Enhanced Malicious App Detection using Multi-Level Paring

1. Kaavya Rekanar, Researcher, Blekinge Institute of Technology, Sweden, kare15@student.bth.se
2. Kireet Muppavaram, *Member, IAENG,* External Research scholar, Lecturer, JNTUCEH, kireet04@gmail.com
3. Dr Meda Srinivas Rao, Professor, JNTUSIT, srmeda@gmail.com

***Abstract*—The astonishing growth of the Android platform in the last few years has given a right platform for the malware writers to introduce malware into the mobile applications. The detection of malware in the mobile applications has become a challenging and tough task and though there are few methods to detect the malware still the malware writers are finding their own ways to target different mobile applications by finding their loopholes. In this paper, we proposed a framework which is useful to detect the malicious apps by finding the loopholes in the form of permission leakages. We concentrated more towards the leakages done through permission by giving the ranking to permissions in terms of bengin and malicious list which makes the detection easier when compared with the existing methods.**

***Index Terms*—Malicious app, malware, bengin, data leakages.**

I. Introduction

MOBILE phone has always been considered as an astonishing invention by the common man. In today's world, competition has been increasing among different mobile phone makers, and also with the fast paced advancements in the hardware industry, a person is now able to have a mobile phone with large amount of memory and processing resources, that are almost equivalent to a computer, giving access to perform multi-tasking, from social network usage to document editing and sharing using different options. With very fast Internet connections, a smart phone user is now able to access the ever growing data available in web pages or email boxes very conveniently. "Mobile now represents 65 percent of digital media time, while the desktop is becoming a *secondary touch point* for an increasing number of digital users [1]".

A mobile application is a software application which is designed to run on mobile devices like smart phones and tablets. As per statistics taken from statista portal [1], 80% of the mobile users are using mobile applications. As Android operating system is open platform for making apps this is giving easy provision for many of the users to develop and run their own mobile applications. This is also giving an easy way for malware writers to find the loopholes in the mobile applications and introduce the malware by using the loopholes of applications.

Manuscript received December, 2016; revised January XX, 2017. (Write the date on which you submitted your paper for review.) This work was supported in part by the U.S. Department of Commerce under Grant BS123456 (sponsor and financial support acknowledgment goes here).

M. Shell is with the Department of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, 30332 USA e-mail: (see http://www.michaelshell.org/contact.html).

J. Doe and J. Doe are with Anonymous University.

Most of the mobile applications are attacked by using the loophole of traditional permission based system. When an application is installed by the user it asks the user to accept the permissions required by the app, most of the mobile users doesn't have technical knowledge about what happens with the acceptance of all permissions. This type of attacks generally called as permission leakage attacks. Permission leakage attacks are the most dangerous attacks which are of three types-confused deputy attack, collusion attack, capability leaks all the three types of attacks use the permission based security system loopholes and by extracting the user sensitive information.

In this paper, we concentrated more towards the detection of malicious permission required by an app which makes the release of user-sensitive data. By the detection of malicious permissions finally, we give directions to the user whether to use the app or ignore app installation. Our work is divided into following sections: Section 2 describes about the related work done on the detection of the malware attacks by different methods; Section 3 describes about our proposed framework and Section 4 describes about the experimental results and finally, we end with conclusion.

II. Related Work

Detecting malware can be done by many efforts that use static and dynamic analyses. Android devices prone to be targeted by malware can be easily detected using static analysis.

ScanDroid is one such tool which helps in detecting any kind of violations in information flow, and it can recommend if some application can be installed without violating another's permissions [12]. The major drawback of this tool is that it asks for the source codes of the applications to perform analysis [12].

CHEX is another such tool that can be used to detect any kind of leakages in Android applications. It tests the application for many common vulnerabilities [13]. But, the drawback with this tool is that it does not check the manifest that is related to the Activitys exported attribute [13].

Kantola fixes vulnerabilities using the heuristic-based approach and marks Activity classes as public by modifying the configuration semantics [14].

Drebin is a static analysis tool that can detect malware by combining permissions and APIs with machine learning techniques [15]. The model embeds features in a vector space and discovered patterns of malware from that; finally using these patterns to build the machine learning detection systems [15]. High detection accuracy can be achieved the proposed

work; this has been evaluated based in their results [15]. But, this analysis has been done on devices, for which they have to be rooted [15].

Dynamic analysis tracks runtime behaviors. Dynamic analysis monitors the application's activities, exercises targeted applications and collects the required data which is relevant to assist in the analysis of behavior during runtime.

TaintDroid tracks multiple sensitive data source in a simultaneous fashion; it is a dynamic taint tracking system [16]. It provides real-time analysis by making use of Android's virtualized execution environment [16]. This tool was tested using 30 popular third party applications; 68 misuses of private information was recorded among 20 of the applications tested [16]. Thus, this application has been successful to an extent in uncovering applications that misbehave [16].

VetDroid is another dynamic analysis platform that can reconstruct sensitive behaviors based on the usage of permissions [17]. Analysts can perceive sensitive information based on these behaviors, i.e., on how applications access resources on the platform, how these resources are being used by the applications and so on [17]. The researchers in [17] applied the framework and constructed malicious behaviors of applications to make malware analysis easier.

Machine learning and data mining techniques like Support Vector Machine, Decision Tree, Random Forest and other clustering algorithms aided in classifying the unknown nature of malware samples into existing families [18]. Researchers have improved the performance of their detection systems by combining machine learning and data mining techniques with program analysis. This method has gained immense popularity in Microsoft Windows workstations [18][20].

Permission-induced risks in Android applications using data mining approaches have been researched in [19]. In this, an analysis of individual and collaborative permissions has been performed and three ranking methods have been applied on the permission features [19]. Risky permission subsets have been identified using Sequential Forward Selection (SFS) and Principal Component Analysis (PCA) post the ranking step [19]. This approach has been evaluated using SVM, Decision Tree and Random Forest algorithms [19]. Risky permissions have achieved 94.62% detection rate with this strategy of identification; 0.6% is the FPR- false positive rate [19].

III. Proposed Method

We proposed a framework which consist of following steps, that gives the user more directions on permissions allowance to a particular app which stops the sensitive information flow through permission leakages. The proposed framework has been described in steps below:

1) Initially taking the app and extracting the Android XML file using Android Asset packaging tool.
2) By using the Proposed Binary Permission extraction algorithm extracting the binary permissions.

```
Algorithm: Binary Permission Extraction
Input: Android Manifest XML file
Output: Binary permission pattern

def remove_prefix(text, prefix):
if text.startswith(prefix):
return text[len(prefix):]
return text
file = open(filename3.txt, mode=w)
from xml.dom.minidom import parseString
data = with open(filename.txt,r) as
f: data = f.read()
dom = parseString(data)
nodes = dom.getElementsByTagName
(uses-permission)
# Iterate over all the uses-permission
nodes
for node in nodes:
perm=node.getAttribute(android:name)
perm1=remove_prefix
(perm,android.permission.)
file.write(perm1 + \n)
file.close()
f4=open(filename3.txt,mode= r)
FO=open(filename.txt,w)
for line in sorted(f4):
print(line, end=)
FO.write(line)
f.close()
```

3) To evaluate risk in the app, we used ranking methodology and classified the requested permissions in terms of malicious or bengin permissions.
Ranking of all the 135 permissions in Android has been given by applying our proposed multi-level paring to the previous malware samples collected from virusshare website. Multi-level paring (MLP) consists of

- Negative rate permission ranking
- Permission ranking through support
- Extracting relevancy of permission, and
- Applying supervised machine-learning classification methods to identify potential malware in terms of bengin or malicious.

Negative permission ranking: Negative rate permission ranking is done by calculating their rate of permission using two matrices of bengin (B) and malicious (ML_{ij})matrices which were taken from the malware samples.

$$NR(P_j) = \frac{ML_{ij} - SP_B(P_j)}{ML_{ij} + SP_B(P_j)}$$

Where $NR(P_j)$ Negative rate of permission j, ML_{ij}= jth Permission required by ith malware sample, $SPB(P_j)$- Support of jth permission in matrix B. The result of $NR(P_j)$ is in terms of [-1, 1]. If $NR(P_j)$= -1, this indicates that permission P_j is only used in benign dataset which is a low risk permission. If $NR(P_j)$=1, this indicates that permission P_j is only used in malicious dataset which is high risk permission If $NR(P_j)$= 0, this indicates that P_j is used in both bengin and malicious sample datasets. Based on this negative rate of permission we classified the permissions into malicious list, bengin list. The Permissions with rates from -1 to 0 is bengin list and the permission list with rates sorted from 1 to 0 are treated as malicious list.

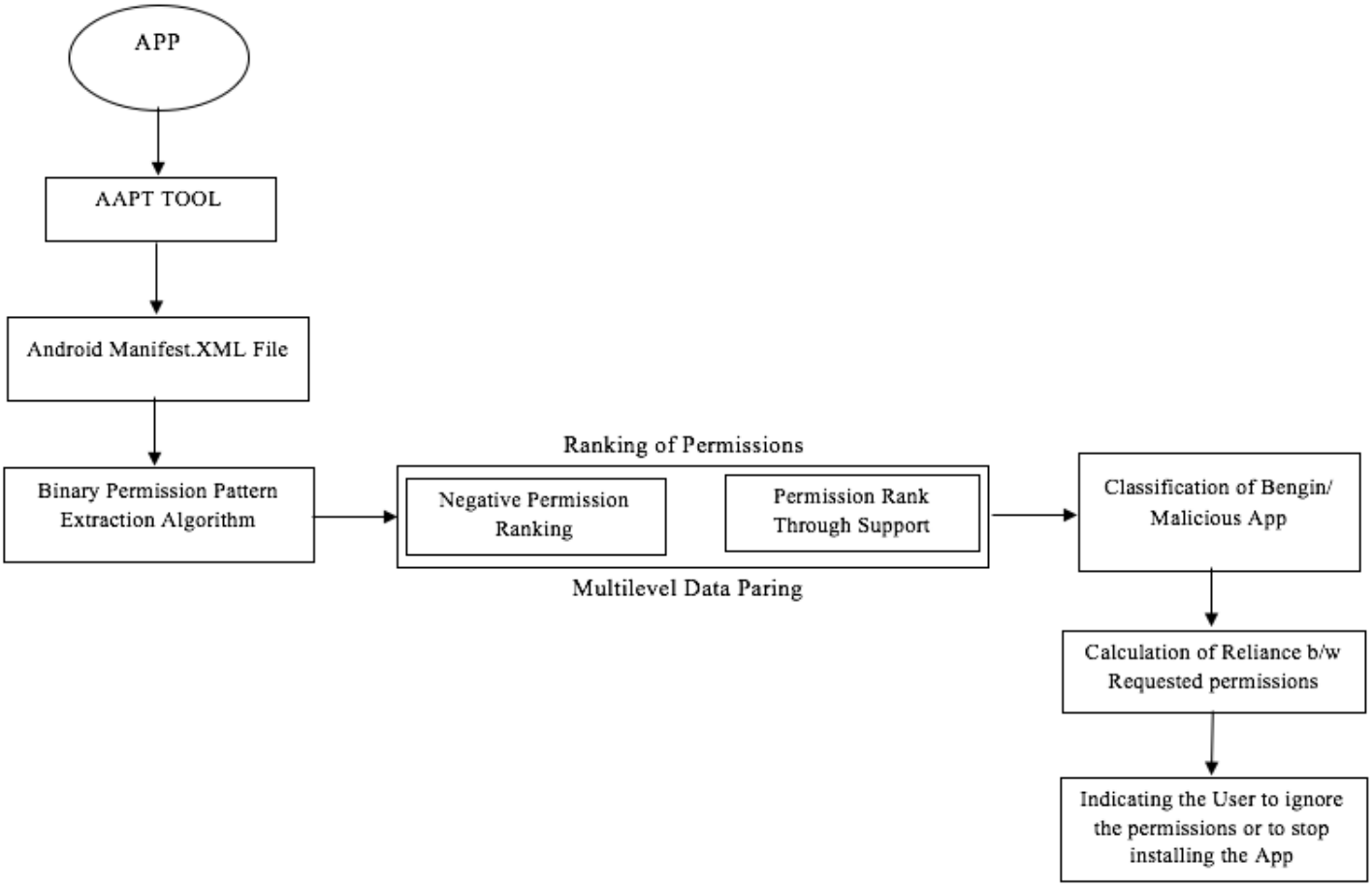

Fig. 1. Model depiction

Permission ranking through support: The main aim is to find the smallest number of permissions that can be used for malware detection in apps, so that increases the detection efficiency. To trim the no of permissions we considered the permissions that have less support should be ignored, so that reduces the detection time. For this we considered a threshold value of α if the support value of permissions in considered malware samples is less than α value the permission can be ignored.

Applying Classification Method to identify malware: To test our MLP model we used SVM that determines a hyper plane which separates both classes with maximal margin based on the training dataset that includes malicious and bengin applications. One class is associated with bengin apps and the other class is associated with malicious apps. We assumed testing data as unknown apps the classification can be done by mapping the data into vector space to decide whether the apps are on the bengin side or malicious side.

4) To further evaluate Risky permissions in the classified bengin/ malicious permissions we calculated the reliance between the requested permissions.
   We calculated the reliance between the bengin and malicious permissions for the classified 66 permissions (33 malicious 33 bengin) of previous malware samples. By using the measures lift, confidence and by applying association mining most frequently requested dependency permission patterns for two categories of apps were extracted. As dependency leads to the collusion attacks, if the app has highest ranked dependency pattern (taken from malware samples) the app is treated as malicious.
5) Finally indicating the user based on the risk permission level whether to use the app by ignoring the risky permissions or to treat the app as malicious app.

The model has been depicted in Fig1.

IV. Implementation of the Model

Considering an app from playstore which is of category education mentioned in the playstore, have downloaded the apk of the app which is by the name "air.nn.mobile.app.main.apk". By using the apk tool by using our proposed algorithm binary permission patterns were extracted from the AndroidXML manifest file of the apk. The presence of permission is indicated by "1" and the absence of permission relevant to the app is indicated by "0". The no of malicious permissions required for this app are two, number of bengin apps required is four, this is taken based on the classification table of malicious/ bengin permissions obtained from our proposed MLP process. The required malicious permissions have highest rank in the table,

they can be ignored after the installation of the app. To check the potential reliance permission patterns in the requested permissions we applied pattern matching algorithm there are is one such potential permission pattern, finally indicating the user to abort the installation of the app.

V. Results

VI. Conclusion

Malicious app detection is one of the major research challenges in the current scenario. In this paper, we proposed a framework which enhances existing detection model, and gives an easy provision for the users to find out if the app is malicious or bengin. In our work, we focused on the extraction of malicious permissions that made classification of apps easier. In future, we would like to enhance our detection model by using natural language processing techniques.

References

[1] Statista portal, http://www.statista.com/statistics/276623/number-of-apps-available-in-leading-app-stores/

[2] Cunningham, Hamish, et al. "A framework and graphical development environment for robust NLP tools and applications." ACL. 2002.

[3] Kao, Anne, and Steve R. Poteet, eds. "Natural language processing and text mining." Springer Science & Business Media, 2007.

[4] Cunningham, Hamish. "GATE, a general architecture for text engineering." Computers and the Humanities 36.2 (2002): 223-254.

[5] Larsen, Bjornar. "A trainable summarizer with knowledge acquired from robust NLP techniques." Advances in Automatic Text Summarization 71 (1999).

[6] Mani, Inderjeet. "Automatic Summarization (Natural Language Processing, 3 (Paper))." (2001).

[7] Cowie, Jim, and Wendy Lehnert. "Information extraction." Communications of the ACM 39.1 (1996): 80-91.

[8] Finkel, Jenny Rose, Trond Grenager, and Christopher Manning. "Incorporating non-local information into information extraction systems by gibbs sampling." Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics. Association for Computational Linguistics, 2005.

[9] Cunningham, Hamish, et al. "A framework and graphical development environment for robust NLP tools and applications." ACL. 2002.

[10] Elworthy, David. "Question Answering Using a Large NLP System." TREC. 2000.

[11] Tiedemann, Jrg. "Improving passage retrieval in question answering using NLP." Portuguese Conference on Artificial Intelligence. Springer Berlin Heidelberg, 2005.

[12] Fuchs, Adam P., Avik Chaudhuri, and Jeffrey S. Foster. "Scandroid: Automated security certification of android." (2009).

[13] Lu, Long, et al. "Chex: statically vetting android apps for component hijacking vulnerabilities." Proceedings of the 2012 ACM conference on Computer and communications security. ACM, 2012.

[14] Mall, Tarun, and Samarth Gupta. "Critical evaluation of security framework in Android applications: Androidlevel security and application-level security." Int. Res. J. of Comp. and Elec. Engg 2 (2014).

[15] Arp, Daniel, et al. "DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket." NDSS. 2014.

[16] Enck, William, et al. "TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones." ACM Transactions on Computer Systems (TOCS) 32.2 (2014): 5.

[17] Pravin, Ms Nigam Paridhi. "VetDroid: Analysis Using Permission for Vetting Undesirable Behaviours in Android Applications."

[18] Shahzad, Raja Muhammad Khurram. "Classification of Potentially Unwanted Programs Using Supervised Learning." (2013).

[19] Wang, Wei, et al. "Exploring permission-induced risk in Android applications for malicious application detection." IEEE Transactions on Information Forensics and Security 9.11 (2014): 1869-1882.

[20] S. Motiee, K. Hawkey, and K. Beznosov. "Do windows users follow the principle of least privilege?: investigating user account control practices." In Proceedings of the Sixth Symposium on Usable Privacy and Security. ACM, 2010.