IJCSE International Journal of Computer Sciences and Engineering Open Access

Research Paper Volume-5, Issue-11 E-ISSN: 2347-2693

Android System Call Analysis for Malicious Application Detection

Sapna Malik

Dept. of Computer Science and Engineering, Maharaja Surajmal Institution of Technology, New Delhi, India

[*]Corresponding Author: sapnadhankhar@gmail.com, Tel.: +919711116044

Available online at: www.ijcseonline.org

Received: 13/Oct/2017, Revised: 27/Oct/2017, Accepted: 19/Nov/2017, Published: 30/Nov/2017

***Abstract*—** Nowadays, Android Malware is coded so wisely that it has become very difficult to detect them. The static analysis of malicious code is not enough for detection of malware as this malware hides its method call in encrypted form or it can install the method at runtime. The System Calls tracing is an effective dynamic analysis technique for detecting malware as it can analyze the malware at the run time. Moreover, this technique does not require the application code for malware detection. Thus, this can detect that Android malware also which are difficult to detect with static analysis of code. The paper presented the framework of detecting malicious application from 81 malware families by analysis of dynamic feature System Calls Invoked with machine learning algorithms.

***Keywords*—**System Call,Malicious application detection,malware families

I. Introduction

The Android has modified Linux 2.6 Kernel at the core. The modifications are done for adopting this operating system for the mobile devices. The Android Specific Kernel enhancement includes power management, shared memory drivers, alarm drivers, binders, kernel debugger & logger and low memory killers. The Android application takes the services of the kernel through the System Calls. Whenever a user requests for services like call a phone in user mode through the phone call application, the request is forwarded to the Telephone Manager Service in the application framework. The Dalvik Virtual Machine in Android runtime transforms the user's request passed by the Telephone Manager Service to library calls, which results in multiple System Calls to Android Kernel. While executing the System Call, there is a switch from user mode to kernel mode to perform the sensitive operations. When the execution of operations requested by the System Call is completed, the control is returned to the user mode. The Kernel Invocation calls are sub-grouped into three types of System Calls: 1) System Call- used to invoke native operations of the kernel. 2) Binder Call- for invocation of the binder drivers in the kernel. 3) The Socket Call - allows the read/write/send/receive operations of Linux socket. In this research work, all these subgroups considered as System Calls during behaviour analysis. There are 250 types of System Calls in Android Operating system for performing operations like allocating resources, performing read/write operations, protecting critical data, etc. As discussed, the System Calls are the interface between the user and the kernel. All requests from the applications will pass through the System Call Interface before its execution through the hardware. Thus capturing and analysing the System Calls can give information about the behaviour of the application.

The rest of the article is divided into 5 sections. Section I discussed the importance of system call analysis. Section II contains the related work of system call analysis for android malicious detection. Section III contains the methodology for the framework of malicious application detection by analyzing the system call features with machine learning algorithms. Section IV has discussed the result and Section V briefed the conclusion.

II. Related Work

Many Researchers used system call features analysis for malicious application detection .In their work Schmidt et al.[1] proposed intrusion detection system for an android device which watches the system activities through process list of open files, network traffic, symbol table and system call traces of the complete system to find out any abnormality in system behavior. Kolbitsch et al. [2] performed an analysis of six malware families Allaple, Bagle, Mytob, Agent, and Netsky by finding the correlation between them in terms of the System Call. Wang et al. [3] proposed software theft detection system with System Call based software birthmark system. Authors addressed the problem of software theft where a small part of the software is theft for doing malicious activities. A.lanziet al. [4] proposed the malicious application detection system for personal computers based on the analysis of System Call invoked by the application, and achieved the detection rate of 89 %. Sanz et al. [5] used machine learning algorithm for classification of android applications in several categories like games, tools, entertainment etc. Authors considered various algorithms such as KNN, SVM, J48, Random Forest and Tree Augmented Naïve (TAN), and achieved maximum ROC 0.93 with TAN. Tchakount and Dayang [6] proposed the System Call based methodology for analyzing two malware families, Super History Eraser and Task Killer. Sato et al. [7] proposed the method of calculating the malignancy score of the android application based on the information permission, Intent filter (action), Intent filter (category), and Process for classification of android applications and have the accuracy of 91.4 %. Huang et al. [8] also used machine learning technique for classification of Android Application and have a maximum accuracy of 81 % with J48. Canfora et al. [9] proposed malware detection approach based on the analysis of System Call and permission feature, and classified the malicious application based on the three metrics. Liu & Liu [10] proposed Two-Layered Permission-Based Android Malware Detection Scheme using machine learning techniques for classification of benign and malicious applications. Jeong et al. [11] proposed a malware detection technique based on system call and binder analysis. B.R. Patel [16] has used classification algorithms for improving student performance.

From literature survey it is observed that the maximum achieved accuracy with anomaly intrusion detection is 89 % by analyzing the malicious application with system call feature by considered only few malware families or no malware families while malicious application detection. The no. of malware families considered is important factor in android malicious application detection and need to consider more malware families in android malicious application detection. The objective of this work is to detect malicious application with analysis of system call with machine learning algorithms. The dataset of malicious application is considered from 81 malware families.

III. Methodology

Data Collection

The .apk files of Android Application Samples are taken from resources Drebin Research Work [12] & Androtracker Research Work [13]. The 1060 android applications- 533 benign applications and 527 malicious applications samples from 81 malware families are taken for creating the Dataset. The Benign Android Application repository has been requested from the research work AndroTracker[13] , the Hacking and Countermeasure Research Lab established in the Graduate School of Information Security of the Korea University, Seoul, Korea.The research work AndroTracker has collected 51,179 Benign applications which have been collected during the period of January 2013 to August 2013 by downloading them from the Android market & Google Play. The Malicious Application Data have been bagged from The Drebin Dataset [12] that have 5,560 apk files of malicious application collected during the period from August 2010 to October 2012.

Feature Extraction and Data Creation

The next phase is dynamic feature “System Calls Invoked” extraction of Android Applications for the analysis. This feature is extracted with proposed automatic tool AndroData [14]. The AndroData tool installs each application automatically on the Emulator and extracts the Android Permission Requested and System Call Invoked features by simulating the android application. The System Calls traces and Android permissions Requested are recorded in the text files on the emulator.

After the uninstallation of an android application, the text files having *Android System Calls Invoked & their frequency* features of the application is pulled back from the emulator and stored on the storage device for creating feature vectors of all android applications. For making the feature vector, the author have developed the java program which read the text files of each application and copied its System Call Invoked attributes in the excel files. While reading the System call Invoked feature from the text file, it places the System Call Frequency value in the column having the same System Call heading and 0 in the mismatched column heading.

After the successful execution of java program, the authors have the Dataset in the excel format.

Refining of System Call Invoked Dataset based on behavior Analysis

The System Calls Invoked dataset has 82 types of *System Calls Invoked* and their frequency of invocation by the Android applications. The rows have the application’s package name and columns have different types of System Calls invoked and their frequency of invocation by applications. Some insignificant System Calls are removed from the dataset and the dataset is left with 75 different types of System Calls Invoked. To make the dataset more valuable, some more attributes are added to the dataset based on the behaviour analysis of System Calls invoked [15] by applications like *no. of System Calls Invoked, no. of System*

Calls Invoked with frequency more than 100, no. of System Calls with frequency more than 1000 which would help the machine learning algorithms to identify the behaviour patterns of benign and malicious applications more accurately during experiments.

Performance Metrics

The performance of the machine learning classifier is measured in terms of True Positive (TP), True Negative, False Positive and False Negative, represented by confusion matrix . Table 1 shows the Confusion Matrix used in the experiments. The True Positive is no. of Android Applications correctly classified as Benign Applications. True Negatives (TN) is the no. of Android Applications correctly classified as Malicious Applications. False Positives is the no. of Android Applications mistakenly classified as Benign and False Negatives (FN) is the no. of Android Applications mistakenly classified as Malicious Applications.

Table 1. Confusion Matrix

Confusion Matrix		Predicted Instances	
		Benign	Malicious
Actual Instances	Benign	TP	FN
	Malicious	FP	TN

True Positive Rate (TPR) means the proportion of correctly identified applications as Benign Applications. False Positive Rate (FPR) represents the proportion of Malicious Applications incorrectly identified as Benign. Accuracy is the proportion of correctly identified applications both Benign and Malicious from the total no. of applications. F1–score is also called F–measure having the range from 0 to 1 where the best is 1 and the worst is 0. A Receiver Operating Characteristic (ROC) curve is a graphical plot in which X axis is True Positive Rate and Y axis is False Positive Rate which is used for illustrating the performance of a binary classifier. ROC curves allow us to evaluate and compare the classification algorithms and help us to select the optimal classifier. The area under the ROC curve known as AUC, judges the performance of a binary classifier. The AUC=1 is considered perfect prediction and below 0.6 is considered poor prediction. The next section evaluates the different classifiers on the basis of above parameters.

IV. RESULTS AND DISCUSSION

In this research work, the experiment with *System Calls Invoked* features is performed with two datasets, Dataset 1 with 1060 Android Applications and Dataset 2 with 260 Android Applications. The Dataset1 has 527 Benign Applications and 533 Malicious Applications. Dataset 2 has 82 Benign Applications and 178 Malicious Applications. Both Datasets have 75 different *System Calls Invoked* by the Android applications. These datasets are used for training and testing classifiers such as RBF classifier, Kernel Logistic Regression, SMO, KStar, Naïve Bayes, Simple Logistic, Random Forest, and SVM. The experiment is executed with 10 fold cross validation techniques.

Table 2 shows the experiment results with malicious applications detection with Dataset1 in which KStar and Random Forest are classifiers with good performance as compared to others. The Random Forest is the best performing classifier with a maximum accuracy of 66% and ROC of 0.7. The Random Forest also has minimum False Positive Rate of 0.33. Although with the Dataset 1 the maximum achieved accuracy is comparatively less as the Dataset 1 needs to be refined further.

Table 2. Experiment Result of Different Classifiers Performance with Dataset 1

Classifier	ACCURACY	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
RBF Classifier	62.67	0.627	0.369	0.718	0.627	0.585	0.651
Kernel Logistic Regression L-0.01	61.9	0.619	0.376	0.716	0.619	0.573	0.676
SMO	61.6	0.616	0.379	0.756	0.616	0.558	0.619
KStar	64.5	0.645	0.353	0.675	0.645	0.63	0.662
Naïve Bayes	60.79	0.608	0.387	0.723	0.608	0.552	0.604
Simple Logistic	61.3	0.613	0.383	0.737	0.613	0.556	0.613
Random Forest	66	0.667	0.33	0.696	0.667	0.655	0.7
SVM	63.9	0.639	0.358	0.673	0.639	0.621	0.64

The Dataset 1 is further refined to form the Dataset 2. Table 3 represents the different classifiers performance after training and testing them with Dataset 2. As it is clearly visible from the experimental results the performance of each classifier is improved significantly with Dataset 2. The performance of Kernel Logistic regression, Simple Logistic and Random Forest is better as compared to other classifiers. The Random Forest is the best performing Classifier with Accuracy of 85% and ROC of 0.922.

Table 3 Experiment Results of Different Classifiers performance with Dataset 2

Classifier	Accuracy	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
RBF Classifier	75.8	0.758	0.256	0.777	0.758	0.763	0.858
Kernel Logistic Regression L-0.01	81.92	0.819	0.182	0.834	0.819	0.823	0.884
SMO	78.84	0.788	0.143	0.842	0.788	0.796	0.822
KStar	80.38	0.804	0.209	0.817	0.804	0.808	0.873
Naïve Bayes	71.92	0.719	0.162	0.822	0.719	0.728	0.833
Simple Logistic	83.84	0.858	0.118	0.876	0.858	0.861	0.926
Random Forest	85	0.85	0.187	0.852	0.85	0.851	0.922
SVM	76.2	0.762	0.281	0.771	0.762	0.765	0.74

V. CONCLUSION

In Anomaly based intrusion detection system, the behaviour of the malicious application is analyzed with machine learning algorithms. System call is important dynamic features for malicious application detection for anomaly based intrusion detection system. The framework presented is used System Call Invoked feature for malicious applications detection and successful in achieving accuracy of 85% and ROC of 0.922 for detection of malicious application from 81 malware families with machine learning algorithms. Random forest Machine Learning algorithms found the promising algorithm for malicious application detection with system call invoked feature. Although the simple logistic algorithms is also found good algorithm for malicious application detection. The accuracy can be further improved by the combining the other features like permission used, API call etc. with the analysis of system call.

REFERENCES

[1] Schmidt, Aubrey-Derrick, Hans-Gunther Schmidt, Jan Clausen, Kamer A. Yuksel, Osman Kiraz, Ahmet Camtepe, and Sahin Albayrak. "*Enhancing security of linux-based android devices.*" In Proceedings of 15th International Linux Kongress, pp. 1-16. 2008.

[2] Kolbitsch, Clemens, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiao-yong Zhou, and XiaoFeng Wang. "*Effective and Efficient Malware Detection at the End Host*." In USENIX security symposium, pp. 351-366. 2009.

[3] Wang, Xinran, Yoon-Chan Jhi, Sencun Zhu, and Peng Liu. "*Detecting software theft via system call based birthmarks*." In Computer Security Applications Conference, 2009. ACSAC'09. Annual, pp. 149-158. IEEE, 2009.

[4] Lanzi, Andrea, Davide Balzarotti, Christopher Kruegel, Mihai Christodorescu, and Engin Kirda. "*Accessminer: using system-centric models for malware protection.*" In Proceedings of the 17th ACM conference on Computer and communications security, pp. 399-412. ACM, 2010.

[5] Sanz, Borja, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, and Pablo Garcia Bringas. "*On the automatic categorisation of android applications.*" In Consumer Communications and Networking Conference (CCNC), 2012 IEEE, pp. 149-153. IEEE, 2012.

[6] E.Tchakount, P.Dayang ."*System calls analysis of malware on android*". International Journal of Science and Technology. Vol. 2 issue 9,2013

[7] Sato, Ryo, Daiki Chiba, and Shigeki Goto. "*Detecting Android malware by analyzing manifest files*." Proceedings of the Asia-Pacific Advanced Network 36 (2013): 23-31.

[8] Huang, Chun-Ying, Yi-Ting Tsai, and Chung-Han Hsu. "*Performance evaluation on permission-based detection for android malware*." In Advances in Intelligent Systems and Applications-Volume 2, pp. 111-120. Springer, Berlin, Heidelberg, 2013.

[9] Canfora, Gerardo, Francesco Mercaldo, and Corrado Aaron Visaggio. "*A classifier of malicious android applications.*" In Availability, Reliability and Security (ARES), 2013 Eighth International Conference on, pp. 607-614. IEEE, 2013.

[10] Liu, Xing, and Jiqiang Liu. "*A two-layered permission-based Android malware detection scheme.*" In Mobile cloud computing, services, and engineering (mobilecloud), 2014 2nd ieee international conference on, pp. 142-148. IEEE, 2014.

[11] Jeong, Youn-sik, Hwan-taek Lee, Seong-je Cho, Sangchul Han, and Minkyu Park. "*A kernel-based monitoring approach for analyzing malicious behavior on android*." In Proceedings of the 29th Annual ACM Symposium on Applied Computing, pp. 1737-1738. ACM, 2014.

[12] Arp, Daniel, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, Konrad Rieck, and C. E. R. T. Siemens. "*DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket.*" In NDSS. 2014.

[13] Kang, Hyunjae, Jae-wook Jang, Aziz Mohaisen, and Huy Kang Kim. " *Comparative analysis of classification algorithm in EDM for improving student performance*." International Journal of Distributed Sensor Networks (2015).

[14] S.Malik and K. Khatter. "*AndroData: A Tool for Static & Dynamic Feature Extraction of Android Apps.*" International Journal of Applied Engineering Research,Vol. 10, issue 94, 2015.

[15] S.Malik and K. Khatter. "*System Call Analysis of Android Malware Families.*" Indian Journal of Science and Technology,Vol. 9, issue 21 ,2016.

[16] B.R. Patel, "*Comparative analysis of classification algorithm in EDM for improving student performance*", International Journal of Computer Sciences and Engineering, Vol.5, Issue.10, pp.171-175, 2017.

Authors Profile

Ms. Sapna Malik, pursed Bachelor of Engineering from M.D.University,Rohtak in 2004, Master of Technology from GGSIPU University,Delhi in 2009 and Ph.D. from Ansal University,Gurgaon. She is currently working as Assistant Professor in Department of Computer Science & Engineering ,MSIT,Delhi,India since 2006. She is life time member of ISTE. She has published more than 20 research papers in reputed international journals including Thomson Reuters (ESCI & Scopus) and conferences including IEEE and it's also available online. Her main research work focuses on Mobile Security, Cryptography Algorithms, Network Security, Cloud Security and Privacy, Big Data Analytics, Data Mining, and Computational Intelligence based education. She has 11 years of teaching experience and 4 years of Research Experience.