ABSTRACT

Malicious activities on mobile devices, especially for Android devices are spreading around the world and infecting not only for end users but also for large organizations and service providers. Therefore malware analyzing and detecting methods are proposed by many researchers. Among many approaches for malware detecting, static approach is still conducting because the advantages of this approach which can reduce cost, time and risk rather than dynamic approach. The main objective of our research work is to propose a static-based malware detection framework which would improve the overall accuracy and performance of the malware detection. Although the proposed framework is described in this paper, the detail explanation and evaluation for the whole framework is not included. This framework is the classification of Android applications based on machine learning techniques. The framework rests on a combination of requested permission of the manifest files analysis and static approach. This paper only emphasizes on the classification of mobile malware according to their level of protection by using Support Vector Machine (SVM) algorithm. The experimented results of classification for malware are also discussed in this paper.

KEYWORDS
Classification, feature selection, Android, Malware, Manifest files, Mobile Security

INTRODUCTION

SMARTPHONE have been a vulnerable target for malware since June 2004. The number of infected applications steadily increased until security measures like application signing and validation of developers were introduced. Android permits application installation from third party vendors mean that Google has no control over the quality or safety of the applications provided in these stores. Several cases were encountered where legitimate applications from the Google Play Store were modified to inject malicious code and the modified applications were sold in these third party stores. It is difficult to determine whether the application is genuine or not. The
reliability of the application depends upon the security measures implemented by the application store. The attacker can easily change the code in order to incorporate the malicious code and repackages the application and publishes them in the application market. Users usually cannot differentiate between the malware application and the legitimate application and thereby end up installing by malwares.

OBJECTIVES OF THE STUDY

The main objective of our research work is to propose a static-based malware detection framework which should improve the overall accuracy and performance of the malware detection.

In this paper, we propose a malicious application detection framework on Android market to solve above problems. This framework is able to perform both detection methods using self-organizing feature map on Android market. So that the problem of static detection based on permission can be solved. Therefore, the permission based malware detection on Android application using the Android manifest and the Android asset packaging tool (aapt) to extract and decrypt the data from the Android Manifest.xml file. Android application is analyzed by using previous behaviors of malware and if it is suspicious application, it can be automatically detected.

RELATED WORKS

This section summarizes the research papers concerning with malware detection. Static approach (code-based), dynamic approach (runtime-based), and hybrid of static and dynamic approach are usually used to detect the malware in the previous research works.

Riecke et al. propose a framework for automatic analysis of malware behavior using machine learning. The framework allows identifying novel classes of malware with similar behavior (clustering) and assigning unknown malware to these discovered classes (classification). Zhou and Jiang collected and analyzed over 1200 malware samples for Android from 2010 to 2011. They found that the use of an update component that downloaded malicious code at runtime. These new techniques make it much harder to recognize an application as malware because most malware detection tools use static analysis or signature matching. If a malicious application is downloading the payload at runtime, traditional malware detection will not work. Barrera et al. use self-organizing maps to analyze permission usage patterns in applications. Felt et al. applied a type of static analysis to verify if an Android application is over-privileged or not. It examines all the permissions an application requests, and if case of not using requested permission, it concluded that the application is over-privileged. Burguera
et al. also applied the static approach to analyze the behavior of malware Android application. After analyzing, he proposed a model called Crowdroid which classifies malware or benign by applying machine learning methods. Enck et al. proposed a framework to detect potentially malicious applications based on permissions requested by Android applications. Static approach is used in their proposed framework. Enck et al. also presented a malware detecting framework called “TaintDroid” which is based on dynamic approach to monitor sensitive information on smartphones. Bläsing et al. used an Android Application Sandbox to perform Static and Dynamic analysis on Android applications. While static analysis scans Android source code to detect malware patterns, dynamic analysis executes and monitors Android applications in a totally secure environment.

PERMISSION BASED ANDROID SECURITY MODEL

Android is free and based on a 2.6 Linux Kernel. Figure 1 shows the Android Operating System. The Linux kernel resides on the lowest layer. The second layer contains the native libraries of the Android operating system. The libraries can be used freely by any of the two layers. Application framework layer contains the Java frameworks that can be accessed by application developers. Application frameworks allow sensitive information to be represented. The Android framework contains many preinstalled and provide main functions of the devices such as telephony and SMS.

![Figure 1. Architecture of Android.](image-url)
A. Android Permission Model

At the core of the Android security model is a permission-based system that provides controlled access to various system resources. The expressiveness of the permission set plays an important role in providing the right level of granularity in access control. Android has a large number of permissions restricting access to advanced functionality on devices, only a small number of these permissions are actively used by developers.

By default, Android permission system denies access to functionality or other applications that could negatively impact installed on the device. Examples of these functionalities are sending messages or making phone calls (which may cost to the user); keeping the device screen on or accessing the vibrator (which may drain the battery); and reading the user’s address book (which may violate the user’s privacy).

To make use of the restricted functionality, Android requires application developers to declare which of the restricted features are intended to be used by their application. There are 100 and over items of functionality which may be implicit (predefined permission) and explicitly (developer-defined permission). Some examples of permission to be requested the system resources in an Android application are:

<table>
<thead>
<tr>
<th>Permission</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>INTERNT</td>
<td>allows accessing the Internet</td>
</tr>
<tr>
<td>RECEIVE_SMS</td>
<td>for monitoring, recording or processing incoming SMS</td>
</tr>
<tr>
<td>RECORD_AUDIO</td>
<td>for recording audio message</td>
</tr>
<tr>
<td>READ_FRAMEBUFFER</td>
<td>directly reading the framebuffer</td>
</tr>
<tr>
<td>SET_WALLPAPER</td>
<td>Allows applications to set the wallpaper.</td>
</tr>
</tbody>
</table>

B. Android Manifest File

Every Android application composes in the form of package APK (apk) file format. After extracting the APK (apk) package, the following different types of file are also composed as shown in Figure 2. Among these files, an Android Manifest File with XML format is one of the important file. Because, the manifest file presents essential information about the application to the Android system. Permissions are declared in the manifest using the permission tag followed by a common namespace (android.permission) as described in Figure 3.
Some information contained in manifest.xml file described in Figure 3 as example.

```xml
<manifest
 xmlns:android="http://schemas.android.com/apk/res/android">
 <use-permission
 Android:name="android.permission.RECEIVE_SMS"/>
 <use-permission
 Android:name="android.permission.SENT_SMS"/>
</manifest>
```

Figure 3. Permission Information in Manifest.xml File

**C. Permission Protection Level**

Android system defines four protection levels, which characterize the potential risks implied in the permission and enforce different install-time approval processes. Permissions have associated protection levels:

*Normal:* They pose a low-risk factor and typically only affect the application’s scope. Normal permissions are granted by the system automatically without explicit approval of the user.

*Dangerous:* They are higher-risk permissions that allow costly access to services. The permissions can be granted by the user during installation. If the permission request is denied, then the application is not installed.

*Signature:* Permissions are only granted if the requesting application is signed by the same developer that defined the permission. Signature permissions are useful for restricting component access to a small set of applications trusted and controlled by the developer.
SignatureOrSystem: permissions are granted if the application meets the Signature requirement or if the application is installed in the system applications folder. Applications from the Android Market cannot be installed into the system applications folder. System applications must be pre-installed by the device manufacturer or manually installed by an advanced user.

**PROPOSED FRAMEWORK**

The first objective of our framework is to reduce the detecting and classification time for malware by introducing the features selection and extraction step in the framework. The second objective is to classify and characterize the malware by only taking the manifest file analysis in opposition to existing machine learning approach.

![PROPOSED FRAMEWORK Diagram](image-url)

**Figure 4. Proposed Malware Detecting Framework**

The cost of time and risk for detecting malware can reduce by means of static approach rather than dynamic approach. Therefore, this framework is also based on static (code-based) approach. The components in this framework are as follows:
Android Application File Accepting Component: This component is firstly responsible to accept the Android Application Package (APK) with (.apk) file format. Secondly the `manifest.xml` file is needed to extract from Android Application Package (APK).

Feature Selection Components: The output of the first component (`manifest.xml`) is used as the input of second component. After that permission information, i.e., `features of application` must be extracted from manifest file. The correlated and more meaningful features should be selected by using existing feature selection methods. (We are proposing a new method which is based on Manifest file analysis.)

Malware Detecting Component: This component is responsible to detect the inputting Android application as malware or goodware (benign) by using features getting from second component. This can be done by proposing a new method which is based on malware feature pattern mining approach versus with the existing classification approach.)

Before finishing the testing of our pattern mining approach for feature classification, the testing results on some classifiers are described in this paper.

Malware Characterization Components: This component is responsible to characterize the malware such as this malware is under the group of privacy concern, memory usage concern, system resources concern, etc. (We have to also propose a new characterization method by also using Manifest file analysis approach.)

Malware Classification Component: This component is responsible to classify the unknown malware as their types, names, security and risky level, permission level based on their training dataset. (We are proposing a new classification approach which is based on malware feature pattern mining approach versus with the existing classification approach.)

Experimental FOR CLASSIFIER

Portions of experiment for different components have been making to evaluate the performance of the whole proposed framework. This paper only describes the experimental results concerning with malware classification according to their permission levels.

Experimental Set Up: Known Android Malwares according to their permission levels (as described in (C)) are downloaded from Cantagio...
Similarly android benign applications are also downloaded from Android Market. There are totally 230 malwares and 86 benign applications are collected as the dataset. Two third (2/3) of this dataset are used as training dataset and one third (1/3) of this dataset are used as testing dataset for classification.

Feature Extraction: Manifest.xml files are extracted from two third of Android application packages (APK). Then application features (application’s permission requests to system resources) are also extracted by using the Android Asset Packaging Tool (aapt). The sample application and permission feature matrix is described in Table 1.

Table 1. Sample Application & Permission Feature Matrix.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>holycolbert.apk</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Normal</td>
</tr>
<tr>
<td>AndroidDogowar.apk</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>Signature</td>
</tr>
<tr>
<td>Lovetrap.apk</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Dangerous</td>
</tr>
<tr>
<td>BatteryDoctor.apk</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Normal</td>
</tr>
<tr>
<td>htc.apk</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>Signature</td>
</tr>
<tr>
<td>MonkeyJump.apk</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>SigOrSys</td>
</tr>
<tr>
<td>Bgserv.apk</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>SigOrSys</td>
</tr>
<tr>
<td>hippo_sample.apk</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>Normal</td>
</tr>
<tr>
<td>SPPush_1314935990854.apk</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>Dangerous</td>
</tr>
<tr>
<td>jimm.apk</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Dangerous</td>
</tr>
</tbody>
</table>
Classification Model: We have to build classification model as a part of our framework. We have employed two different algorithms for the classification: ID3 and SOM. The nature of these two algorithms are different from each other. The nature of ID3 algorithm is based on supervised learning approach. Although actual nature of SOM is based on unsupervised learning approach, SOM is employed for classification purpose in some research works. One of our objective is to evaluate the strength of SOM for classification in oppose to other classifiers. Therefore the features dataset of an android application is tested by using SOM and ID3 algorithms.

There are two main phases in classification as described in Figure 5. The first one is model learning and classifier generation phase and the second one is testing phase with unknown dataset. The classifier is generated by the first phase and classifier is used by the second phase for testing.

We build the classification model based on the training set using SOM and ID3 algorithms.

In the Self Organizing Map (SOM) method, the applied learning is an unsupervised learning where the network does not utilize the class membership of sample training, but use the information in a group of neurons to modify the local parameter. The SOM system adaptively classify samples data into classes determined by selecting the winning neurons as competitive and the weights are modified.

The algorithm on the SOM
neural network as follows:

a. If the feature vector matrix of size $k \times m$ ($k$ is the number of feature vector dimensions, and $m$ is the number of data), the initialization:
   - The number of the desired $j$ class or cluster
   - The number of component $i$ of the feature vector matrix ($k$ is the row of matrix)
   - The number of vector $X_{mi} = \text{amount of data(matrix column)}$
   - The initial weights $W_{ji}$ were randomly with interval 0 to 1
   - The initial learning rate $\alpha(0)$
   - The number of iteration ($e$ epoch)

b. Execute the first iteration until the total iteration (epoch)

c. Calculate the vector permission to start from 1 to $m$:

$$D(j) = \sum_{i} (W_{ji} - X_{mi})^2$$

For all of $j$

- Then determine the minimum value of $D(j)$
- Make changes to the $j$ weight with the minimum of $D(j)$

$$\sum_{W_{ji}} W_{ji} + \alpha (X_{mi} - W_{ij})$$

d. Modify the learning rate for the next iteration:

$$\alpha (t + 1) = 0,5 \times (t)$$

Which $t$ start from the first iteration to $e$.

e. Test the termination condition
   Iteration is stopped if the difference between $W_{ji}$ and $W_{ji}$ the previous iteration only a little or a change in weights just very small changes, then the iteration has reached convergence so that it can be stopped.

f. Use a weight of $W_{ji}$ that has been convergence to grouping feature vector for each malware, by calculating the distance vector with optimal weights.

g. Divide the malware features ($X_m$) into classes:

   If $D(1)<D(2)<D(3)<D(4)$, then the malware protection included in normal.
   If $D(2)<D(1)<D(3)<D(4)$, then the malware protection included in dangerous.
   If $D(3)<D(2)<D(1)<D(4)$, then the malware protection included in signature.
   If $D(4)<D(3)<D(2)<D(1)$, then the malware protection included in signature or system.

Thus, applications can be assigned to the nearest neuron, effectively classifying if the applications requesting similar permission into the same neighborhood. To visualize the classify structure of high dimensional weight vectors of SOM neurons, a
graphic display called U-matrix is used. Describe the process used to collect the permission datasets from the cantagio malware site and android market of the 300 applications. The experimental 45 application results will be seen from the large percentage of SOM accuracy in classifying the malware corresponding to the class by using manifest files Fig. 6, performance results were best achieved by SOM protection level, based on the analysis of the tests and experimental results of all four classifier.

**Figure 6.** Classifier of android malware protection level in SOM

Machine learning methods on mobile devices are important to select an appropriate method depending on the particular application. ID3 decision tree method is well suited to our problem of filtering large amounts of applications as it can perform relatively fast classification with low computational overhead once trained. Detecting suspicious Android applications is the ability to model both an ‘expert’ and ‘learning’ system with relative ease compared to other machine learning techniques. Moreover, this method is reducing the amount of features should be performed while preserving a high level of accuracy. ID3 depends on entropy of the attributes and selects the largest value of gain as the best feature. Figures 7 depict for each experiment, the average Accuracy and TPR, TNR results of the experiments undertaken to evaluate the classifier of ID3 and SOM. X-axis different sets of features were used containing 15, 30, 45, 60, 75, 90, 105, 120 and 135 features respectively and y-axis each of the android applications.
The testing instances are fed into the model to evaluate each classifier’s performance. True positive ratio, true negative ratio and overall model’s accuracy are measured to evaluate the model’s performance.

- **True Positive Ratio**: is the proportion of malware instances that were correctly classified.
- **True Negative Ratio**: is the proportion of benign instances that were correctly classified.
- **Overall Accuracy**: is the total number of benign and malware instances correctly classified divided by the total number of instances.

\[
TPR = \frac{TP}{TP + FN} \\
TNR = \frac{TN}{TN + FP} \\
\text{Overall Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}
\]
by the total number of the dataset instances:

**Total Accuracy:**
\[
\frac{TP+TN}{TP+FN+FP+TN}
\]

**CONCLUSION**

In this paper, we proposed a new framework to obtain and analyze the protection level of malware detection. We can use only manifest files to detect malware. Manifest files are required in Android applications, and thus, the proposed method is applicable to all Android applications. Static based malware detection framework for Android applications using the Self-Organizing Map (SOM) algorithm. Malware analysis shows that in Android, a small number of permissions are frequently used and a large number of permissions are occasionally used. Moreover, malware analysis shows correlations between several of the infrequently used permissions. Testing and training applications are then passed through behavior based module for identification of Android permission model. We realize that permission levels of Android are small number of permission levels are frequently used and a large number of permissions are occasionally used as how that it can achieve high accuracy rate.

Testing of other major applications in the Android Market, and other testing to discover additional mobile device vulnerabilities. The analysis on malicious and benign application to create a malware detection system based on unsupervised learning, capable to learn a concept of normality and therefore capable to detect malware, which has been unknown until now.

**BIBLIOGRAPHY**


Android market API.http://code.google.com/p/android-market-api/


