ATLANTIS PRESS

Advances in Engineering Research, volume 119

2nd International Conference on Automatic Control and Information Engineering (ICACIE 2017)

Detecting Unknown Malware on Android by Machine Learning Using the Feature of Dalvik Operation Code

Wang Quanmin
Department of Information Technology,
Beijing University of Technology
Beijing, China
wangqm@bjut.edu.cn

Li Zhenguo
Department of Information of Technology,
Beijing University of Technology
Beijing, China
13120006042@163.com

Zheng Shuang, Gu Shi, Sun Yanfeng, Wang Kaiyang
Department of Information of Technology,
Beijing University of Technology
Beijing, China

***Abstract*—The recent growth in network usage has motivated the creation of new malicious code for various purposes, including economic ones. Today's signature-based anti-viruses are very accurate, but cannot detect new malicious code. Recently, classification algorithms were employed successfully for the detection of unknown malicious code. However, most of the studies use byte sequence n-gram representation of the binary code of the executable files on windows. We propose the use of Dalvik Operation Code on Android, generated by disassembling the application. We then use n-gram of the operation code as features for the classification process. We present a full methodology for the detection of unknown malicious code, based on text categorization concepts. The experiment results show that the method results are in a high accuracy rate.**

Keywords—malicious; Dalvik operation code; detection

I. INTRODUCTION

Anti-virus skill is mainly based on two methods: signature-based methods, specialists only determine a signature for a new malware application after it had damaged computers [24]. So, they are useless against unknown malicious code. The second approach involves heuristic-based methods, which are based on rules defined by experts that define a malicious behavior, or a benign behavior, in order to enable the detection of unknown malicious codes. Other proposed methods include behavior blockers, which attempt to detect sequences of events in operating systems, and integrity checkers, which periodically check for changes in files and disks [6]. However, besides the fact that these methods can be bypass by viruses, their main drawback is that, by definition, they can only detect the presence of a malicious code after it has been executed [10].

A great amount of research applying the anomaly detection approach for finding malware files is conducted nowadays [26]. In such studies, usually byte sequences of executable binaries are analyzed to find deviations from benign software behavior [5]. Study proposes a method to analyze the binary content of files using n-gram analysis and efficient statistical modeling techniques in order to determine the validity of file type in network traffic flows or on a local disk [9]. In paper [11], byte sequence frequencies are investigated to profile benign software binaries and identify malicious executables as outliers or anomalies. The investigation uses the principal component analysis and a one-class support vector machine, without predefining the malicious patterns to be classified. A detection model based on byte frequency to deal with the problem of malware variants detection is proposed in [11]. The authors claim that if a suspicious malware is similar to any known malware in terms of byte frequency the suspicious malware is determined to be a variant of the latter.

Recent studies have investigated the ability of operational codes (opcodes) to detect malicious software [12-14]. An opcode is the portion of a machine language instruction that specifies the operation to be performed. Opcodes reveal significant statistical differences between malware and legitimate software and even single opcodes are able to serve as the basis for the detection of malicious executables [12]. In[13], detection of unknown malicious code is based on previously seen examples and carried out with the help of opcode n-gram representation and several well-known classifiers. A new method for unknown malware detection using a data-mining-based approach is proposed in [14]. The method is based on the frequency of appearance of opcode-sequences and on such data-mining algorithms as decision trees, support vector machines, knearest neighbors and Bayesian networks [21][29].

In this study, we come up with a methodology for malicious code categorization based on concepts from text categorization.

The rest of the paper is organized as follows. The extraction and selection of feature is considered in Section II. Section III introduces the algorithm of random forest and scikit-learn tool. In Section IV, we present several simulation results to evaluate the algorithm proposed with ROC. Finally, Section V draws the conclusions.

II. Feature Extraction and Selection

A. Data Creation

We create a dataset of malicious and benign Android application package(APK). The malicious files are acquired from VirusShare website. The benign files are downloaded from one mobile application of yingyongbao. An APK file is an archive that usually contains the following files and directories: META-INF, lib, res, res, assets, AndroidManifest.xml, resources. arsc and calss.dex. Among the files, class.dex contains classes compiled in the dex file format understandable by the Dalvik virtual machine. To acquire the smali files, we disassembled the class.dex by Apktool [23][27][28]. Using python, we merge whole. smali files in the directory of smali.

B. N-gram Models

In the fields of computational linguistics and probability, an n-gram is a contiguous sequence of n items from a given sequence of text or speech. The items can be phonemes, syllables, letters, words or base pairs according to the application. The n-grams typically are collected from a text or speech corpus. When the items are words, n-grams may also be called shingles [2-4].

An n-gram of size 1 is referred to as a "unigram"; size 2 is a "bigram" (or, less commonly, a "digram"); size 3 is a "trigram". Larger sizes are sometimes referred to by the value of n, e.g., "four-gram", "five-gram", and so on.

To extract features from each such sequence, n-gram model is applied. N-gram models are widely used in statistical natural language processing and speech recognition. An ngram is a sub-sequence of n overlapping items (characters, letters, words, etc) from a given sequence[15-16]. For example, the result of the application of 2-gram character model to the string “benign” is “be”, “en”, “ni”, “ig”, “gn”. Specially, we are dealing with this as “Table.1”.We use the frequency of the term frequency (TF) to represent the feature.

TABLE I. N-GRAM MODEL

N-Gram Model	Operation Code
Opcode name	iput-object vx, vy, field_id, iget-object vx, vy, field_id, if-eq vx, vy, target, move-result vx, return-object vx, goto target
Classfy and describe	P(save data), V(get data), I(if), M(move), R(return), G(jump)
2-Gram	PV, VI, IM, MR, RG

C. Feature Selection

In ML applications, the large number of features (many of which do not contribute to the accuracy and may even decrease it) in many domains presents a significant problem [17][25]. In our study, the reduction of the number of features is crucial and must be performed while maintaining a high level of accuracy. This is due to the fact that the vocabulary size may exceed millions of features; far more than can be processed by any feature selection tool within a reasonable period of time. Additionally, it is important to identify the terms that appear in most of the files in order to avoid vectors that contain many zeros. Thus, we extracted the 500 features (i.e., OpCode n-grams patterns) with the highest Document Frequency values and on which three feature selection methods were later applied.

III. Algorithm

A. Random Forest

The first algorithm for random decision forests was created by Tin Kam Ho [1] using the random subspace method,[8] which, in Ho's formulation, is a way to implement the "stochastic discrimination" approach to classification proposed by Eugene Kleinberg [10-12].

An extension of the algorithm was developed by Leo Breiman and Adele Cutler,[8] and "Random Forests" is their trademark.[15] The extension combines Breiman's "bagging" idea and random selection of features, introduced first by Ho[1] and later independently by Amit and Geman[16] in order to construct a collection of decision trees with controlled variance.

The random forests algorithm (for both classification and regression) is as follows:

- Draw ntree bootstrap samples from the original data.
- For each of the bootstrap samples, grow an unpruned classification or regression tree, with the following modification: at each node, rather than choosing the best split among all predictors, randomly sample mtry of the predictors and choose the best split from among those variables. (Bagging can be thought of as the special case of random forests obtained when mtry = p, the number of predictors.)
- Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression).

An estimate of the error rate can be obtained, based on the training data, by the following:

- At each bootstrap iteration, predict the data not in the bootstrap sample (what Breiman calls “out-of-bag”, or OOB, data) using the tree grown with the bootstrap sample.
- Aggregate the OOB predictions. (On the average, each data point would be out-of-bag around 36% of the times, so aggregate these predictions.) Calculate the error rate, and call it the OOB estimate of error rate.

In random forests each tree in the ensemble is built from a sample drawn with replacement (i.e., a bootstrap sample) from the training set [22]. In addition, when splitting a node during the construction of the tree, the split that is chosen is no longer the best split among all features [19]. Instead, the split that is picked is the best split among a random subset of the features.

As a result of this randomness, the bias of the forest usually slightly increases (with respect to the bias of a single non-random tree) but, due to averaging, its variance also decreases, usually more than compensating for the increase in bias, hence yielding an overall better mode [1][18].

B. Scikit-learn

Finally we use random forest algorithm by scikit-learn: simple and efficient tools for data mining and data analysis.

The main parameters to adjust when using these methods is n_estimators and max_features. The former is the number of trees in the forest. The larger the better, but also the longer it will take to compute. In addition, those results will stop getting significantly better beyond a critical number of trees. The latter is the size of the random subsets of features to consider when splitting a node. The lower the greater the reduction of variance, but also the greater the increase in bias. Empirical good default values are max_features=n_features for regression problems, and max_features=sqrt(n_features) for classification tasks (where n_features is the number of features in the data). So, we select the latter. Good results are often achieved when setting max_depth=None in combination with min_samples_split=1 (i.e., when fully developing the trees). Here, max_depth = 100. Bear in mind though that these values are usually not optimal, and might result in models that consume a lot of RAM. The best parameter values should always be cross-validated. In addition, note that in random forests, bootstrap samples are used by default (bootstrap=True) while the default strategy for extra-trees is to use the whole dataset (bootstrap=False). When using bootstrap sampling the generalization accuracy can be estimated on the left out or out-of-bag samples. This can be enabled by setting oob_score=True.

IV. NUMERICAL RESULT

A. Feature Representation vs. N-Grams

We first wanted to find the best terms representation (i.e., TF or TFIDF) [20]. Figure 1 and Figure 2 presents the mean ROC of the combinations of the term representation and n-grams size. Through calculating the AUC of ROC, the feature representation by TF was better, which is good because maintaining the TFIDF requires additional computational efforts each time a malcode or benign files are added to the collection. Following this observation we opted to use the TF representation for the rest of our experiments.

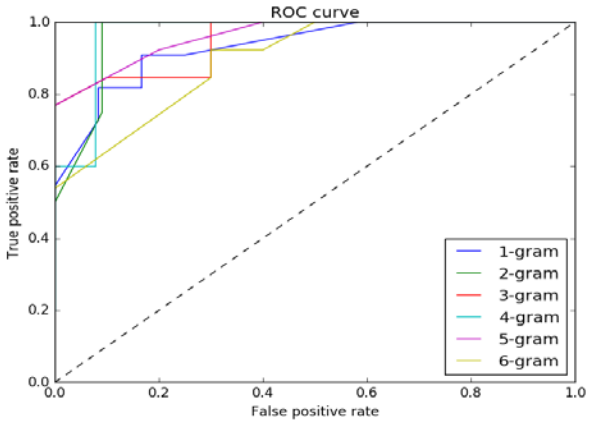

Fig. 1. Feature Representation by TF

B. Feature Selections and Top Selections

To identify the best feature selection method and the top number of features, we calculated the mean TPR, FPR, accuracy of DF and FS [30]. Generally, DF outperformed on all sizes of top features. The DF is a simple feature selection method which favors features which appear in most of the files. This can be explained by its criterion, which has an advantage for fewer features. In the other method, the lack of appearances in many files might create zeroed vectors and might consequently lead to a lower accuracy level.

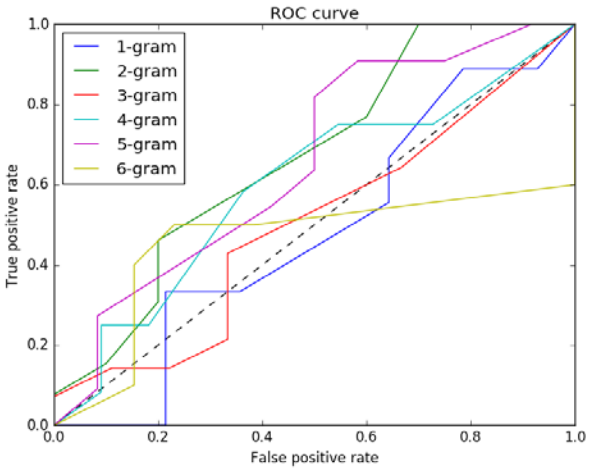

Fig. 2. Feature Representation by TF-IDF

C. Classify

1) Decision tree learning

Decision trees are a popular method for various machine learning tasks. Tree learning "come[s] closest to meeting the requirements for serving as an off-the-shelf procedure for data mining", say Hastie et al., because it is invariant under scaling and various other transformations of feature values, is robust to inclusion of irrelevant features, and produces inspectable models. However, they are seldom accurate.

2) Tree bagging

The training algorithm for random forests applies the general technique of bootstrap aggregating, or bagging, to tree learners. Given a training set X = x1, ..., xn with responses Y = y1, ..., yn, bagging repeatedly (B times) selects a random sample with replacement of the training set and fits trees to these samples:

For b = 1, ..., B:

Sample, with replacement, n training examples from X, Y; call these Xb, Yb.

Train a decision or regression tree fb on Xb, Yb.

After training, predictions for unseen samples x' can be made by averaging the predictions from all the individual regression trees on or by taking the majority vote in the case of decision trees.

This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets.

3) From bagging to random forests

The above procedure describes the original bagging algorithm for trees. Random forests differ in only one way from this general scheme: they use a modified tree learning algorithm that selects, at each candidate split in the learning process, a random subset of the features. This process is sometimes called "feature bagging". The reason for doing this is the correlation of the trees in an ordinary bootstrap sample: if one or a few features are very strong predictors for the response variable (target output), these features will be selected in many of the B trees, causing them to become correlated. An analysis of how bagging and random subspace projection contributes to accuracy gains under different conditions is given by Ho [1].

We finally use the equal number test files to verify the algorithm with the represent of TF and the feature selection of DF. It has higher accuracy rate.

V. CONCLUSION

In this research, we employ a malware detection approach to detect unknown malware. Smali files are analyzed in order to extract operation code sequences and n-gram models are employed to discover essential features from these sequences. The Random Forest is applied to analyze feature vectors obtained and to build a benign software behavior model. This model is then used to detect new malicious applications. Thus, the algorithm proposed allows one to detect malware unseen previously.

REFERENCES

[1] Ho, Tin Kam (1998). "The Random Subspace Method for Constructing Decision Forests" (PDF). IEEE Transactions on Pattern Analysis and Machine Intelligence 20 (8): 832‑844. doi:10.1109/34.709601.

[2] Abou-Assaleh T, Keselj V, Sweidan R: N-gram based detection of new malicious code. Proc of the 28th Annual

[3] Hastie, Trevor; Tibshirani, Robert; Friedman, Jerome (2008). The Elements of Statistical Learning (2nd ed.). Springer. ISBN 0-387-95284-5.

[4] Kleinberg, Eugene (1996). "An Overtraining-Resistant Stochastic Modeling Method for Pattern Recognition" (PDF). Annals of Statistics 24 (6): 2319‑2349. doi:10.1214/aos/1032181157. MR 1425956.

[5] L. Gordon, M. Loeb, W. Lucyshyn, and R. Richardson. Computer Crime and Security Survey. Technical report, Computer Security Institute, 2005.

[6] Rieck K, Holz T, Düssel P, Laskov P: Learning and classification of malware behavior. Conference on Detection of

[7] Moskovitch R, Elovici Y, Rokach L: Detection of unknown computer worms based on behavioral classification of the host. Computational Statistics and Data Analysis 2008, 52(9):4544-4566.

[8] G. Ollmann. The evolution of commercial malware development kits and colour-by-numbers custom malware, Computer Fraud & Security, pp.4-7, 2008.

[9] W. Li, K. Wang, S. Stolfo, B. Herzog. Fileprints: Identifying file types by n-gram analysis. Proc. of the IEEE Workshop on Information Assurance and Security, 2005.

[10] D. Cai, J. Theiler, M. Gokhale. Detecting a malicious executable without prior knowledge of its patterns. Proc. of the Defense and Security Symposium. Information Assurance, and Data Network Security, vol. 5812, pp. 1-12, 2005.

[11] S. Yu, S. Zhou, L. Liu, R. Yang, J. Luo. Malware variants identification based on byte frequency. Proc. of Networks Security Wireless Communications and Trusted Computing (NSWCTC), vol. 2, pp. 32-35, 2010.

[12] D. Bilar, Opcodes as predictor for malware. International Journal of Electronic Security and Digital Forensics, pp. 156-168, 2007.

[13] R. Moskovitch, C. Feher, N. Tzachar, E. Berger, M. Gitelman, S. Dolev and Y. Elovici. Unknown Malcode Detection Using OPCODE Representation. Proc. of the 1-st European Conference on Intelligence and Security Informatics (EuroISI ’ 08), 2008.

[14] I. Santos, F. Brezo, X. Ugarte-Pedrero, P. G. Bringas. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Information Sciences, vol. 231, pp. 64-82, 2013.

[15] C. Y. Suen. n-Gram Statistics for Natural Language Understanding and Text Processing. Pattern Analysis and Machine Intelligence, IEEE Transactions. Vol. PAMI-1, Is. 2. pp. 164-172. 1979.

[16] T. Hirsimaki, J. Pylkkonen, M. Kurimo. Importance of High-Order NGram Models in Morph-Based Speech Recognition. Audio, Speech, and Language Processing, IEEE Tran., Vol. 17, Is. 4. pp. 724-732. 2009.

[17] K. Kira, L. Rendell. The feature selection problem: traditional methods and new algorithm. Proc. of Conference on Artificial Intelligence, 1992.

[18] Freund Y, Schapire RE: A brief introduction to boosting. International Joint Conference on Artificial Intelligence Morgan Kaufmann Publishers Inc; 1999, 1401-1406.

[19] Weiss GM, Provost F: Learning when training data are costly: the effect of class distribution on tree induction. Journal of Artificial Intelligence Research 2003, 19:315-354.

[20] W. Li, K. Wang, S. Stolfo, B. Herzog. Fileprints: Identifying file types by n-gram analysis. Proc. of the IEEE Workshop on Information Assurance and Security, 2005.

[21] I. Santos, F. Brezo, X. Ugarte-Pedrero, P. G. Bringas. Opcode sequences as representation of executables for data-mining-based unknown malware detection. Information Sciences, vol. 231, pp. 64-82, 2013.

[22] Kam HT: Random Decision Forest. Proc of the 3rd International Conference on Document Analysis and Recognition 1995, 278-282.

[23] Linn C, Debray S: Obfuscation of executable code to improve resistance to static disassembly. Proc of the 10th ACM conference on Computer and communications security ACM Press; 2003, 290-299.

[24] Henchiri, O., Japkowicz, N.: A Feature Selection and Evaluation Scheme for Computer Virus Detection. In: Proceedings of ICDM 2006, Hong Kong, pp. 891‑895 (2006)

[25] Chawla, N.V., Japkowicz, N., Kotcz, A.: Editorial: special issue on learning from imbalanced data sets. SIGKDD Explorations Newsletter 6(1), 1‑6 (2004)

[26] D.H. Shih, B. Lin, H.S. Chiang, M.H. Shih, "Security aspects of mobile phone virus: a critical survey," Industrial Management & Data Systems, vol. 108(4), 2008, pp. 478-494.

[27] J. Cheng, S.H. Wong, H. Yang, S. Lu, "SmartSiren: virus detection and alert for smartphones," Proc. International Conference on Mobile Systems, Applications and Services, 2007.

[28] A. Shabtai, Y. Fledel, U. Kanonov, Y. Elovici, S. Dolev, C. Glezer, "Google Android: A Comprehensive Security Assessment," IEEE Security and Privacy, vol. 8(2), March/April 2010, pp. 35-44.

[29] Shabtai A, Moskovitch R, Feher C, et al. Detecting unknown malicious code by applying classification techniques on OpCode patterns[J]. Security Informatics, 2012, 1(1):1-22.

[30] Bin Cao, Dou Shen, Jian-Tao Sun, Qiang Yang, Zheng Chen. Feature selection in a kernel space. Proc. of the 24th international conference on Machine learning (ICML’ 07), pp. 121-128. 2007.