Final Report: Android Malware Detection

Megha Katwala, Matthew Chan

May 1, 2017

1 Introduction

With an estimated market share of 70% to 80%, Android has become the most popular operating system for smartphones and tablets. Unsurprisingly, cyber-criminals have followed, expanding their malicious activities to mobile platforms.

Mobile threat researchers have recognized an alarming increase of Android malware from 2012 to 2013 and estimate that the number of detected malicious applications is in the range of 120,000 to 718,000. To efficiently detect malware from applications available from official and third-party sources, many efforts have contributed to studying the nature of smartphone platforms and their applications in the past decade.

The Android platform employs the permission system to restrict applications privileges to secure the sensitive resources of the users. The developer is responsible for determining appropriately which permissions an application requires, but an application needs to get a user's approval of the requested permissions to access private or otherwise-restricted resources. Although the permission system can protect users from applications with invasive behaviors, its effectiveness highly depends on a user's comprehension of the consequences of granting a permission.

According to recent studies, many users do not understand what each permission means and blindly grant them, potentially allowing an application to access sensitive/private information. Another flaw is that the user cannot decide to grant single permissions, while denying others. Many users, although an app might request a suspicious permission among many seemingly legitimate permissions, will still confirm the installation.

The Android security model is based mainly on permissions. As a result, the implementation of these permissions is of interest to us. An Android permission is a restriction limiting access to a part of the code or to data on the device. The limitation is imposed to protect critical data and code that could be misused to distort or damage a user's experience. Permissions are also used to allow or restrict application access to restricted APIs and resources. For example, the Android 'INTERNET' permission is required by apps to perform network communications; so, opening a network connection is restricted by the 'INTERNET' permission. Furthermore, an application must have the 'READ CONTACTS' permission in order to read entries in a user's phonebook as well. To require a permission, the developer specifies them using the Manifest file in declaring a "<uses-permission>" attribute. The "android:name" field specifies the name of the permission in the code. A permission can be associated with one of the following four protection levels:

- Normal: A low-risk permission which allows applications to access API calls ('SET WALLPAPER') causing no harm to users.
- Dangerous: A high-risk permission which allows applications to access potential harmful API calls('READ CONTACTS') such as leaking private user data or control over smartphone device.Dangerous permissions are explicitly shown to the user before an app is installed and the user must decide to grant the permissions or not, determining whether the installation continues or fails,respectively.
- Signature: A permission which is granted if its requesting application is signed with the same certificate as the application which defines the permission is signed.
- Signature-or-system: A permission which is granted only if its requesting application is in the same Android system image or is signed with the same certificate as the application which defines the permission is signed.

2 Literature Review

Felt et al. [1] evaluated whether permissions of an application are effective at protecting users after collecting the permission requirements of a large set of Google Chrome extension and Android applications. Their results indicated that application permissions can have a positive impact on system security when applications permission requirements are declared upfront by the developer, but can be improved. However, this study shows that users are frequently presented with requests for dangerous permissions during application installation in install time systems. As a consequence, installation security warnings may not be an effective malware prevention tool, even for alerting users. Sarma et al. [2] investigate the feasibility of using both the permissions requested by an app, the category of the application and what permissions are requested by other applications in the same category to better inform users whether the risks of installing an app is commensurate with its expected benefit. He proposes a multi-criteria evaluation of Android applications, to help the user to easily understand the trustworthiness degree of an application, both from a security and a functional side.

Malicious applications are those that have access capabilities to sensible data and transmission capabilities as well, at the same time deceiving the users, by pretending to offer services that typically do not require such capabilities, or to make a legitimate use of them. The methodology relies on the comparison of the Android security permission of each application, with a set of reference models, for applications that manage sensitive data.

Another way is, given a dataset of applications, classifying the malicious(Bad) and the benign(Good) ones. The techniques are sub-grouped in approaches based on permission individually, combination of permissions, using machine learning techniques.

3 Methodology

3.1 Permission

Zhou and Jiang [3] characterize existing Android malware from various aspects, including the permissions requested. They identified individually the permissions that are widely requested in both malicious and benign apps. According to this work, malicious apps clearly tend to request more frequently on the SMS-related permissions, such as 'READ SMS', 'WRITE SMS', 'RECEIVE SMS', and 'SEND SMS'. They found that malicious apps tend to request more permissions than benign ones.

They found no strong correlation between applications categories and requested permissions, and introduce a method to visualize permissions usage in different app categories. The aim of their work is to classify Android applications into several categories such as entertainment, society, tools, and productivity, multimedia and video, communication, puzzle and brain games. Mentions a method that analyses manifest files in Android application by extracting four types of keyword lists:(1) Permission, (2) Intent filter (action), (3) Intent filter (category), and (4) Process name. This approach determines the malignancy score by classifying individually permissions as malicious or benign.

3.2 Combination of Permissions

Rassameeroj and Tanahashi [4] performed a high level contextual analysis and an exploration of Android applications based on their implementation of permission-based security models by applying network visualization techniques and clustering algorithms. From that, they discovered new potentials in permission-based security models that may provide additional security to the users. This method on network classification helps to define irregular permission combinations requested by abnormal applications. The nature, sources and implications of sensitive data on Android devices in enterprise settings. They characterized malicious apps and the risks they pose to enterprises. Finally, they have proposed several approaches for dealing with security risks for enterprise. From the analysis of third-party applications, Permission additions dominate the evolution of third-party apps, of which Dangerous permissions tend to account for most of the changes.

Canfora et al. [5] propose a method for detecting malware based on three metrics, which evaluate: the occurrences of a specific subset of system calls,a weighted sum of a subset of permissions

that the application required, and a set of combinations of permissions.

3.3 Using Machine Learning Techniques

Sanz et al. [6] introduce a new method to detect malicious Android applications through machine learning techniques by analyzing the extracted permissions from the application itself. Features used to classify are the presence of tags uses-permission and uses-feature into the manifest as well as the number of permissions of each application. These features are the permission requested individually and the «uses-feature» tag. Huang et al. [7] explore the possibility of detection malicious Android applications based on permissions and 20 features from Android application packages.

According to them, by combining results from various classifiers, it can be a quick filter to identify more suspicious applications. And propose a framework that intends to develop a machine learning-based malware detection system on Android to detect malware applications and to enhance security and privacy of smartphone users. This system monitors various permission-based features and events obtained from the android applications, and analyses these features by using machine learning classifiers to classify whether the application is benign or malware. Once, the Support Vector Machine trained offline on a dedicated system and only it is transferred the learned model to the smartphone for detecting malicious applications.

4 Design and Experimental Setup

Our project involves using machine learning techniques to detect Android malware. We utilize decision tree classifiers to predict the likelihood that an application is malware based on only its permissions. We implement and test this machine learning approach by developing our own Android application capable of providing antivirus-like features, scanning applications for their permissions and assessing the danger it may pose.

To perform this assessment, we use a decision tree classifier. The choice to use a decision tree classifier is mainly due to our approach of using only application permissions. The decision tree is an intuitive and flexible way to analyze the risks posed by certain permissions, as well as the patterns of permissions that may indicate malicious behavior. This white box approach allowed us to learn more about the underlying predictions.

4.1 Background

In implementing our project, we attempt to independently implement and verify the work of Aung and Zaw[11], who discuss in their paper "Permission-Based Android Malware Detection" their work and results in using machine learning algorithms to detect malware in Android applications.

We attempt our own implementation of permission-based Android malware detection, and compare our results with the ones they discuss in their paper. Although they discuss the theory behind their work, they did not provide the datasets that they used or discuss specific implementation details. Therefore the implementation and results that we discuss are our own.

4.2 Tools

In designing and implementing this project, we used several tools which we discuss below.

4.2.1 Android Studio

To implement our Android malware detection, we decided to use an Android application, so that users could install our application similar to an anti-virus application. For this we used Android Studio to code our application. This allowed us to present an easy-to-use interface to the user, and also access pertinent parts of the Android device that would allow us to access application permissions and listen for important events like application installation or updates.

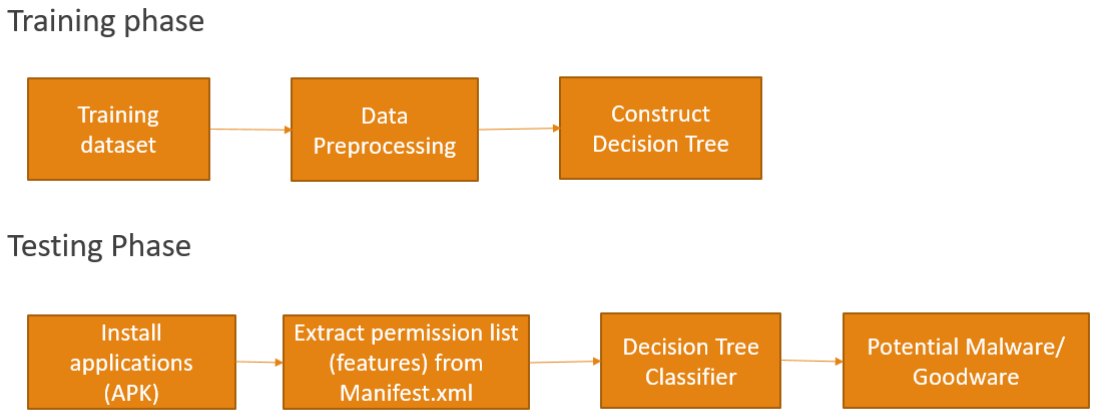
4.2.2 Python

Although Android applications are coded in Java, we used Python to implement our machine learning algorithms due to the well-known `scikit-learn` library, which allowed us a large amount

of flexibility in designing and tuning our algorithms. After growing the regression tree, we exported it to a Java function to import into the Android application.

4.3 Experimental Design

Our experimental design is made up of two main steps, which can be broken down further. These two portions, corresponding to the building of the decision tree and the development of the mobile application, can be seen below.



4.3.1 Training Phase

First, we acquired a suitable dataset of Android permissions to begin building our classifier. We use a dataset provided by Urcuqui and Navarro in their research presented at COLCOM 2016 [12]. The dataset consists of 398 application samples, with over 300 permissions listed. Of the 398 samples, half were labeled as malware, and the other half were benign applications (199 each). To begin, we preprocessed the data, removing permissions that none of the samples used (which therefore contained no information). This pruning step lowered the number of features to 92.

Next, the pruned dataset was used to grow a classification tree. A decision tree classifier chooses one feature at every step to split on, and divides the dataset based on that feature. It continues this process recursively until a constraint has been reached, or the dataset is perfectly classified. Although perfectly classifying the data seems like a good idea, doing this can be seen as over-training, as the tree produced may not generalize well.

The way that the decision tree algorithm chooses which feature to split on is one of several metrics, the most popular being Information Gain. A concept from Information Theory, information gain is the decrease in entropy (or amount of information gained) by splitting on that feature. By choosing the feature that best splits the data correctly, a fairly simple tree can be grown.

Once our decision tree had been grown, we measured its accuracy and exported the tree to the Android application.

4.3.2 Testing Phase

Our Android application was installed on a physical device. This allowed us to test our implementation as well as implement features that would allow the application to act similar to an antivirus. The application does several things: scan the phone's installed applications, display results that exceeded the prediction threshold, give a diagnostic report of all the applications on the user's phone, and allow the user to change permissions after viewing. Additionally, the app will ask the user to scan applications after installs and updates. The permissions of each application are stored in a list, which is converted into a binary feature vector that can be input into the decision tree classifier. The application performs this classification for each app on the phone, and returns the results to the user.

5 Results

First and foremost, we were able to implement much of what was discussed in the experimental setup phase. We were able to create the decision tree classifier, implement the Android application, and implement the classifier on the Android application.

Below is a listing of the decision tree created, showing the learning process, and the final classifier's nodes and their relative importance in the tree structure. Along with that is an example of the function that was exported to the Android application. While the function was automatically generated using Python, at this time it still needs to be put into the application manually through Android Studio. Though this is mainly due to the difficulty of implementing a decision tree regressor in Java, future iterations of our application will possibly allow a decision tree model to be read in through a file or through other means.


```
(0, 'android.permission.READ_PHONE_STATE', 0.90074546529970889)
(1, 'android.permission.WRITE_SMS', 0.049600714250285172)
(2, 'android.permission.RECEIVE_BOOT_COMPLETED', 0.02008252174645269)
(3, 'android.permission.WAKE_LOCK', 0.01317264059801526)
(4, 'android.permission.READ_CONTACTS', 0.0094998241020324125)
(5, 'android.permission.READ_EXTERNAL_STORAGE', 0.0018600267843857053)
(6, 'android.permission.ACCESS_NETWORK_STATE', 0.0014634034259505134)
(7, 'android.permission.CHANGE_WIFI_STATE', 0.0013285905602755024)
(8, 'android.permission.VIBRATE', 0.0010730923756071447)
(9, 'android.permission.ACCESS_FINE_LOCATION', 0.00029665498953519471)
(10, 'android.permission.READ_SMS', 0.00029598979570685829)
(11, 'android.permission.INTERNET', 0.00025869338035708399)
(12, 'android.permission.ACCESS_WIFI_STATE', 0.00022766545708515133)
(13, 'android.permission.WRITE_EXTERNAL_STORAGE', 9.4717234602469024e-05)
```

```
// Automatically generated decision tree function
public double tree(int[] feature_array){
    if(feature_array[54] <= 0.5)
        if(feature_array[86] <= 0.5)
            if(feature_array[74] <= 0.5)
                if(feature_array[40] <= 0.5)
                    if(feature_array[80] <= 0.5)
                        if(feature_array[51] <= 0.5)
                            if(feature_array[13] <= 0.5)
                                return 0.0238095238095;
                            else
                                return 0.0;
                        else
                            return 0.0;
                    else
                        return 0.0;
                else
                    if(feature_array[59] <= 0.5)
                        if(feature_array[8] <= 0.5)
```

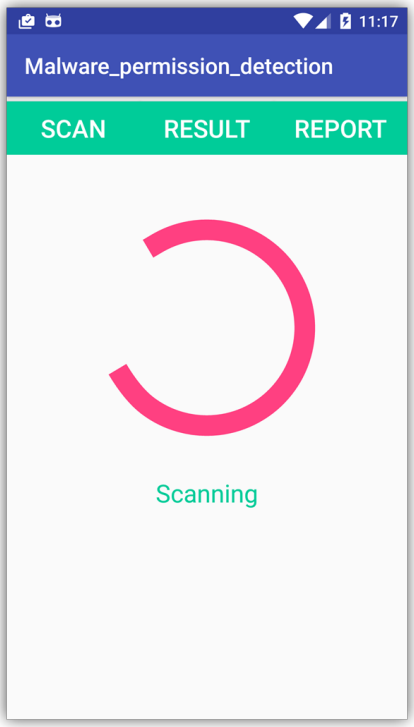
```
                        if(feature_array[10] <= 0.5)
                            if(feature_array[4] <= 0.5)
                                if(feature_array[46] <= 0.5)
                                    if(feature_array[7] <= 0.5)
                                        if(feature_array[80] <= 0.5)
                                            return 0.166666666667;
                                        else
                                            return 0.0;
                                    else
                                        if(feature_array[80] <= 0.5)
                                            return 0.0;
                                        else
                                            return 0.111111111111;
                                else
                                    return 0.0;
                            else
                                return 0.0;
                        else
                            return 0.0;
                    else
                        return 0.0;
                else
                    return 0.0;
            else
                if(feature_array[7] <= 0.5)
                    return 0.0;
                else
                    return 0.2;
        else
            return 0.5;
    else
        if(feature_array[75] <= 0.5)
            if(feature_array[55] <= 0.5)
                if(feature_array[23] <= 0.5)
                    if(feature_array[7] <= 0.5)
                        return 1.0;
                    else
                        return 0.75;
                else
                    return 1.0;
            else
                return 1.0;
        else
            if(feature_array[83] <= 0.5)
                if(feature_array[5] <= 0.5)
                    if(feature_array[51] <= 0.5)
                        if(feature_array[7] <= 0.5)
                            return 0.25;
                        else
                            return 0.6;
                    else
                        return 0.0;
                else
                    return 1.0;
            else
                return 1.0;
}
```

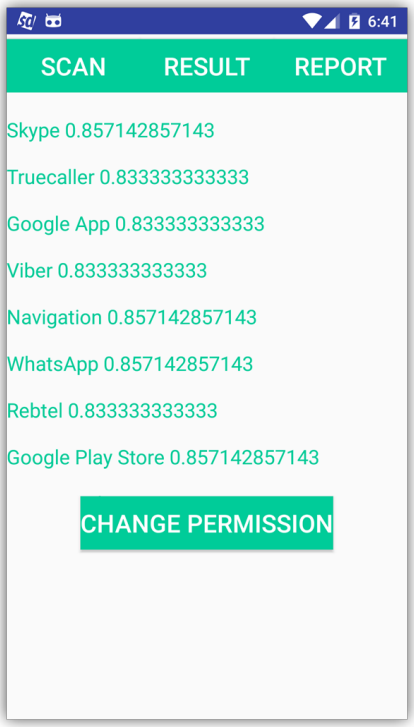
We collected data comparing the true and false positive rate of our classifier, shown below.

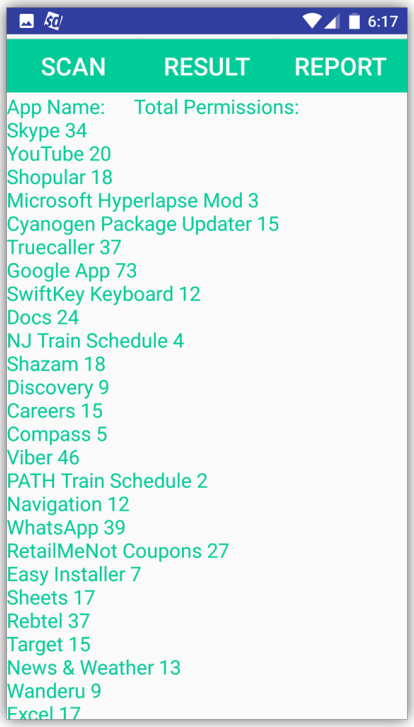
Actual \ Prediction	Malicious	Benign
Malicious	TP:185 (93.0%)	FN:16 (8.0%)
Benign	FP:14 (7.0%)	TN:183 (92.0%)

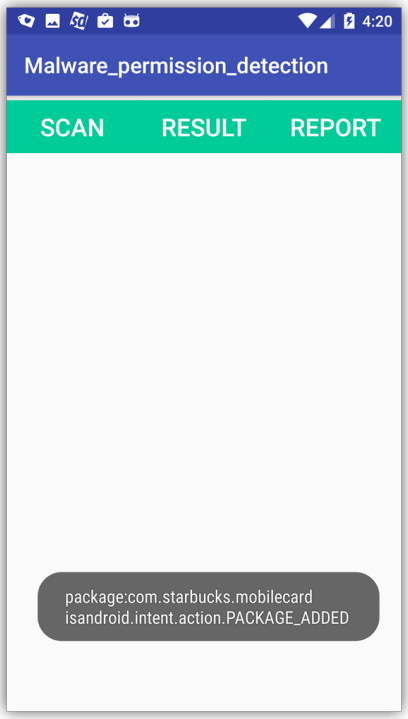
True Positive Rate (TP/(TP+FN)) = 92% (compared to 97.8%)
False Positive Rate (FP/(TN+FP)) = 7% (compared to 15.7%)
Overall Accuracy (% of correctly identified) = 92.5%

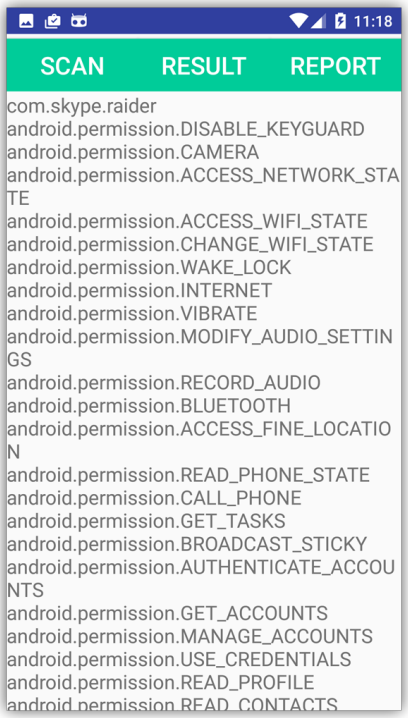
Below are images of the Android application.




6:41
SCAN
RESULT
REPORT
Skype 0.857142857143
Truecaller 0.833333333333
Google App 0.833333333333
Viber 0.833333333333
Navigation 0.857142857143
WhatsApp 0.857142857143
Rebtel 0.833333333333
Google Play Store 0.857142857143
CHANGE PERMISSION


6:17
SCAN
RESULT
REPORT
App Name: Total Permissions:
Skype 34
YouTube 20
Shopular 18
Microsoft Hyperlapse Mod 3
Cyanogen Package Updater 15
Truecaller 37
Google App 73
SwiftKey Keyboard 12
Docs 24
NJ Train Schedule 4
Shazam 18
Discovery 9
Careers 15
Compass 5
Viber 46
PATH Train Schedule 2
Navigation 12
WhatsApp 39
RetailMeNot Coupons 27
Easy Installer 7
Sheets 17
Rebtel 37
Target 15
News & Weather 13
Wanderu 9
Excel 17





6 Conclusion and Future Work

In conclusion, our project can identify, with moderate success, applications that pose a potential threat based on the permissions that they request. Our application can scan applications on a phone at any time, and alerts the user to do so when an installation or app update occurs.

We believe that this is an important step in preventing Android malware, because this application brings to the user's attention all the possibly dangerous applications, allowing them to scrutinize the applications that they trust more carefully. This in turn will help users become more security-conscious overall.

Even so, this is only a first step. Future work for this project will include increasing the accuracy of the classifier, migrating the Python portions of this project to Java, and integrating more advanced methods of detecting malicious behavior such as looking at API calls (this follows a "defense in depth" strategy). One benefit of the decision tree classifier is its speed. It can serve as a preliminary screen for more advanced but slower methods, to focus the applications they will inspect. Lastly, taking into account application categories such as being a game or email-client would also help detect suspicious permissions and behaviors.

7 References

1. A. P. Felt, K. Greenwood, and D. Wagner, “The effectiveness of install-time permission systems for third-party applications”,2010.
2. B. P. Sarma, N. Li, C. Gates, R. Potharaju, C. Nita-Rotaru, and I. Molloy, “Android permissions: a perspective combining risks and benefits,” 2012.
3. Y. Zhou and X. Jiang, “Dissecting android malware: Characterization and evolution,2012.
4. V. Rastogi, Y. Chen, and X. Jiang, “Droidchameleon: evaluating android anti-malware against transformation attacks, 2013.
5. G. Canfora, F. Mercaldo, and C. A. Visaggio, “A classifier of malicious android applications,” 2013.
6. B. Sanz, I. Santos, C. Laorden, X. Ugarte-Pedrero, P. G. Bringas, and G. A´ lvarez, “Puma: Permission usage to detect malware in android,” ,2013.
7. C.-Y. Huang, Y.-T. Tsai, and C.-H. Hsu, “Performance Evaluation on Permission-Based Detection for Android Malware,” 2013.
8. Franklin Tchakount´, Computers & Security “Permission-based Malware Detection Mechanisms on Android: Analysis and Perspectives”, 2014.
9. Z. Fang, W. Han, and Y. Li, “Permission-based Android security: Issues and countermeasures,”
10. W. Enck, M. Ongtang, P. McDaniel, "Understanding Android Security"
11. Aung, Zarni, and Win Zaw. "Permission-based android malware detection."International Journal of Scientific and Technology Research2.3 (2013): 228-234.
12. Urcuqui, C., and A. Cadavid. "Machine learning classifiers for Android malware analysis." Proceedings of the IEEE Colombian Conference on Communications and Computing. 2016.