Evaluating Efficiency of Android Anti-Malware System

Ms S. M. Shelake[1] Prof. P. B. Dhainje[2]
[1]ME Student
[1,2]Department of Computer Science & Engineering
[1,2]SIETC, Paniv Solapur University, Solapur India

Abstract— The admiration of Android OS for mobile is appealing the threats such as malwares. The term 'malware' is defined as mixture of form of invasive software. Malware is any program or data which disturbs the working of a device. Thus malware detection is the issue in the computer security. To evade the malware attacks different anti-malwares are also have been established. But there is a need to assess these anti-malwares which can be done by using Droid Chameleon. Droid Chameleon does the transformation of malwares automatically and helps to check the efficiency of anti-malware. Here we propose a system that classifies the malicious apps affected due to malwares. The authorizations given by android apps are used as the dataset. The ID3 algorithm is used to apply mining on these datasets i.e. training is provided to produce the trained dataset. The Admin will take care of new entries of malwares as well as apps in the database. The results are presented as whether the given app can notice all malware or not.

Key words: Malware, Anti-Malware, Obfuscation Techniques, Transformation

I. INTRODUCTION

Software's such as anti-virus, anti-malware, and firewalls are used by home users and organizations about the world to effort to protection by malware attacks. Malware is code that has malicious intent. Examples of this kind of code include computer viruses, worms, Trojan horses, and backdoors. It can be discovered that there are many malware threats trying to break the security chains of android even though there is a huge security provided by different anti-malware software. The threats are guiding mobile devices - particularly the Android platform. However, it's probably not precise to say the predictable explosion has in fact occurred. The truth is that cyber criminals are quiet very much in the investigative phase of figuring out how to monetize the exploitation of mobile devices. So here, aim is to evaluate the efficiency of anti-malware software on Android in the face of numerous techniques. To do so the term 'transformation' which refers to polymorphic variations is done on malware samples. It does not involve code-level changes. These transformed malwares are useful to the anti-malwares to check their efficiency. The technique called Droid Chameleon is used to conduct these common transformations. This technique changes android applications repeatedly. That is the purpose why we are trying to study the anti-malwares and their proficiency to defeat the attacks.

We proposed the algorithm that should be able to recover the usage of android apps. The main objective is to discover possible ways to develop current Anti-malware solutions. The system should point out advanced detection techniques because of high-level bytecodes of android. The system should be capable to check whether the given android app is malicious or not. For that we are using ID3 algorithm on various transformations.

II. RELATED WORK

Studies have been added to decline the malware attacks and to increase the performance of the mobile devices.

A. Semantics-Aware Malware Detection

Mihai Christodorescu et.al. Proposed [3] a malware detector system that checks whether a program has malicious behavior. In order to avoid detection, malware writers (hackers) often use obfuscation to transform malware. Anti-malwares uses a pattern-matching approach (such as commercial virus scanners) are vulnerable to complications used by hackers. The essential deficit in the pattern-matching method to malware detection is that it is purely syntactic and ignores the semantics of instructions. In this presented algorithm which reports this deficit by including instruction semantics to detect malicious program qualities. Experimental outcomes are revealed that this malware-detection algorithm can notice variations of malware with a moderately low run-time overhead. Furthermore, our semantics-aware malware detection algorithm is tough to public problems used by hackers.

B. ADAM

ADAM [1], a mechanical and extensible system that can evaluate, the effectiveness of anti-virus with a numerous malware samples for the Android platform. It inevitably converts an Android malware sample into different variations through various repackaging and obfuscation techniques, while conserving the original malicious behavior. Fig.1 shows the design of ADAM.

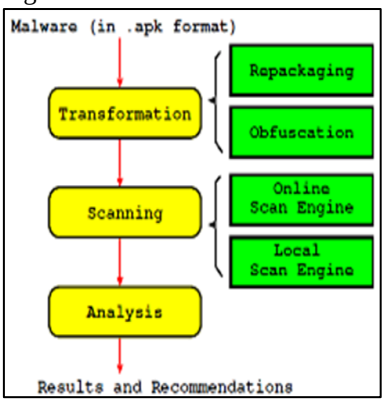

Fig. 1: Design of ADAM

ADAM is calculated by joining different building blocks. These blocks are used to test different anti-viruses against malware samples. It can be used for studies of very large-scale malware samples. As transformation is done manually there is no need to apply manual alteration of malwares. This results less overhead on codes. ADAM is not always capable to evade an anti-malware tool. It implements only some of transformations, such as renaming methods, introducing junk methods. It cannot be said that ADAM will always offer the better finding mechanisms.

C. Effective and Efficient Malware Detection at the End Host

Clemens Kolbitsch et al. [4] proposed a novel malware detection method that is both *effective* and *efficient*, and thus, can be used to substitute old anti-virus tool at the end host. This method examines a malware to figure a model that describes its behavior. Such models describe the information flows between the system calls essential to the malware's mission, and therefore, cannot be easily evaded by simple obfuscation or polymorphic techniques. Then, extract the program slices responsible for such information flows. For detection, execute these slices to match with these models against the runtime behavior of an unknown program. It can efficiently detect consecutively malicious code on an end user's host with a small overhead. It produce effective tool that capture detailed information about the performance of a malwares variation. Scanner that can proficiently tie the activity of an unknown program against this system. It cannot produce system call signatures or find a starting point for the slicing process.

D. Automated Security Analysis of Smartphone Applications

To do the automation of security investigation the tool Apps Playground [5] is used. It integrates multiple components containing different recognition and automatic examination techniques for this purpose .The system can be assessed using numerous large and small scale experiments relating real benign and malicious application. It gives real analysis even with large number of applications. It is less effective at inevitably detecting privacy leakages and malicious functionality in application.

III. System Implementation

A. System Architecture

The Fig.2 shows the overall system architecture. As per shown in figure the process flow goes according to the system architecture. The system architecture includes following components:

1) Application Permissions and Data Import

The various android application authorizations are fetched from android applications. These authorizations are used as dataset for process.

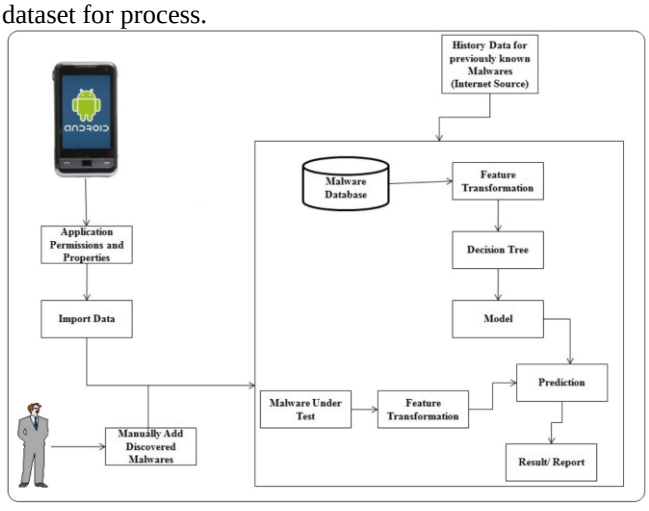

Fig. 2: System Architecture.

2) Malware Database

The malware samples will be deposited at database.

3) Feature Transformation and Decision Tree

The Decision tree is produced using ID3 algorithm. The permissions of application are given as an input for feature transformation. The permissions are training datasets for the algorithm. These permissions are the initial inputs for feature transformation.

4) Prediction

The datasets of application will be examined here and the related evaluation of malware samples is done with the help of Decision Tree.

5) Report

The result of whether the given Android App is malicious or not will be shown.

B. Algorithm

The classifier algorithm ID3 works as follows:

1) Assumptions

- I: is the set of input attributes i.e. permissions (Here we will consider the permissions given by android to app to find out classification attribute.)
- O: is the output attribute
- T: is a set of training data

FunctionID3: returns a decision tree
FunctionID3(I, 0, T) {
if (T is empty) {
return a single node with the value "Failure";
}
if (all records in T have the same value for O) {
return a single node with that value;
}
if (I is empty) {
return a single node with the value of the most frequent value of
O inT;
}
Compute the information gain for each attribute in I relative to T using permissions;
let X be the attribute with largest Gain(X, T) of the attributes in I;
let {xj|j=1,2,….,m } be the values of X;
let {Tj|j=1,2,….,m} be the subsets of T when T is partitioned according the value of X;
return a tree with the root node labeled X and
arcs labeled x1,x2,…., xm; where the arcs go to the
trees ID3(I-{X},O, T1), ID3(I- {X},O, T2),…, ID3(I- {X},O, Tm);

The Entropy and Information gain are calculated as:

- Entropy: Entropy provides the measure in information theory, which describes the unwanted attributes of an arbitrary collection. If target attribute takes on r different permissions (values), then entropy X relative to this r-wise classification is defined as:

$$Entropy(X) = \sum_{i=1}^{r} -p_i \ log_2 \ p_i$$

- Information Gain: The information gain, Gain(X; T) of an attribute T (here the permission), relative to the collection of examples X, is defined as:

$$Gain(X,T) = Entropy(X) - \sum_{v \in Values(X)} \frac{|X_v|}{|X|} Entropy(X_v)$$

Where, Values (T) is the set of all possible values for attribute T and Xv is the subset of X for which the attribute T has value v.

IV. RESULTS

The system is developed by using JAVA (Version JDK). The progress tool used is NetBeans for desktop application. Eclipse is used for development of Android Application for smartphone. The database used for storing the malwares is Apache Tomcat. The experiments are performed on Core2Duo Intel processor 2 GB RAM. The results are shown as the report whether the app is malicious or not.

A. Dataset of Android permissions for Training

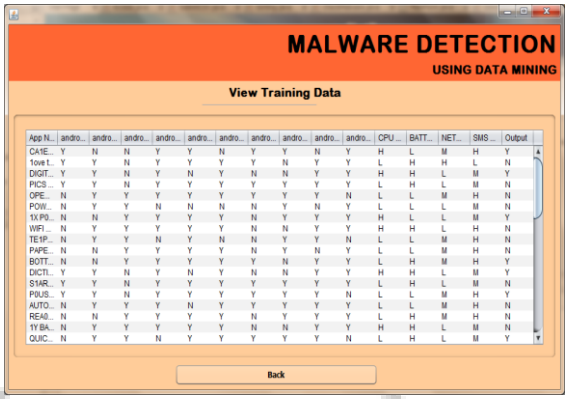

Fig. 3: Dataset of Android permissions for Training

B. Single Entry of App for Testing the single App

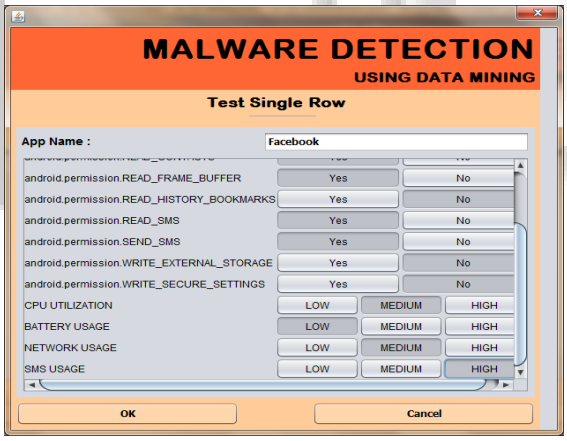

Fig. 4: Single Entry of App for Testing the single App

C. The result of malware on Android App

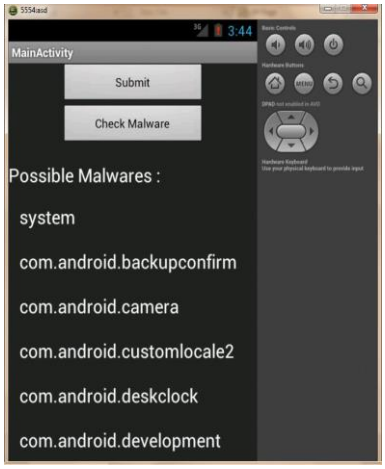

Fig. 5: The result of malware on Android App

D. Graph of Retrieved Objects

Dataset Name	Actual Objects	Retrieved Objects	Correct Retrieved Objects
Malware Apps	20	18	17
Nonmalware-Apps	25	24	23

Table 1: Retrieved Objects

Dataset Name	Precision	Recall
Malware Apps	0.94444444	0.85
Nonmalware-Apps	0.95833333	0.92
Total	0.951388889	0.885
Accuracy ercentage	0.885	-

Table 2: Total Accuracy

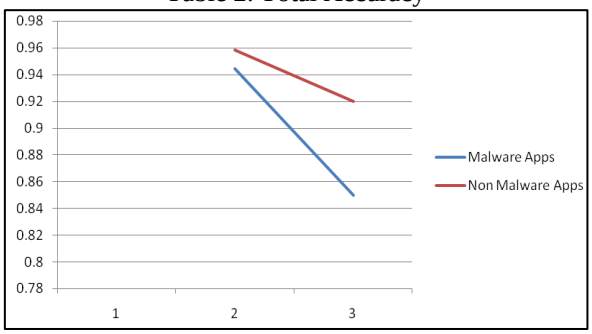

Fig. 6: Accuracy Plotted

Table 1 shows the correct retrieved objects i.e. which apps are found to be malicious and which are not correctly. The actual retrieved objects are showing how much malicious apps are found correctly and how much malicious apps are not found correctly.

Table 2 shows the accuracy plotted in terms of precision and recall.

V. CONCLUSION

Mobile malwares are violent the android systems which cause the susceptibility to the whole application. To evade this we have proposed classifier based anti-malware which will identify the malwares with different functionalities. The permissions of apps with malicious characteristics are used for finding the malicious behaviour. Using the ID3 algorithm the most permission requests used by an app help to classify the smartphone application into malicious and non-malicious application. As a future work a more comprehensive anti-malware tool is possible to implement using artificial-intelligence. There is a scope to detect large number of malwares.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to Prof.P.B.Dahinje for his continuous support, patience, motivation, enthusiasm, and immense knowledge. His guidance helped me in all the time of research and writing of this paper

REFERENCES

[1] M. Zheng, P. Lee, and J. Lui, "ADAM: An automatic and extensible platform to stress test Android anti-virus systems," in Proc. DIMVA, Jul. 2012, pp. 1–20.
[2] C. Collberg, C. Thomborson, and D. Low, "A taxonomy of obfuscating transformations," Dept. Comput. Sci.,

Univ. Auckland, Auckland, New Zealand, Tech. Rep. 148, 1997.
[3] M. Christodorescu, S. Jha, S. Seshia, D. Song, and R. Bryant, "Semantics-aware malware detection," in Proc. IEEE Symp. Security Privacy, May 2005, pp. 32-46.
[4] C. Kolbitsch, P. Comparetti, C. Kruegel, E. Kirda, X. Zhou, and X. Wang, "Effective and efficient malware detection at the end host," in Proc. 18th Conf. USENIX Security Symp., 2009,pp. 351–366..
[5] Y. Nadji, J. Giffin, and P. Traynor, "Automated remote repair for mobile malware," in Proc. 27th Annu. Comput. Security Appl. Conf., 2011, pp. 413-422.
[6] M. Grace, Y. Zhou, Q. Zhang, S. Zou, and X. Jiang, "RiskRanker: Scalable and accurate zero-day android malware detection," in Proc.10th Int. Conf. Mobile Syst., Appl., Services, 2012, pp. 281–294.
[7] V.Rastogi, Y.Chen, and W.Enck, "AppsPlayground: Automatic security analysis of smartphone applications," in Proc. ACM CODASPY, Feb. 2013, pp. 209–220.
[8] 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," in Proc. 19th Netw. Distrib. Syst. Security Symp., 2012, pp. 1–13.
[9] Vaibhav Rastogi, Yan Chen, and Xuxian Jiang, "Catch Me If You Can: Evaluating Android Anti-Malware against Transformation Attacks", IEEE transactions on information forensics and security, VOL. 9, NO. 1, Jan 2014
[10] M. Zheng, P. Lee, and J. Lui, "ADAM: An automatic and extensible platform to stress test Android anti-virus systems", in Proc. DIMVA, Jul. 2012, pp. 1-20.
[11] M. Christodorescu, S. Jha, and C. Kruegel, "Mining specifications of malicious behavior," in Proc. 6th Joint Meeting Eur. Softw. Eng. Conf., ACM SIGSOFT Symp. Found. Softw. Eng., 2007, pp. 5-14.
[12] H. Lockheimer. (2012, Feb.). Android and Security [Online]. Available: http://googlemobile.blogspot.com/2012/02/android-and-security.html
[13] F-Secure, Helsinki, Finland. (2012). Mobile Threat Report Q3 2012 [Online]. Available: http://www.f-secure.com/static/doc/labs_.

