Results Report

Megha Katwala, Matthew Chan

May 1, 2017

1 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.

1.1 Background

In implementing our project, we attempt to independently implement and verify the work of Aung and Zaw[1], 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.

1.2 Tools

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

1.2.1 Android Studio

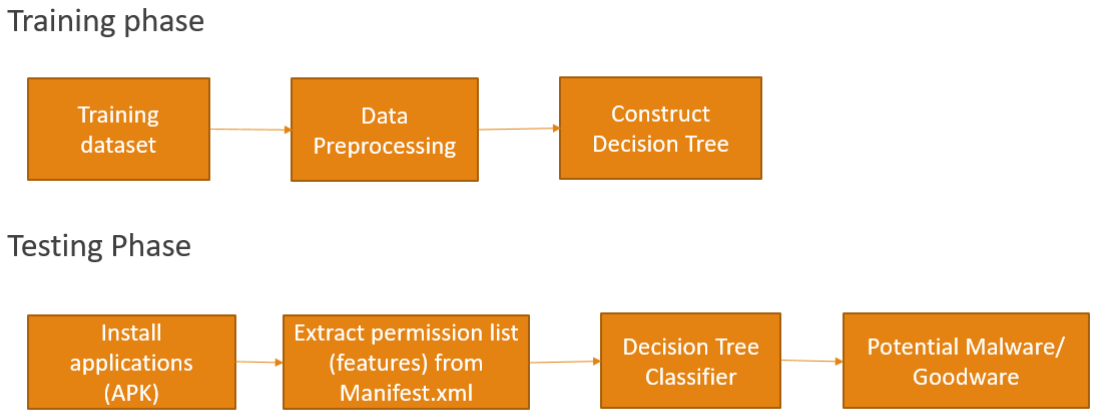
To implement our Android malware detection, we decided to use an Android application, so that users could install our application similar to an anti-virus application. For this we used Android Studio to code our application. This allowed us to present an easy-to-use interface to the user, and also access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

1.2.2 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.

1.3 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.



1.3.1 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 [2]. 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.

1.3.2 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.

2 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 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

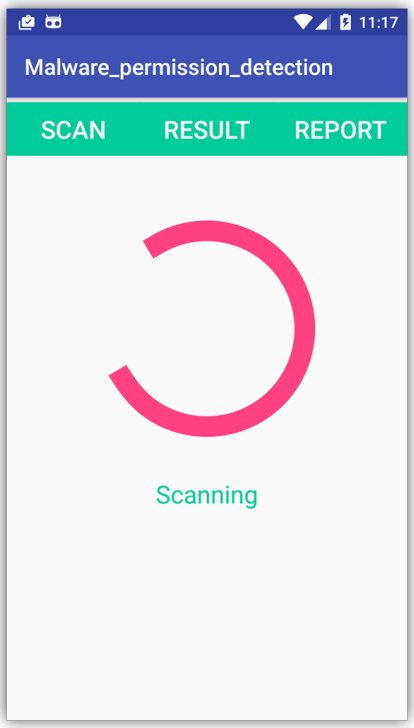
our application will possibly allow a decision tree model to be read in through a file or through other means.

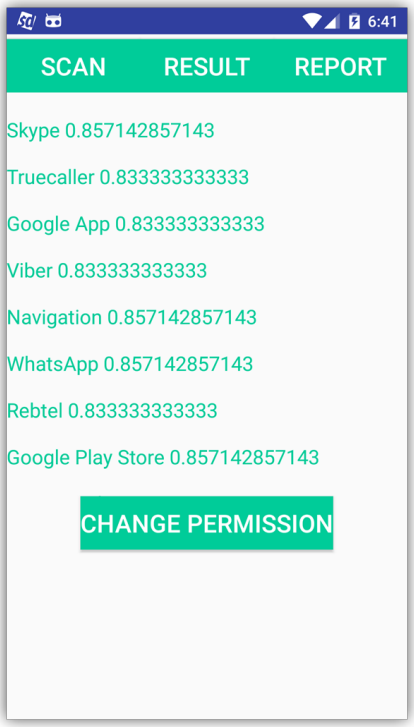
```
(0, 'android.permission.READ_PHONE_STATE', 0.90074546529970889)
(1, 'android.permission.WRITE_SMS', 0.049600714250285172)
(2, 'android.permission.RECEIVE_BOOT_COMPLETED', 0.02008252174645269)
(3, 'android.permission.WAKE_LOCK', 0.01317264059801526)
(4, 'android.permission.READ_CONTACTS', 0.0094998241020324125)
(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)
```

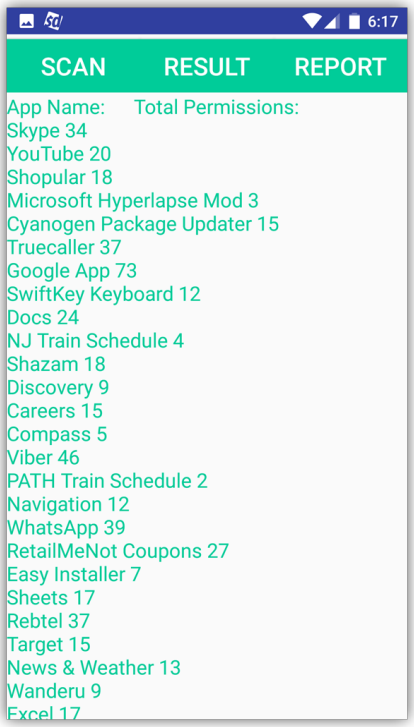
```
// Automatically generated decision tree function
public double tree(int[] feature_array){
    if(feature_array[54] <= 0.5)
        if(feature_array[86] <= 0.5)
            if(feature_array[74] <= 0.5)
                if(feature_array[40] <= 0.5)
                    if(feature_array[80] <= 0.5)
                        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
```

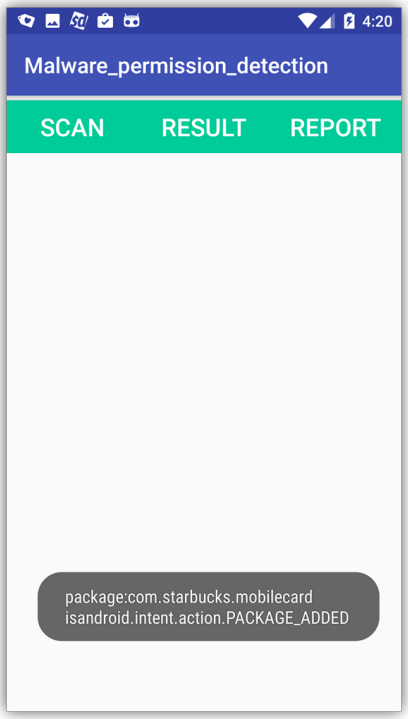
```
            return 0.5;
    else
        if(feature_array[75] <= 0.5)
            if(feature_array[55] <= 0.5)
                if(feature_array[23] <= 0.5)
                    if(feature_array[7] <= 0.5)
                        return 1.0;
                    else
                        return 0.75;
                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;
}
```

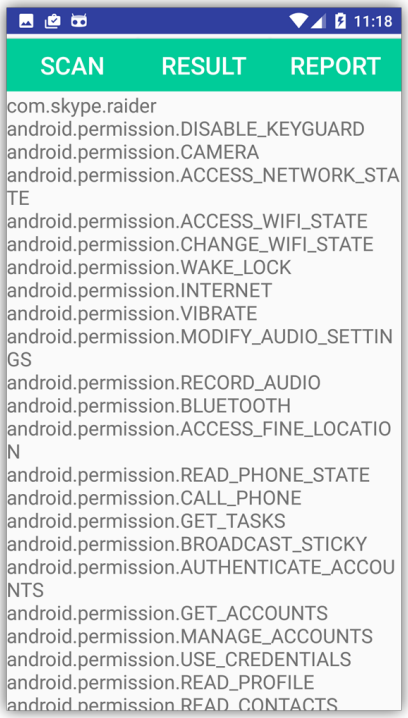
Below are images of the Android application.




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





3 References

1. Aung, Zarni, and Win Zaw. "Permission-based android malware detection."International Journal of Scientific and Technology Research2.3 (2013): 228-234.

2. Urcuqui, C., and A. Cadavid. "Machine learning classifiers for Android malware analysis." Proceedings of the IEEE Colombian Conference on Communications and Computing. 2016.