Permission Based Malware Protection Model for Android Application

Chit La Pyae Myo Hein

Abstract—Mobile malware performs malicious activities like stealing private information, sending sms, reading contacts and can even harm by exploiting data. Malwares are spreading around the world and infecting not only for end users but also for large organizations and service providers. Users can download/upload the APK files from third party servers and can install into their mobile devices. Most of the applications from trusted sources are not malware, but the third party server providing malwares in modified APK. In this paper, we present the permission based malware protection model for android application then use the self-organizing feature map algorithm. This is express to make small subsequent adjustments of the protection level and to improve the accuracy of the android permissions. We illustrate the effectiveness of android application permissions through extensive data analysis.

Keywords—Android, Malware, Manifest files, Mobile Security.

I. INTRODUCTION

SMARTPHONE have been a vulnerable target for malware since June 2004. The number of infected applications steadily increased until certain security measures like application signing and validation of developers was introduced. Android phones are one such Smartphone that were and continue to be a prime target for hackers. Android is an open-source mobile phone operating system with Linux-based platform. 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.

One of the very common permission is access to the Internet. Users can download the malicious software is by repackaging applications using reverse engineering tools. The attacker changes 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. 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 Asset Packaging Tool (aapt) to extract and decrypt the data from the AndroidManifest.xml file. Android Application is analyzed by using previous behaviors of malware and if it is suspicious application, it can be automatically detected.

The rest of the paper is organized as follows. Section 2 presents the related works. Section 3 explain the malware detection system framework, detailing the process of building the application to collect and give information about malware detection system. Section 4 presents the results of the malware detection testing and evaluation methods. Section 5 concludes and gives possible future work to reduce limitations of the system proposed.

II. RELATED WORKS

Two approaches have been proposed for the analysis and detection of malware: static analysis and dynamic analysis. Static analysis, mostly used by antivirus companies, is based on source code or binaries inspection looking at suspicious patterns. Although some approaches have been successful, the malware authors have developed various obfuscation techniques especially effective against static analysis. On the other hand, dynamic analysis or behavior-based detection involves running the sample in a controlled and isolated environment in order to analyze its execution traces.

Bläsing et al. [4] used an Android Application Sandbox to perform Static and Dynamic analysis on Android applications. Static analysis scans android source code to detect Malware patterns. Dynamic analysis executes and monitors Android applications in a totally secure environment.

Enck et al. presented TaintDroid in [6]. Their system used dynamic taint analysis techniques to monitor sensitive information on smartphones.

Enck et al. [7] describe the design and implementation of a framework to detect potentially malicious applications based on permissions requested by Android applications.

Burguera et al. approached Crowdroid [5] to analyze the behavior of malware Android application.

Felt et al. [8] applies 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 in case of not using requested permission, it concluded that the application is over-privileged.
Security plays a vital role in today’s mobile world. We need to provide a comprehensive security assessment in android mobile communication. Protection levels are quickly classified by static analysis detection method. Android Asset Packaging Tool (aapt) extraction tool is expected to help code analysis for static detection.

III. ANDROID PERMISSION MODEL

The permissions model is based on permissions, which are constructs that various APIs require calling apps to have before they will provide certain services. Applications must declare in their manifest file which permissions they request or require. When an application is installed, the Android system will present the various malicious applications uploaded in Google market which misuse the deficiencies in malware detection framework and the user must decide to allow the installation or not.

B. Permission Protection Level

Android permissions control the access to sensitive resources and functionalities. There are 134 Android-defined permissions are available to third party applications [12]. Permissions are defined with one of the four different protection levels, which characterize the potential risks implied in the permission and enforce different install-time approval processes. Permissions have associated protection levels:

Normal: permissions are granted automatically.

Dangerous: 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.

As well as defending protected framework, the permission mechanism is needed and should be applied in order to protect different components in an application. The protection level of the malware is identified the permission based.

C. Features

The static analysis of applications is often to classify an application as malicious or benign. In classification features are used to make informed decisions. Permissions allow an application to access potentially dangerous API functionality. Many applications require several permissions to function properly. These permissions must be listed explicitly in the application’s AndroidManifest.xml file and accepted by the user during installation as shown in Figure 1.

ACCESS_CHECKIN_PROPERTIES: Allows the read/write access to the "properties" table in the check in database, to change values that get uploaded.

BLUETOOTH: Allows applications to connect to pair Bluetooth devices.

READ_SMS: Allows an application to read SMS messages.

SET_WALLPAPER: Allows applications to set the wallpaper.

WRITE_CALL_LOG: Allows an application to write (but not read) the user's contacts data.

INTERNET: Allows applications to open network sockets.

IV. ANALYSIS OF SELF-ORGANIZATION MAP

SOM purposes to summarize complex datasets while preserving the topological properties of the input space. SOM consists of neurons, which have the same dimensionality as
the input space. The neurons are typically arranged in a rectangular or a hexagonal grid. SOM neurons can be considered as pointers in the input space, in which more neurons point to regions with high concentration of inputs. Training is competitive.

Specifically, when an input is presented, its Euclidean distance to each SOM neuron is calculated. The neuron with the minimum distance is the best matching neuron identified. The weight values of the best matching neuron and its adjacent neurons are adjusted towards the input vector. Updating neurons this way associates them with groups of patterns in the input dataset. Training is repeated for each input until the input dataset is processed several times. There are three basic steps involved in the application of the algorithm after initialization: sampling, similarity matching, and updating. These three steps are repeated until formation of the feature map has completed. The algorithm is summarized as follows:

**Initialization:** Choose random values for the initial weight vectors $W_j(0)$. The only restriction here is that for $j = 1, 2, ..., l$ where $l$ is the number of neurons in the lattice. It may be desirable to keep the magnitude of the weights small.

**Sampling:** Draw a sample $x$ from the input space with a certain probability; the vector $x$ represents the activation pattern that is applied to the lattice. The dimension of vector $x$ is equal to $m$.

**Similarity Matching:** Find the best-matching (winning) neuron $i(x)$ at time step $n$ by using the minimum-distance Euclidean criterion:

$$i(x) = \arg \min_j \| x(n) - W_j \|, \quad j = 1, 2, ..., l$$

**Updating:** Adjust the synaptic weight vectors of all neurons by using the update formula

$$W_j(n+1) = W_j(n) + \eta(n) h_{j,i(x)}(n) (x(n) - W_j(n))$$

where $\eta(n)$ is the learning-rate parameter, and $h_{j,i(x)}(n)$ is the neighborhood function $\eta(n)$ and $h_{j,i(x)}(n)$ are varied dynamically during learning for best results.

**Continuation:** Continue with step 2 until no noticeable changes in the feature map are observed.

V. PROPOSED FRAMEWORK

Malwares are evolving in a rapid manner and combat measures to stop them have become difficult because they use new signatures, encapsulation which prevents it from being detected. Anti-Virus products have been releasing daily updates which detect almost all the attacks, some of them narrowly escape. It is essential to analyze such malwares which change the registry values, tamper data, and download payloads in short which shows unusual behavior.

To detect these malwares, there are two ways which are static detection and dynamic detection to detect these malicious applications. The static detection has a small amount of resources and quick detection time but has low detection rate. On the other hand, dynamic detection has a high detection rate but uses a lot of resources and takes long time to detect malicious behaviors. In addition, it is possible to infect malicious code after implementing detection method.

To this feebleness, we propose a malware detection framework on android market and extract the Android Asset Packaging Tool (aapt) of malware features for static detection. This framework is able to perform both detection methods using SOM and is expected to perform a thorough analysis of these frameworks. Also, the proposed extraction tool is expected to help a code analysis for static detection.

![Fig. 3 The proposed malware detection framework](Image)

VI. RESULTS AND DISCUSSION

We tested our system against a collection of 300 sample Android applications. We created datasets 45 android applications by extracting features and used MATLAB 7.0.1 is used for deriving the SOM map the evaluation of the proposed framework. Normal level permission is access to the phone's vibration hardware. After it is an isolated functionality that is user's privacy or other applications cannot be compromised, it is not considered a major security risk. Dangerous permission can use telephony services, network access, and location information or gain other private user data. After dangerous permissions present a high security risk, the user is promoted to confirm them before the installation. Signature permission is perform to the read history book mark for accessing personal info (contacts and calendar) in effect a refinement of the shared user ID approach and provides more control in sharing application data and components. SignatureOrSystem permission extends the signature permission by granting access checkin properties, private only signature. Malware protection model is required permissions will grant access rights without asking for the user's explicit approval.So U-
matrix representation of the SOM for Android permissions for each of the level as shown in below.

Fig. 4 U-matrix representation of SOM for Normal, Dangerous, Signature and SignatureOrSystem protection level of the permission

Permissions are the most recognizable security feature in Android. We used the Android Asset Packaging Tool (aapt) to extract and decrypt the data from the AndroidManifest.xml file, provided by the Android SDK

Fig. 5 Performance of permission-based malware protection model

http://dx.doi.org/10.15242/IIE.E0314102
We extract the malware feature from the android manifest file and the permission request features from each application. We build a dataset in a .data format with the extracted data and then the classification level of the SOM for detecting android malware analysis and enable us to utilize a suitable distance metric to express similarities between applications. In this protection level, we evaluate the capacity of permissions to detect malware by using true and false positive ratio and then by finding out total accuracy:

1) True Positive Ratio (TPR) : (TP/ TP+FN)
2) False Positive Ratio (FPR) : (FP / FP+FN)
3) Total Accuracy: (TP+TN / TP+FN+FP+TN)

VII. CONCLUSION

In this paper, we proposed the 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 all Android applications, and thus, the proposed method is applicable to all Android applications. Empirically analyzing permission-based models and using the Self-Organizing Map (SOM) algorithm. Testing and training applications are then passed through the behavior based module for identification of android permission model. We realize that permission levels of android a small number permissions are very frequently used and a large number of permissions are only occasionally used and show that it can achieve high accuracy rate.

REFERENCES


