International Journal of Trend in Scientific Research and Development (IJTSRD)
Volume 3 Issue 5, August 2019 Available Online: www.ijtsrd.com e-ISSN: 2456 – 6470

Malware Detection in Android Applications

Mr. Tushar Patil, Prof. Bharti Dhote

Department of Computer Engineering, SIT Lonawala, SPPU, Pune, India

How to cite this paper: Mr. Tushar Patil | Prof. Bharti Dhote "Malware Detection in Android Applications" Published in International Journal of Trend in Scientific Research and Development (ijtsrd), ISSN: 2456-6470, Volume-3 | Issue-5, August 2019, pp.2401-2403, **https://doi.org/10.31142/ijtsrd26449**

IJTSRD26449

ABSTRACT

Android is a Linux-based operating system used for smart-phone devices. Since 2008, Android devices gained huge market share due to its open architecture and popularity. Increased popularity of the Android devices and associated primary benefits attracted the malware developers. Rate of Android malware applications increased between 2008 and 2016. In this paper, we proposed dynamic malware detection approach for Android applications. In dynamic analysis, system calls are recorded to calculate the density of the system calls. For density calculation, we used two different lengths of system calls that are 3-gram and 5-gram. Furthermore, Naive Bayes algorithm is applied to classify applications as benign or malicious. The proposed algorithm detects malware using 100 real-world samples of benign and malware applications. We observe that proposed method gives effective and accurate results. The 3-gram Naive Bayes algorithm detects 84% malware application correctly and 14% benign application incorrectly. The 5-gram Naive Bayes algorithm detects 88% malware application correctly and 10% benign application incorrectly.

KEYWORDS: *Malware Detection • Naive Bayes Classifier • System Calls • Frequency • Density*

INTRODUCTION

Android is a most popular and fastest growing mobile application development framework. Since 2008, the adoption rate of Android has increased quickly. There are approximately 1.5 million Android devices being activated every day[18]. In the first quarter of 2017, Android occupy approximately 86.1% market share[17]. It is an open-source platform based on Linux kernel. Android OS is developed and maintained by Google and promoted by Open Handset Alliance. Android applications are developed in Java, Python. Android provide very user-friendly functionalities at truly low cost. Android users use Android phones for storage, communication, the Internet surfing, etc. To analyze malware static, dynamic and hybrid analysis methods are used[1]. Static analysis method identifies malware by unpacking and decompiling application. Mostly, Commercial anti-virus uses signature-based malware detection technique. The dynamic analysis identifies malware behavior after deploying and executing the application. The hybrid analysis is the combination of static and dynamic methods. There are two main steps to overcome malware named as identification of malware and detection of malware. Application signature, permissions, and Dalvik bytecode are the parameters used for static analysis of malware[3]. System calls, network traffic, user interactions are the parameters used by dynamic analysis[3]. Hybrid analysis technique uses a combined feature of static and dynamic approach.

In this work, we describe dynamic malware detection techniques. For dynamic analysis first, we install all samples on Emulator. Then, we run all applications for a 2-3 minute and record system calls. After that, we calculate the frequency of the system call. Next, we apply the filter on system calls. Filtered system calls are used for calculating density. Furthermore, system calls are parsed and mapped into the machine learning algorithm. We use Naive Bayes classifier for classification of the application as benign or malicious. Using mapped system calls as input, we train the classifier. After that, we apply classifier and classify the application as benign or malicious. The whole system applied to real world benign and malware application samples.

Our main contribution in this work is: 1. we performed system call based dynamic malware analysis techniques. We used Naive Bayes classification algorithm for detection. 2. We used 3-gram and 5-gram length of system calls which reduces time complexity of system. While, we filtered system calls on basis of their frequency. It reduces overhead without losing accuracy. 3. Performance of the overall dynamic malware detection system is better and gives more accurate results. Proposed system gives 85% and 89% accuracy in results for 3-gram and 5-gram Naive Bayes algorithm.

LITERATURE SURVEY

Faruki Parvez et *al.*[1] and Arshad Saba et *al.*[2], gives a detailed survey of Android architecture and malware. Parvez Faruki et *al.*[1], gives Android security architecture and its issues, malware types, and its penetration techniques. They discussed malware detection methods that are static malware detection and dynamic malware detection. Also, they covered malware analysis and detection approaches according to their goal, methodology, and deployment. Finally, they proposed a hybrid approach to analyze and

detect Android Malware. Arshad Saba et *al*.[2] gives details of different Android malware types and its penetration techniques. They categorized different antimalware techniques like static and dynamic malware detection. At the end, they proposed the hybrid antimalware concept to overcome limitations of the static and dynamic approach.

Feizollah et *al*.[3] provide details about feature selection from Android applications for malware detection. Based on deep research, they categorized four different feature selection group like application meta-data, hybrid, dynamic and static features. It gives a novel introduction about Android malware detection types and related features. They proposed permission, signature, Java's code, etc. features used for static malware detection. While the system calls, network traffic, user interactions are the feature set for dynamic malware detection.

In paper [4], [5], [6], [7] authors suggested static malware analysis techniques with a different approach. Geoffroy Gueguen et al.[4] propose static malware analysis tool named as Androguard. Androguard is the Python-based static malware analysis tool used to disassemble and decompile Android apps by using reverse engineering. Androguard calculates application similarities and differences by using NCD(Normalized Compression Distance), fuzzy risk score and signatures of the malicious application. Faruki Parvez *et al.[*5] describe the tool Androsimilar. Androsimilar is a signature based static malware analysis tool. Androsimilar automatically generates the signature of the test application. Generated signature is compared against malware signature database. Then identify it as the normal or malicious application. Daniel Arp *et al*.[6] propose the static malware analysis tool called as Drebin. Drebin is a static malware analysis tool which detects malicious application directly on Android phone. Drebin collects various features from application code and manifest file. Then machine learning approach is used to distinguish normal and malicious application. Sanz Borja *et al*.[7] propose permission based static malware detection tool called PUMA. PUMAs extract application permission from the manifest file. Then use the machine learning algorithm to identify normal and malicious permissions

In paper [8],[9],[10],[11] authors suggest dynamic malware analysis techniques with the different approach. Suarez Tangil *et al*.[8] proposes the dynamic analysis tool named as AlterDroid. AlterDroid a tool for dynamic analysis of hidden malware distributed over application components. Alterdroid analyses the behavioral difference between original application and fault injected application. It creates behavior signatures for both applications. It then analyze differential signature with the help of pattern matching. Tam Kimberly *et al*.[9], describe tool CopperDroid. CopperDroid is virtual machine based automatic dynamic analysis system. It reconstructs the behaviour of Android malware by monitoring system calls. Shabtai Asaf *et al*.[10] suggest tool Andromaly. Andromaly is the host-based malware detection tool. Andromaly continuously monitors various metrics of the device like battery usage, CPU usage, the number of active processes and amount of data transferred through a network. Then it applies the machine learning algorithm for classifying data as normal and malicious. Lok Kwong Yan *et al*.[15] proposes the dynamic malware tool called as Droidscope. Droidscope is a dynamic malware analysis platform which is based on virtual machine introspection. Droidscope is built upon QEMU emulator. It is monitoring whole operating system to get more information regarding malware and also detect kernel level attack.

PROPOSED METHODOLOGY

Preprocessing: The first step of proposed system is to collect real-world samples of benign and malware applications. After collection of application sample, system next go to the second step of recording system calls. Figure 1 shows the flow of system call recording. Initially, we installed every application on Android emulator and run for a 2-3 minute. After that, we recorded system calls of each application and copied into an external file(.csv). To trace system calls we used.We know that each line in training set represents single application features with multiple feature integer and feature values. Now we labeled each line that means each application with 1 or 0. Where 1 means benign application, and 0 means a malicious application in training set. We have used 70% of application from system data set for training data and remaining for testing. After all this data preprocessing, we applied Naive Bayes classifier in next step

Algorithm 1: Naive Bayes Algorithm for Malware Detection

- The duplicated files are mapped with a single copy of the file data by mapping with the existing file data in the cloud
- The comprehensive requirements in multi-user cloud storage systems and introduced the model of deduplicatable dynamic PoS.

Input: Android Application System calls stored in .csv file
Output: Class from which given system calls belong.

1. Foreach line in file .csv do
2. Remove all parameters except system call name;
3. Store all system call names in another file called system call name;
4. End
5. Foreach system call name in file system call name do
6. Assign unique integer number;
7. Store all integers in file integer system call file;
8. End
9. Foreach integer system call do
10. Calculate 3-gram and 5-gram length;
11. End
12. Foreach length of system call
13. Compute frequency of each integer then;
14. Foreach system call if frequency is less than 100;
15. Remove from file ;
16. Compute density of each integer then;
17. Store data into value pair format in data file;
18. End
19. After all this data processing apply Naive Bayes classifier.
20. Foreach class instance
21. Calculate prior probability;
22. P(C) = Nc N
23. End
24. Foreach known value pair
25. Calculate conditional probability;
26. P(w|c) = countw, c() + 1/count(c) + |V
27. | End
28. Foreach unknown value pair
29. Calculate posterior probability;

30. Cmap = argmaxP(x1, x2, x3, , xn)P P(C) End
31. Compare posterior probability for each class then return class with highest probability as result.

Algorithms

Let D be the Whole system which consists,
D= {I, P, O}

Where,
Q- Users Query {q1, q2…, qN}
P- Procedure,
F-Files set of {f1, f2,…,fn}
I-Input,
I-{F, Q},
O- Output.

Where**:**
F = Represents the file,
m1, m2, m3, m4= representing the i^{th} block of the file,
e = encryption key
Phase 1: Pre-process Phase
In the pre-processing phase,
e← H(F), id ← H(e).

Then, the user announces that it has a certain file via id. If the file does not exist, the user goes into the upload phase. Otherwise, the user goes into the De-Duplication phase.

Phase 2 The Upload File
(C, T)← Encoding(e, F)
Let the file F = (m1, . . . ,mn).
The user first invokes the encoding according

Phase 3. The De-Duplication Data(file)
res∈ {0, 1} ← De-Duplication {U(e, F), S(T)}
If a file announced by a user in the pre-process phase exists in the cloud server, the user goes into the De-Duplication phase and runs the De-Duplication protocol

Phase 4: The Update File
res∈ {he∗, (C∗, T ∗)i,⊥} ← Updating{U(e, i, m, OP), S(C, T)}
In this phase, a user can arbitrarily update the file by invoking the update protocol

Phase 5: The Proof of Storage to Owner
res∈ {0, 1} ←Checking{S(C, T), U(e)}
At any time, users can go into the proof of storage phase if they have the ownerships of the files. The users and the cloud server run the checking protocol.

RESULT AND DISCUSSIONS

User can upload, download update on cloud server and provide data De-Duplication.

CONCLUSIONS

In this work, we developed dynamic malware detection system to detect malware in Android applications. For dynamic detection, we used system calls invoked by the application during execution. After that, Naive Bayes classifier is used to classify runtime behavior of applications. In addition, we used 3-gram and 5-gram length of system calls. Instead of using every system calls; we filter system calls based on frequency. Filtered system calls are used to calculate density. Then, by applying Naive Bayes classifier, we classified application in two different classes that are benign and malware. For all system implementations, we used real-world malware and benign application samples. Proposed method gives more accurate results and performs better than previous work. For 3-gram Naive Bayes classifier, the system gives 85% accuracy while in 5-gram Naive Bayes classifier; the system gives 89% accuracy. This indicates the performance of the system is proportional to the length of system calls.

REFERENCES

[1] Faruki Parvez, Ammar Bharmal, Vijay Laxmi, Vijay Ganmoor, Manoj Singh Gaur, Mauro Conti, and Muttukrishnan Rajarajan. "Android security: a survey of issues, malware penetration, and defenses." IEEE communications surveys & tutorials 17, no. 2(2015): 998-1022.

[2] Arshad Saba, Munam Ali Shah, Abid Khan, and Mansoor Ahmed. "Android malware detection & protection: a survey." Int. J. Adv. Comput. Sci. Appl 7, no. 2 (2016): 463-475.

[3] Feizollah Ali, Nor Badrul Anuar, Rosli Salleh, and Ainuddin Wahid Abdul Wahab. "A review on feature selection in mobile malware detection." Digital Investigation 13 (2015): 22-37.

[4] Desnos Anthony. "Androguard: Reverse engineering, malware and goodware analysis of android applications." URL code. google. com/p/androguard (2013).

[5] Faruki Parvez, Vijay Ganmoor, Vijay Laxmi, Manoj Singh Gaur, and Ammar Bharmal. "AndroSimilar: robust statistical feature signature for Android malware detection." In Proceedings of the 6th International Conference on Security of Information and Networks, pp. 152-159. ACM, 2013.

[6] 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.

[7] Sanz Borja, Igor Santos, Carlos Laorden, Xabier Ugarte-Pedrero, Pablo Garcia Bringas, and Gonzalo lvarez. "Puma: Permission usage to detect malware in android." In International Joint Conference CISIS12-ICEUTE 12-SOCO 12 Special Sessions, pp. 289-298. Springer Berlin Heidelberg, 2013.

[8] Suarez-Tangil, Guillermo, Juan E. Tapiador, Flavio Lombardi, and Roberto Di Pietro. "ALTERDROID: differential fault analysis of obfuscated smartphone malware." IEEE Transactions on Mobile Computing 15, no. 4 (2016): 789-802.

[9] Tam Kimberly, Salahuddin J. Khan, Aristide Fattori, and Lorenzo Cavallaro. "CopperDroid: Automatic Reconstruction of Android Malware Behaviors." In NDSS. 2015.

[10] Shabtai Asaf, Uri Kanonov, Yuval Elovici, Chanan Glezer, and Yael Weiss. "Andromaly: a behavioral malware detection framework for android devices." Journal of Intelligent Information Systems 38, no. 1 (2012): 161-190.

[11] Yan, Lok-Kwong, and Heng Yin. "DroidScope: Seamlessly Reconstructing the OS and Dalvik Semantic Views for Dynamic Android Malware Analysis." In USENIX security symposium, pp. 569-584. 2012.