires a large storage space, high computational power and are time-consuming (Rao & Hande, 2017).

Hybrid approaches make use of both dynamic and static approaches' analysis procedure to identify and segregate malware from normal applications. They combine dynamic and static approaches features to identify the malware and have a high accuracy rate, but the entire process makes the analysis, time and computationally expensive. Also, inherit traditional static and dynamic approaches limitations (Xu, Zhang, Jayasena, & Cavazos, 2016).

Among the aforementioned approaches, hybrid approaches are considered being more precise however their limitations and the increasing number of new malware pose new challenges which need to addressed by researchers. Considering these observations, the focus of the research is on devising a hybrid approach which tries to detect malware applications by minimizing the known limitations (of the static and dynamic approaches) and achieve high accuracy. Though a lot of research conducted and a lot of hybrid approaches exists, however to the best of our knowledge, a combination of URL, emails, IPs and text features have not used. The proposed strategy uses static and dynamic features stepwise to test, identify, reevaluate and reconfirm for malware identification. The proposed approach is a supervised learning approach which classifies the application as malware or clean class.

2.Review of Related Studies

Android security issues especially identification of malware from legitimate applications have become a hot research area, since the exponential use of smart phones worldwide. Various techniques have been proposed for identifying malware using static features, dynamic features or combination of both of them, having their own benefits and limitations.

1) Techniques based on static analysis

As mentioned above, static analysis of applications is a procedure to analyze the functionalities and behaviours through its source code without actually executing it. A technique proposed by (Sato, Chiba, & Goto, 2013) analyzes only the manifest files of an android application. The technique extracts specific information from the manifest.xml file of an app and then look up this information in the keyword lists that are built by them gradually. A malignancy score is computed based on the manifest file information with respect to the keyword list. If the malignancy score is higher than the threshold the app is considered to the malicious otherwise non-malicious. The results shown by the experiments had 91.4\% accuracy rate for benign application recognition whereas 87.5\% accuracy rate against malware detection. Whereas 90.0\% is the overall total accuracy rate. The technique, as evident is unable to detect much malware, due to its static nature.

(Feizollah et al., 2017) discusses the value of an applications Intents. The authors are of a view that intents are powerful features that can be used to encode the malicious intentions of malware, especially when used in combination with permissions. The proposed technique detects an application as a malware using number of different modules like; decompiler, extractor and decision-maker. Decompiler decomposes the apk files, extractor extracts intents, intents filters & permissions from the Java code. Whereas the intelligent learner & decision-maker use features database to extract data use them in Bayesian network algorithm to learn patterns and decide whether the application is malicious or not. The authors used DREBIN(Arp et al., 2014) as a malware dataset, consisting of 5560 applications, belonging to one of 179 malware family. The dataset also contains 1846 clean applications. The technique achieved 91% detection rate using application's Intents while 83% using android permissions. While combining both of them achieved 95.5% accuracy, concluding that intents based detections can be considered as a more reliable detection scheme.

02)Techniques based on dynamic analysis

Besides Google, the research community of this field has put a lot into this as well. As, (Canfora et al., 2015) tried to identify malware based on system call. They are of the view that system calls are the basis of an application's malicious behaviour. They used SVM classifier to identify the specific sequence calls which are associated with malware. The experimentation was conducted on 2000 applications and achieved 97% detection rate. In another study conducted by (Wong & Lie, 2016), the authors proposed IntelliDroid which is based on specific API calls. The proposed tool is flexible enough to integrate itself with other tools and analyze the behaviour of an application. The experimentations are conducted on 75 malware, among which 70 were successfully traced.

(Kiet al.,2015) used API call sequence to analyze the malicious behaviours of the applications. The authors used a hooking process which monitors and keeps track of the API call sequence. These API call sequences are then compared with the already stored ones and generate a security alert upon suspect.

(Alzaylaee, Yerima, & Sezer, 2016) the proposed system called DynaLog extracts features like logging of high-level behaviour and API calls during the analysis phase. The authors used a mix of 2226 malware and clean apps. The application is dependent on a Google tool Monkey for application testing. The proposed application methodology is convincing but is dependent on Monkey tool, also, those applications which are unable to run in emulated environment will remain unchecked.

03) Hybrid Techniques

Hybrid malware analysis techniques use a combination of both dynamic & static features to detect malware applications. It is a relatively new aspect of the solution and many researchers have started focusing on this premise. The researchers use different features from the static and dynamic feature pools and come up with many diverse and effective schemes for malware identification. Some of them are highlighted in this section.

AASandbox (Android Application Sandbox) proposed by (Bläsing, Batyuk, Schmidt, Camtepe, & Albayrak, 2010) is another hybrid approach which uses (.dex files) for the static analysis. Whereas low-level details of system interactions are used for dynamic analysis purpose. During the static analysis the .dex files are decompiled into a human-readable format which is then scanned further for suspicious patterns. During the dynamic analysis, the applications low-level details are used generated during the execution of the application within the sandbox environment. As it is known sandbox environment is used to ensure the security of analyzing a system and to ensure the safety of data. For the dynamic analysis the technique also observes the behaviour of an application through producing random events.

(Zhao, Li, Xu, Zhang, & Feng, 2014) proposed a hybrid malware detection technique named AMDetector. The technique uses a modified attack tree model (Schneier, 1999) which takes static features to get necessary information about an application. This information is then used by the classifier to classify the applications into normal or harmful classes. The application behaviour which triggers different code components of an application is also considered which are the basis of dynamic analysis. Making use of organized rules (with attack trees) achieve good code coverage and accuracy of up to 96.5\% by this technique, however the manual rule generations and dynamic analysis are time costly.

Another hybrid technique proposed by (Yuan, Lu, Wang, & Xue, 2014) used deep learning to classify Android malware using Droid-Sec. the technique selects over 200 dynamic & static features of an application and uses these features for classification purpose to the deep neural network. Experiments were conducted on 599 applications which were a mixture of both malicious and benign sample having no class imbalance problem. The technique achieved 96.5% accuracy.

(Yuan, Lu, & Xue, 2016), hybrid, deep learning-based technique, uses different kernels and graph kernels to develop a deep neural network, however, the technique is costly in terms of time & training.

(N. Zaman and F. A. Almusalli, 2017), (N. Z. Jhanjhi, et. al, 2018), and (F. A. Almusalli, et. al, 2017), Malware caused several issues, and it effect mainly power savaging capabilities of Android phone as well. (Humayun, M. et. al, 2020), and (B. Hamid, et. el., 2019), further elaborate the smart efficient and security issues of different smart phone application for smart environment and smart cities. (Seungjin, L., et. al, 2020), and (Lee S, et.al, 2021) further consider the security issues and security attacks on smart factory and in smart home.

3. Methodology

The proposed system will be able to classify the variety and android malware. In static mode, I used android intents and permissions as the basic feature for malware classification. And in dynamic mode, I used URL,Email and IPs as the basic features of malware classification. . The Decompiler takes an APK file and decodes it into readable components. Every APK file has various parts like Java files XML files and manifest file it's decoded every part and make it readable. The extractor is a module which extracts different type information which is needed for malware detection like Intents and permissions. Androguard is used to reverse dex file and beautiful soup package to extract permissions and intent from manifest file. Similarly its extracts network features (IP addresses, email addresses, and URL) from dissembled dex files and stores them into .txt files. then the resulted txt files used to make feature vectors. then different classifer like Random forest(RF), Naive Bayes(NB), Gradient Boosting(GB) and ada boosting(AB) applied on these feature vectors.At the end result will be shown weather the application is malware or clean.

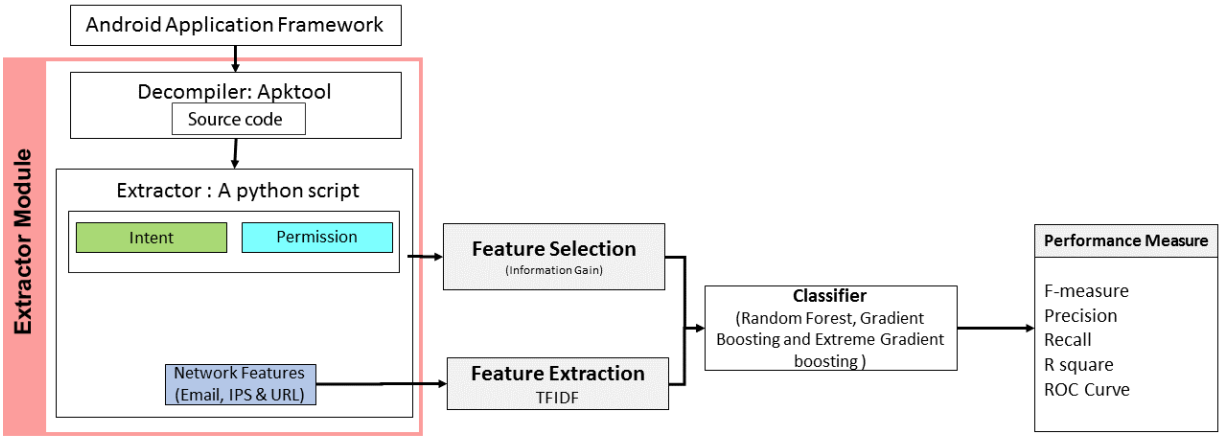

Figure 1 Context diagram of proposed system

A. Feature Extraction:

Android applications are packaged into .apk files. APK means (Android Package Kit). It is format of file that is used to distribute applications on the Android Operating System. APK files are generally compressed files that can be downloaded directly on Android gadgets through Google play store or third-party app stores. APK files consist of some of the files and folders such as META-INF folder, res folder, resource.arsc files, AndroidManifest.xml files, and classes.dex files as shown in Figure 3.1. META-INF folder stores Meta information about the contents of the JAR. This information can be sometimes stored in another folder named original.

Res folder contains resources Resources.arsc files contain precompiled application resources such as strings, colors, and styles in binary XML. Android Manifest.xml file is binary XML file format. This contains application metadata such as name, version, intents, permissions, etc. The classes.dex files contain application code compiled in the dex format. For feature extraction we have used a python feature extraction script. This Feature extraction script takes the APK file, disassemble it into classes.dex files, AndroidManifest.xml, read permissions and intents tags from AndroidManifest.xml file extract them and store in .txt files. Similarly it extracts API calls and Network features (IP addresses, email addresses, and URL) from dissembled dex files and stores them into .txt files. These text files are further used for feature vector creation. The detail working of feature extraction script is as follows.

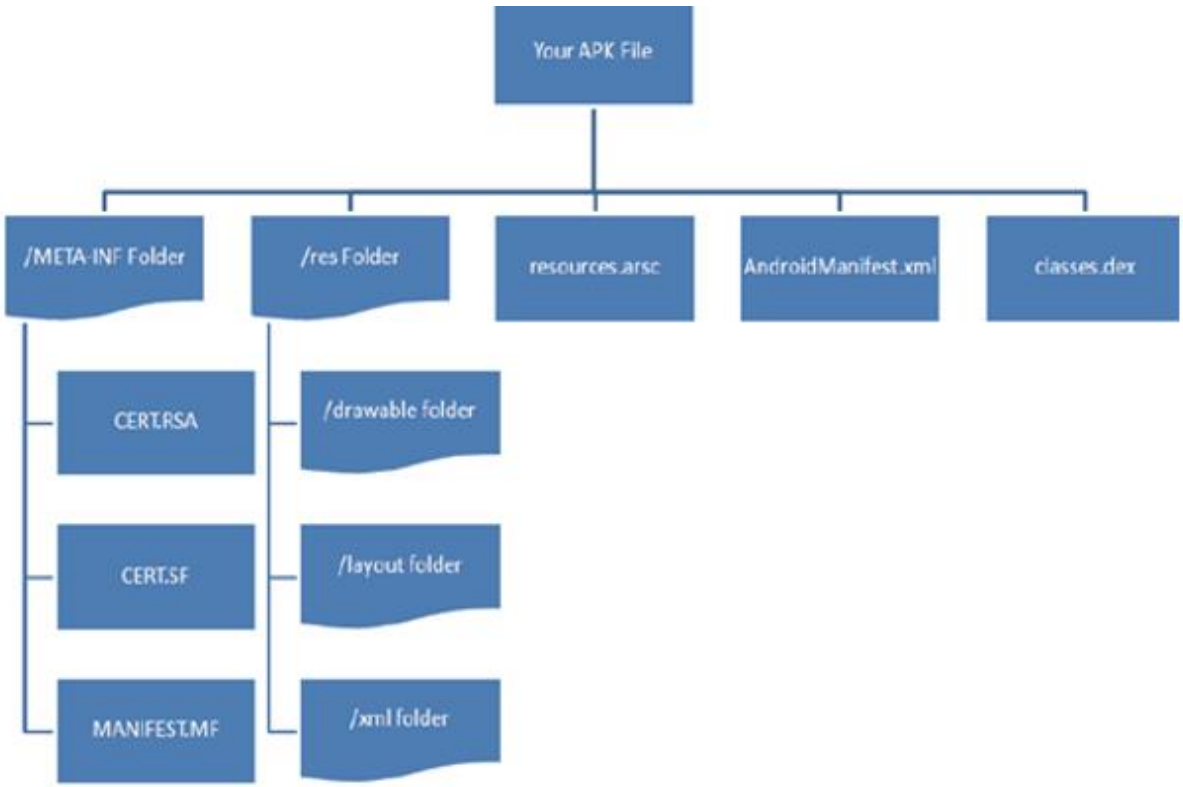

Figure 2 Context diagram of proposed system

B. Feature Selection:

One of the big challenges in creating a machine learning model is figuring out which of the data should be incorporated into the model. To achieve a better prediction it's not necessary and not even desired to use the whole data. Since a portion of data may have no or little influence on results. It may be even detriment to the computing results. Feature selection is a process that of selecting a subset of features from large feature for use in model construction that helps model to perform efficient prediction. And remove irrelevant and redundant features. Top reasons for using feature selection are:

- It is essential for machine learning classifier to train faster.
- Reduces the model complexity.
- By choosing Right Subset its Improves the accuracy of a model.
- Reduce over-fitting.

Here for the purpose of feature selection, we used Information gain method.

C. Model Selection:

All the extracted features (Permissions, intents, API calls and network-based features) are used for training machine learning classifiers. Selection of appropriate machine learning classifier is the most critical part of this research. During literature analysis, we have observed that previous studies suggested various machine learning classifiers such as random forest(RF), Gradient Boosting(GB) ,Naive Bayes(NB) and ada Gradient Boosting etc., on the bases on their obtained results. We used these three machine learning algorithms for experimentation to choose the best performing algorithm for the proposed Android malware detection and classification framework

D. Model Training and Testing:

After getting feature selection and model selection. The model is trained with an ensemble of three model and their combinations. The model is trained using sci-kit learn library and language is Python. We used 80:20 ratio. 80% for training dataset and 20% for testing data set.

E. Model Performance Measure:

To evaluate the classification, we use standard metrics i.e. recall f-measure and precision. The precision is the portion of correct positive classification (true positives) from cases that are predicted as positive. The recall is the portion of correct positive classification (true positive) from cases that are actually positive. F measure is the harmonic mean of recall & precision.

4. Experimental results and discussion

A. Data set:

The data set used for experiments consists of Android malware and benign applications in the form of .apk files. Data set of Android malware is collected from Drebin [15]. The Drebin data set consists of 5,560 samples belonging to 179 different malware families. The data set of the benign application was randomly collected from Google Play store. Our whole experimental data set consists of 5560 Android malware apps and 1980 clean apps. 80\% malware and benign applications are used for training and 20\% applications are used for testing.

B. Performance of All Features:

1) Permission

The variation in precision, recall and F-measure of permission data set against classifiers are shown in table 1 The Table describes that Gradient Boosting and Random Forest has the highest values of performance measures Precision, recall and F-measure which are 0.98, 0.98, and 0.98 respectively and the R-Square value 0.91 for both classifiers. While Nive Bayes and Ada boosting Classifier produce 0.96 values of Precision, Recall and F-measure.

Classifier	R_sqaure	Fmeasure	Precision	Recall
Random Forest	0.91	0.98	0.98	0.98
Naive Bayes	0.80	0.96	0.96	0.96
Gradient Boosting	0.91	0.98	0.98	0.98
Ada Boosting	0.83	0.96	0.96	0.96

Table 1 Result of Permissions features by applying different classifier

The training and testing of the best model are shown in figure 3. I used 70:30 70% for the training and the 30% for the testing data.

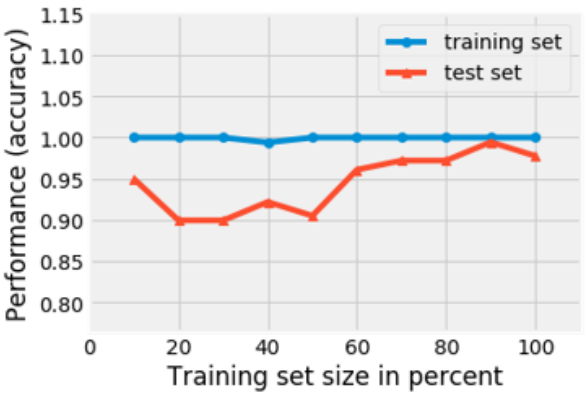

Figure 3 Result of Permissions Training set size

2) Intent

Table 2 Performance Measures of Intents using full data. We can see the performance of classifiers for Intents. Table 2 shows that F-measure and recall of Naive Bayes is produce best result 0.93 which is highest among all. While the other classifier F-measure values of Random Forest, Gradient Boosting and ada boosting are respectively 0.92,0.92 and 0.91.

Classifier	R_sqaure	Fmeasure	Precision	Recall
Random Forest	0.60	0.90	0.90	0.90
Naive Bayes	0.71	0.93	0.92	0.93
Gradient Boosting	0.64	0.91	0.91	0.92
Ada Boosting	0.60	0.90	0.90	0.90

Table 2 Result of Intent features by applying different classifier

The Training and testing of the best model is shown in figure 4. I used 70:30 combination 70 percent for the training and 30 percent for testing data.

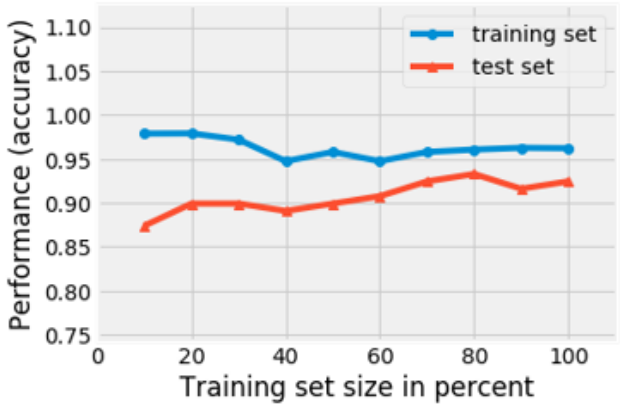

Figure 4 Result of Intent training set size

03) Network

Table 3 Performance Measures of the network using full data. We can see the performance of classifiers for networks. Table 3 shows that F-measure value of both Random Forest and Ada Boosting classifier is 0.97 which is highest value. While the naive bayes and the Gradient boosting produce 0.93 F-measure values. For the network Features Random Forest and the Ada Boosting perform best from others.

Classifier	R_sqaure	Fmeasure	Precision	Recall
Random Forest	0.89	0.97	0.97	0.97
Naive Bayes	0.86	0.96	0.96	0.96
Gradient Boosting	0.86	0.96	0.96	0.96
Ada Boosting	0.89	0.97	0.97	0.97

Table 3 Result of Network features by applying different classifier

The Training and testing of the model is shown in figure 5. I used 70:30 ratios. The 70% for training data set and 30% for testing data set.

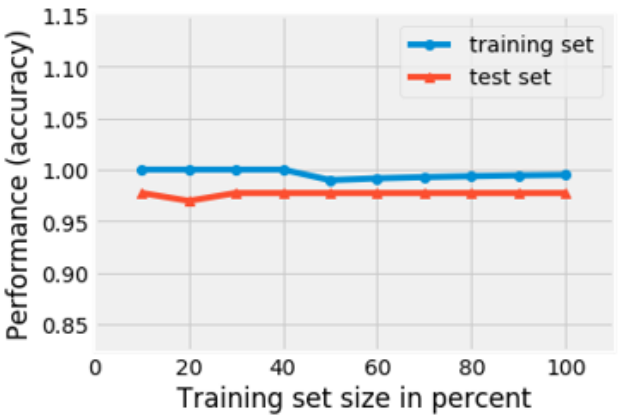

Figure 5 Result of Permissions Training set size

5. Conclusion and Future work

Around, 80% of Smartphone worldwide users have a Android operating system device. The high dimension rate has a direct connection with an increase in the development of Android malware. This makes it more important to use a security mechanism for protecting Android devices from known or zero-day malware attacks. In this scenario, researchers have proposed many malwares detection approaches. These approaches use static analysis features (Permissions, Text classification, opcode sequences, control flow graphs etc.) and dynamic analysis features (System call logs, CPU and memory usage etc.) or both of them. Static analysis is a more efficient and cost-effective approach that can use for Android malware detection. From the comprehensive literature analysis of state-of-the-Art Android malware detection approaches, we concluded that there is a lack of an Android malware detection approach based on the analysis that uses permissions, intents, Network features (IP addresses, Email addresses, URL) collectively on the same dataset in one approach.

In this research work, we dissembled the APK files of Android malware and benign applications and performed a static analysis on AndroidMenifest.xml files to extract intents and permissions. We extracted Network-based features from java files. Afterword we transformed all these extracted features into feature vectors for training & testing of machine learning algorithms. In this research, we have used four machine learning algorithms: Random forest(RF), Naive Bayes(NB), Ada Boosting , Gradient Boosting(GB). To reduce the dimensionality of features data, we performed feature selection processes using dimensionality reduction algorithms: Info Gain method. We have compared the performance of these feature selection algorithms for Android malware detection by generating separate feature vectors of permissions, intents, and Network features and training all three machine learning classifiers on four feature vectors. We have seen from the results got that info is a best feature selection method. it can select more effective features helpful in malware detection.. The results displayed this that Random Forest and Gradient Boost is the best performing classifier that classified Android malware samples with high F-measure values for the permission features data. While for the intent features data naïve bayes perform best and the network features Random forest and Ada Boosting is. This research work suggests that networks features are the most relevant features that help in malware detection as compared to the other Android application features used in this research.

References (APA)

[1] Almin, S. B., & Chatterjee, M. (2015). A novel approach to detect android malware. Procedia Computer Science, 45, 407–417.

[2] Alzaylaee, M. K., Yerima, S. Y., & Sezer, S. (2016). DynaLog: An automated dynamic analysis framework for characterizing android applications. 2016 International Conference On Cyber Security And Protection Of Digital Services (Cyber Security), 1–8.

[3] Arora, A., Peddoju, S. K., Chouhan, V., & Chaudhary, A. (2018). Poster: Hybrid Android Malware Detection by Combining Supervised and Unsupervised Learning. Proceedings of the 24th Annual International Conference on Mobile Computing and Networking, 798–800..

[4] Breiman, L. (1997). Arcing the edge.

[5] B. Hamid, N. Jhanjhi, M. Humayun, A. Khan and A. Alsayat, "Cyber Security Issues and Challenges for Smart Cities: A survey," 2019 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), 2019, pp. 1-7, doi: 10.1109/MACS48846.2019.9024768.

[6] Bläsing, T., Batyuk, L., Schmidt, A.-D., Camtepe, S. A., & Albayrak, S. (2010). An android application sandbox system for suspicious software detection. 2010 5th International Conference on Malicious and Unwanted Software, 55–62.

[7] Canfora, G., Medvet, E., Mercaldo, F., & Visaggio, C. A. (2015). Detecting android malware using sequences of system calls. Proceedings of the 3rd International Workshop on Software Development Lifecycle for Mobile, 13–20.

[8] Choudhary, M., & Kishore, B. (2018). HAAMD: Hybrid Analysis for Android Malware Detection. 2018 International Conference on Computer Communication and Informatics (ICCCI), 1–4.

[9] Dash, S. K., Suarez-Tangil, G., Khan, S., Tam, K., Ahmadi, M., Kinder, J., & Cavallaro, L. (2016). Droidscribe: Classifying android malware based on runtime behavior. 2016 IEEE Security and Privacy Workshops (SPW), 252–261.

[10] David, O. E., & Netanyahu, N. S. (2015). Deepsign: Deep learning for automatic malware signature generation and classification. 2015 International Joint Conference on Neural Networks (IJCNN), 1–8.

[11] Faruki, P., Ganmoor, V., Laxmi, V., Gaur, M. S., & Bharmal, A. (2013). AndroSimilar: robust statistical feature signature for Android malware detection. Proceedings of the 6th International Conference on Security of Information and Networks, 152–159.

[12] F. A. Almusalli, N. Zaman and R. Rasool, "Energy efficient middleware: Design and development for mobile applications," 2017 19th International Conference on Advanced Communication Technology (ICACT), 2017, pp. 541-549, doi: 10.23919/ICACT.2017.7890149.

[13] Feizollah, A., Anuar, N. B., Salleh, R., Suarez-Tangil, G., & Furnell, S. (2017). Androdialysis: Analysis of android intent effectiveness in malware detection. Computers & Security, 65, 121–134.

[14] Hussain, S. J., Ahmed, U., Liaquat, H., Mir, S., Jhanjhi, N. Z., & Humayun, M. (2019). IMIAD: Intelligent Malware Identification for Android Platform. 2019 International Conference on Computer and Information Sciences (ICCIS), 1–6.

[15] Humayun, M., Jhanjhi, N. Z., & Alamri, M. Z. Smart Secure and Energy Efficient Scheme for E-Health Applications using IoT: A Review. International Journal of Computer Science and Network Security, 20(4), 55-74.

[16] Ho, T. K. (1995). Random decision forests. Proceedings of 3rd International Conference on Document Analysis and Recognition, 1, 278–282.

[17] Idrees, F., & Rajarajan, M. (2014). Investigating the android intents and permissions for malware detection. 2014 IEEE 10th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob), 354–358.

[18] Kharpal, A. (2016). Google Android hits market share record with nearly 9 in every 10 smartphones using it. Retrieved from https://www.cnbc.com/2016/11/03/google-android-hits-market-share-record-with-nearly-9-in-every-10-smartphones-using-it.html.

[19] Ki, Y., Kim, E., & Kim, H. K. (2015). A novel approach to detect malware based on API call sequence analysis. International Journal of Distributed Sensor Networks, 11(6), 659101.

[20] Kim, J., Yoon, Y., Yi, K., Shin, J., & Center, S. (2012). ScanDal: Static analyzer for detecting privacy leaks in android applications. MoST, 12(110), 1.

[21] Lantz, P., Desnos, A., & Yang, K. (2012). DroidBox: Android application sandbox. 2014-10-08]. Https://Code. Google. Eom/p/Droidbox.

[22] Maiorca, D., Ariu, D., Corona, I., Aresu, M., & Giacinto, G. (2015). Stealth attacks: An extended insight into the obfuscation effects on android malware. Computers & Security, 51, 16–31.

[23] Narudin, F. A., Feizollah, A., Anuar, N. B., & Gani, A. (2016). Evaluation of machine learning classifiers for mobile malware Oberheide, J., & Miller, C. (2012). Dissecting the android bouncer. SummerCon2012, New York, 12.

[24] N. Zaman and F. A. Almusalli, "Review: Smartphones power consumption & energy saving techniques," 2017 International Conference on Innovations in Electrical Engineering and Computational Technologies (ICIEECT), 2017, pp. 1-7, doi: 10.1109/ICIEECT.2017.7916593.

[25] N. Z. Jhanjhi, F. A. Almusalli, S. N. Brohi and A. Abdullah, "Middleware Power Saving Scheme for Mobile Applications," 2018 Fourth International Conference on Advances in Computing, Communication & Automation (ICACCA), 2018, pp. 1-6, doi: 10.1109/ICACCAF.2018.8776711.

[26] Rao, V., & Hande, K. (2017). A comparative study of static, dynamic and hybrid analysis techniques for android malware detection. Int. J. Eng. Dev. Res.(IJEDR), 5, 1433–1436.

[27] Sato, R., Chiba, D., & Goto, S. (2013). Detecting android malware by analyzing manifest files. Proceedings of the Asia-Pacific Advanced Network, 36(23–31), 17.

[28] Seungjin, L., Abdullah, A., & Jhanjhi, N. Z. (2020). A Review on Honeypot-based Botnet Detection Models for Smart Factory. International Journal of Advanced Computer Science and Applications, 11(10.14569).

[29] Lee S, Abdullah A, Jhanjhi N, Kok S. 2021. Classification of botnet attacks in IoT smart factory using honeypot combined with machine learning. PeerJ Computer Science 7:e350 https://doi.org/10.7717/peerj-cs.350